CS6320: 3D Computer Vision
Project 4
Optical Flow and Structured Light

Arthur Coste: *coste.arthur@gmail.com*

March 2013

Contents

1 Introduction

In this fourth project we are going to discuss and analyse Optical Flow techniques and Structured light.

In this report, we will first discuss some theoretical problems related to Optical Flow to allow us to introduce the practical and also some theoretical questions about Structured Light. Then we will present our implementation, results and discussion about Optical Flow. Optical flow will be presented with two methods the Lucas and Kanade method and the Horn and Schunck.

In this project, the implementation is made using MATLAB, the functions we developed are included with this report and their implementation presented in this document.

The following functions are associated with this work :

- $optical\_flow.m$: $optical\_flow()$ Stand alone function
- $separable\_gaussian.m$: $OutputIm = separable\_gaussian(InputImage, sigma, display)$
- $convol\_separable.m$: $OutputIm = convol\_separable(InputImage, 1Dkernel\_1, 1Dkernel\_2, display)$

Warning : Our implementation present the three methods : Lucas and Kanade, Local Weighted Neighbourhood Lucas and Kanade, Horn and Schunck method in the same program so according to the size of the image it could be time consuming and can take up to several minutes to generate the 3 resulting displacement field.

Note : All the images and drawings were realized for this project so there is no Copyright infringement in this work.

2 Theoretical Problems

2.1 Structured Light

In this section we are going to discuss structured light which is a technique used to reconstruct depth and object geometry using special types of illumination. This technique intends to improve the matching complexity that could sometimes occur with shape from multiple views techniques. Here is a drawing that present the practical set up for such a system:

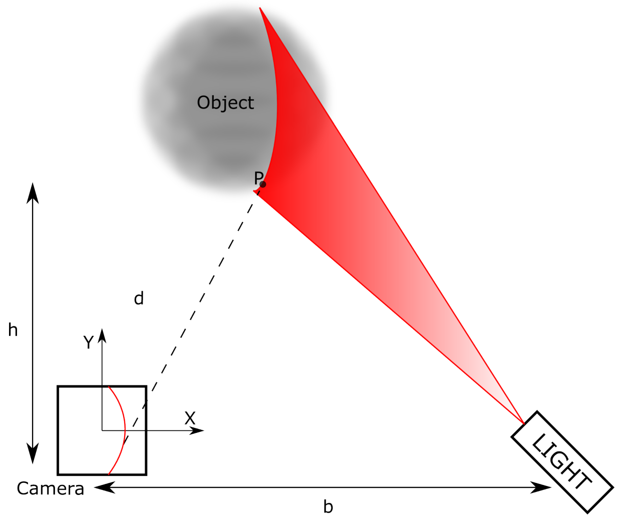

Figure 1: Illustration of structured light set up

2.1.1 2D Triangulation

In this section we are going to focus on the 2D set up of the problem and we will try to derive a relationship for one of the parameters. Based on the previous schematic presentation of the set up of the system, we can create a geometrical bi-dimensional representation as follow:

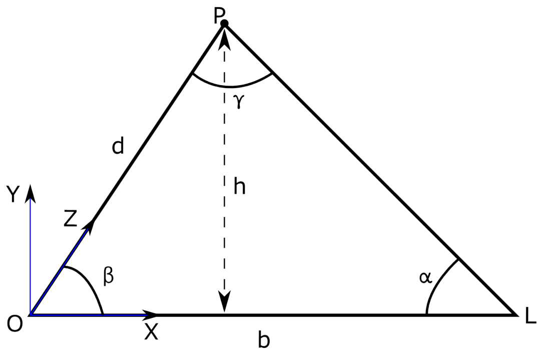

Figure 2: Bi-dimensional geometry model

General trigonometry and geometric relations in triangle provides the following equation :

$$\frac{d}{sin(\alpha)} = \frac{b}{\sin(\gamma)} = \frac{\bar{P}\bar{L}}{sin(\beta)} \tag{1}$$

Where $\bar{P}\bar{L}$ is the algebraic distance between points P and L. Then another very classical rule of Euclidean geometry on triangles gives that the sum of the angles of a triangle add up to π. So :

$$\alpha + \beta + \gamma = \pi \qquad \Rightarrow \qquad \pi - \gamma = \alpha + \beta \tag{2}$$

A final trigonometry rule : $sin(\pi - \gamma) = sin(\gamma)$ will allow us to get the distance:

$$\frac{d}{sin(\alpha)} = \frac{b}{\sin(\alpha+\beta)} \qquad \Rightarrow \qquad d = b\frac{sin(\alpha)}{sin(sin(\alpha+\beta)} \tag{3}$$

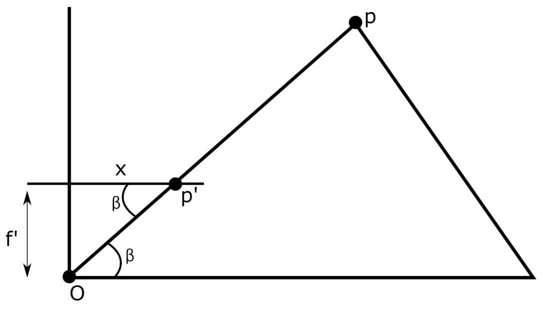

Figure 3: Projection of point P in the virtual image plane

so using trigonometry in the triangle we have:

$$tan\beta = \frac{f'}{x} \qquad \Rightarrow \qquad \beta = arctan\left(\frac{f'}{x}\right) \tag{4}$$

we can also derive another equation using the cotangent :

$$cot\beta = \frac{x}{f'} \qquad \Rightarrow \qquad \beta = arccot\left(\frac{x}{f'}\right) \tag{5}$$

2.1.2 3D Triangulation

As we presented previously in the bi dimensional case, we did not consider the angle γ between the light source and the camera sensor at the object point P. In this section we are going to show that this angle can be used in the three dimensional triangulation to verify the reconstruction result.

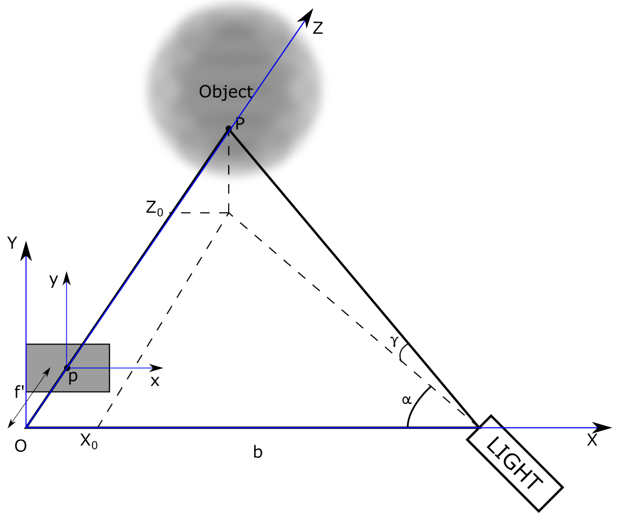

Figure 4: Three-dimensional geometry model

We have the following relation between length and projections of P on the images plane in $p' = (x', y')$ (Klette):

$$\frac{X_0}{x'} = \frac{Y_0}{y'} = \frac{Z_0}{f'} \tag{6}$$

$$tan(\alpha) = \frac{Z_0}{B - X_0} \tag{7}$$

So if we use the previous equations, we can then compute the expressions of the spatial coordinates of the point $P = (X_0, Y_0, Z_0)$ of the object.

$$Z_0 = (B - X_0)tan(\alpha) = \frac{f'X_0}{x'} \qquad \Rightarrow \qquad X_0 = \frac{Bx'tan(\alpha)}{f' + x'tan(\alpha)} \tag{8}$$

Using the same equations and derivation we end up with the equations to compute the spatial position of point P:

$$X_0 = \frac{Bx'tan(\alpha)}{f' + x'tan(\alpha)} \quad , \quad Y_0 = \frac{By'tan(\alpha)}{f' + x'tan(\alpha)} \quad \text{and} \quad Z_0 = \frac{Bf'tan(\alpha)}{f' + x'tan(\alpha)} \tag{9}$$

In the previous equations, we can notice that the reconstruction of depth and position of the object in the real world space. So now we are going to discuss how using the scanning angle γ can be used to verify the accuracy of the results obtained with the previous equations. Indeed, the scanning angle γ is a given an known parameter chosen to perform analysis.

Assuming γ is known, we can in the projection in the YZ plane obtain the following relationship:

$$tan(\gamma) = \frac{Z_0}{Y_0} \tag{10}$$

This previous relationship is exactly the same as the one we used to derive our equations to determine the position of $P = (X_0, Y_0, Z_0)$. So by knowing the value of γ we can at each step verify the consistency of this determined position by comparing the previous ratio of coordinates to the tangent of the angle γ.

2.2 Motion and Optical Flow

Let's consider a rectangle which presents two different kinds of motion in a plane of the Euclidean Space.
The first motion considered is an horizontal translation of it. Here is a sketched reconstruction of the displacement field associated to an homogeneous grid to illustrate the local displacement vectors.

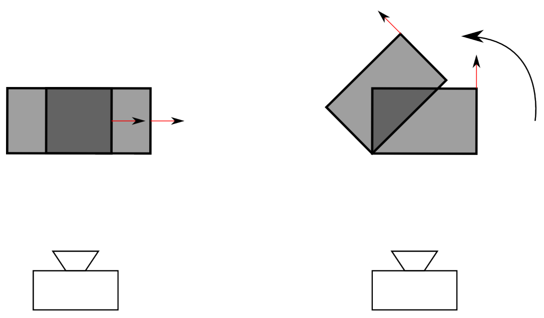

Figure 5: The two different kind of motion studied in his question

The first movement is quite easy to analyse and sketch because our object is having the rigid body property so it's not deformed by the motion. So in this case we have in a case of rectilinear translation an homogeneous translation vector field. The following sketch is presenting the motion field in the real world space.

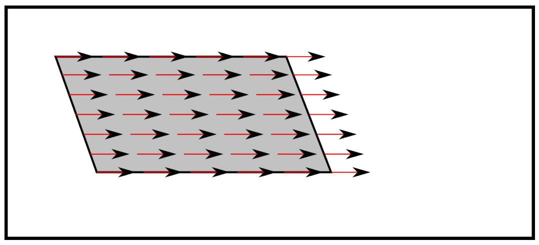

Figure 6: Displacement vector field for translation

Let's now consider a second kind of motion of the rectangle as being a rotation of it around one of it's corner. This motion is more complicated to draw because it's not as homogeneous as the previous one. Indeed, in this case we deal with rotation and the magnitude of the speed vector is proportional to the radius from the rotation center and the associated point. So the displacement vector associated to each point of the rectangle is not the same the closer to the rotation center the smaller the displacement. Trajectories of points lie on a set of concentric circles around the center of rotation.

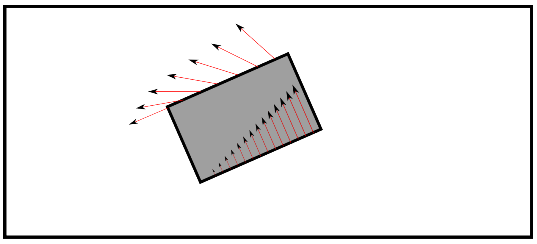

Figure 7: Displacement vector field for rotation around single

The result we presented are the associated vector field in the real world. But now we have to deal with perspective projection due to camera acquisition. So in the rotation case, the rotation is parallel is to the image plane so all the points of the rectangle will have the same scaling factor $\frac{1}{z}$ in the projection. But in the translation case, if the translation plan is slanted to the image plane in the real world, the perspective projection will convert it into a Trapezoid.

Then, when we start processing the images with our optical flow pipeline we have to take in account the issues linked to processing. Indeed, as we present later in the document we have to deal with windows operator that only analyse a small area of the image at each time. So these mathematical and processing operators don't consider the object motion globally as our human vision and perception system does. This issue is called the aperture problem and illustrate the fact that we only analyse motion through a small aperture.

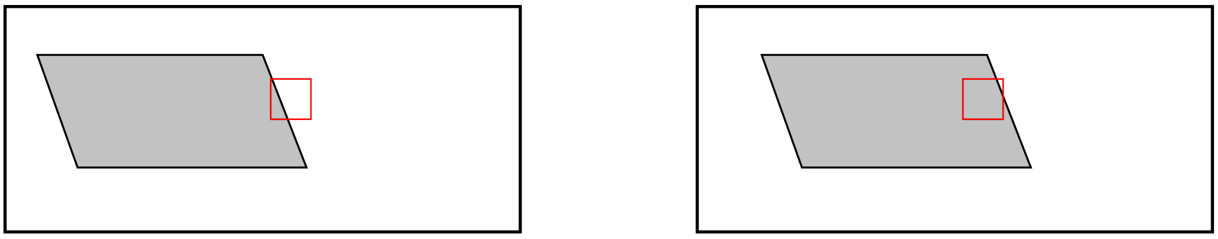

Figure 8: Aperture problem for translation motion

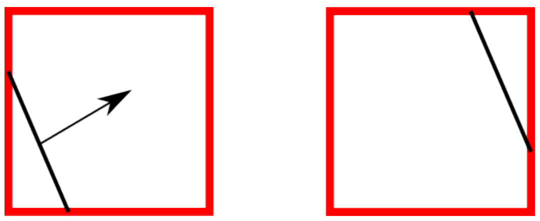

Figure 9: Zoom on aperture problem with wrong displacement vector

Thanks to the previous drawings, we can clearly see that the motion viewed locally by the window processing operator is not exactly the true global motion of the object. Indeed, we see a possible diagonal translation while the true motion is a real horizontal translation. The same effect also appears for the rotating rectangle.

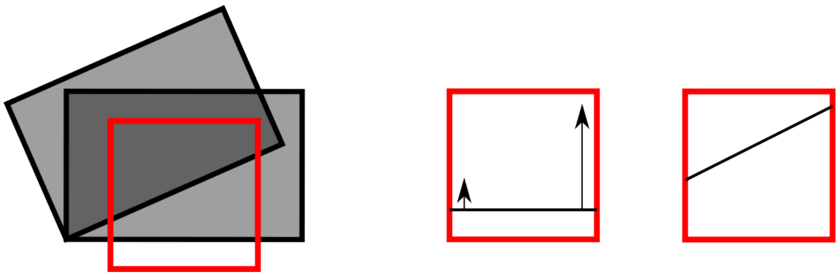

Figure 10: Aperture problem illustration in case of rotating object

In the translation motion based on the local estimation of motion it seems to be quite impossible to reconstruct the real motion. On the rotation motion the local motion is not completely wrong but the reconstruction of the global vector field based on the local estimation is also pretty complicated. So we can say that global motion reconstruction based on local estimation in this case appears to be difficult.

A solution to solve this issue is by considering a corner of the object which will allow us to disambiguate the local displacement estimation.

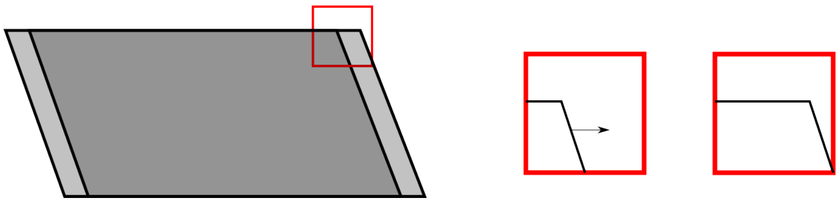

Figure 11: Use of a corner to resolve aperture problem in translation

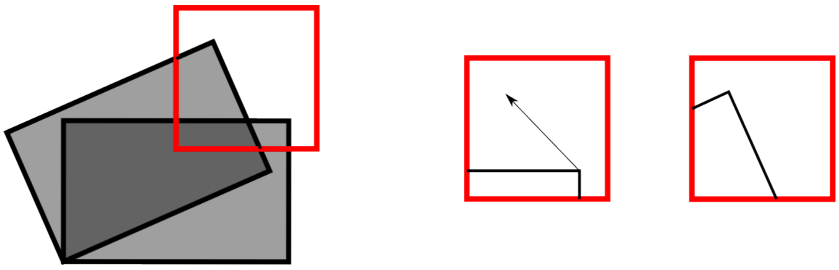
Figure 12: Use of a corner to resolve aperture problem in rotation

As we can see analysing motion from a corner of the object is a good solution to disambiguate the aperture problem. So the solution is to find corner features where the motion is unambiguous. An other solution is to consider at the same time multiple location on edges to find the accurate motion by gathering information from various parts of the objects assuming that we are under the rigid body assumption which gives to each local part of the object the same global motion.

3 Practical Problems

In this assignment, the practical problem will focus on Optical flow reconstruction from a sequence of images. It will involve some image processing notions that have already been covered but that we will remind to explain the method and our implementation.

3.1 Optical Flow

The objective of the Optical Flow method is to reconstruct the displacement vector field of objects captured with a sequence of images. In fact, based on a set of images capturing the motion of one or multiple objects, we want to be able to reconstruct the displacement field associated to each pixel during the time difference from one frame to an other.

3.1.1 Theoretical Presentation

The Optical flow method to characterise motion of objects is defined with a specific equation that we are going to solve. In this section we are going to illustrate and present the steps to derive this equation and some methods to solve it.

Our input data set is composed of a set of images acquired over time. So we have a three dimensional data set composed of two spaces variables that describe the image and one time variable which describe the evolution of space variables in time.

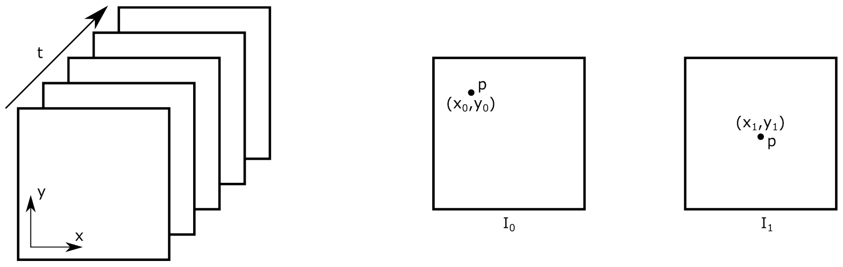

Figure 13: Input data set

In the previous drawing, we present our set of data and its three dimensions. Then, we present two consecutive images where a pixel p has a motion between the two frames that changes its position from (x_0, y_0) to $(x_1, y_1) = (x_0 + u, y_0 + v)$. So, it exists a function of the time variable that describe the evolution of the spatial parameters of each pixel position between two frames.

$$f(t) = I(x(t), y(t), t) \tag{11}$$

The Optical flow method relies on two assumptions that will allow us to derive its equation :
Assumption 1: The motion between frames is small
Assumption 2: Conservation of pixel brightness over time

The second assumption leads to formulate the brightness consistency constraint:

$$\frac{df}{dt} = 0 \qquad \Rightarrow \qquad I(x, y, t) = I(x + \delta x, y + \delta y, t + \delta t) \tag{12}$$

The previous equations means that we assume that in this framework, only the object is moving and its brightness properties remain constant over time so that the shading and light parameters are supposed to be the same.

Let's now considerate the change of position of pixel p introduced before, we can then write the following equations :

$$u = \frac{\partial x}{\partial t} \qquad \text{and} \qquad v = \frac{\partial y}{\partial t} \tag{13}$$

So now let's differentiate the f function with Taylor expansion and use the brightness consistency constraint:

$$\frac{dI}{dt} = 0 \qquad \Rightarrow \qquad \frac{\partial I}{\partial x}\frac{\partial x}{\partial t} + \frac{\partial I}{\partial y}\frac{\partial y}{\partial t} + \frac{\partial I}{\partial t} \tag{14}$$

Then if we use the previous equation:

$$\frac{dI}{dt} = 0 \qquad \Rightarrow \qquad \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v + \frac{\partial I}{\partial t} \tag{15}$$

The two terms before u and v respectively are the image gradients in the x respectively y direction of the image. So if we introduce the ∇ operator of first order partial derivative :

$$\nabla I = \begin{pmatrix} \frac{\partial I}{\partial x} \\ \frac{\partial I}{\partial y} \end{pmatrix} = \begin{pmatrix} I_x \\ I_y \end{pmatrix} \tag{16}$$

and finally if we define the displacement vector $\overrightarrow{v}$ such as:

$$\overrightarrow{v} = \begin{pmatrix} \frac{\partial x}{\partial t} \\ \frac{\partial y}{\partial t} \end{pmatrix} = \begin{pmatrix} u \\ v \end{pmatrix} \tag{17}$$

We can formulate the compact form equation of Optical Flow:

$$\nabla I \overrightarrow{v} + I_t = 0 \tag{18}$$

The goal of this method is to determine the unknown $\overrightarrow{v} = (u, v)$, because the time and spatial gradients of the image can be computed and here we try to estimate motion.

In this method, we rely on estimation of the image gradients in both time and space variables. There is one point to discuss regarding the computation of spatial gradients. Indeed, the spatial gradient being a derivative, if we have a very sharp and straight edge, our gradient computation can fail. To avoid this issue, we are going to introduce a pre processing step that consists in a Gaussian smoothing of the image to smooth edges in the following way:

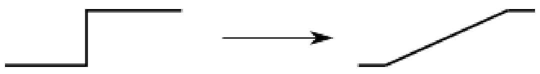
Figure 14: Edge filtering principle

Which produce the following result on an edge:

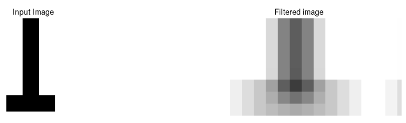

Figure 15: Edge filtering result

The smoothing used will be bi-dimensional and separable Gaussian and will be presented briefly in the implementation section. But something needs to be discussed here. Indeed, smoothing is necessary to ensure mathematical definition of the spatial gradients, but has to be done with respect to the motion we try to capture between images. So our Gaussian smoothing kernel is defined by:

$$\begin{cases} f : \mathbb{R} \to \mathbb{R} \\ x \mapsto f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{(x-\mu)}{\sigma}\right)^2} \end{cases} \tag{19}$$

where σ is the standard deviation of the Gaussian curve, so this is the parameter that will have to match the size of the displacement we try to characterize.

An other aspect will be discussed later and deals with the computational technique to estimate the spatial gradients of the image. Indeed, the time gradient is easily defined by:

$$I_t = \frac{\partial I}{\partial t} = I^*(x, t + \delta t) - I^*(x, t) \tag{20}$$

where I^* is the smoothed image.
But there are several ways to compute image gradients here are two examples to compute them :

$$I_x = \frac{\partial I}{\partial x} = I^*(x + \delta x, y) - I^*(x, y) \qquad \text{or} \qquad I_x = \frac{\partial I}{\partial x} = I^*(x + \delta x, y) - I^*(x - \delta x, y) \tag{21}$$

3.1.2 Normal Optical Flow

In this method we only deal with pixel information and do not take in account the neighbourhood information. As we mentioned in the theoretical part, this could lead to an aperture problem where the overall motion of the object is not taken in account. We only compute for each pixel an estimation linked to the three gradients available. Here is the definition of the normal flow we can compute thanks to a single pixel information:

$$u_{\perp} = \frac{-I_t}{||\nabla I||}\frac{\nabla I}{||\nabla I||} \qquad \text{with} \qquad ||\nabla I|| = \sqrt{I_x^2 + I_y^2} \tag{22}$$

We implemented this method to see what kind of result we could get. Here are some results.

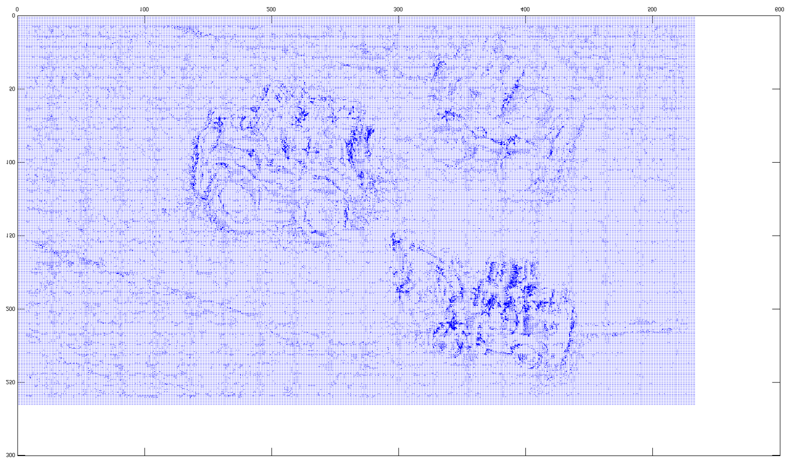

Figure 16: Normal magnified flow computed for each pixel

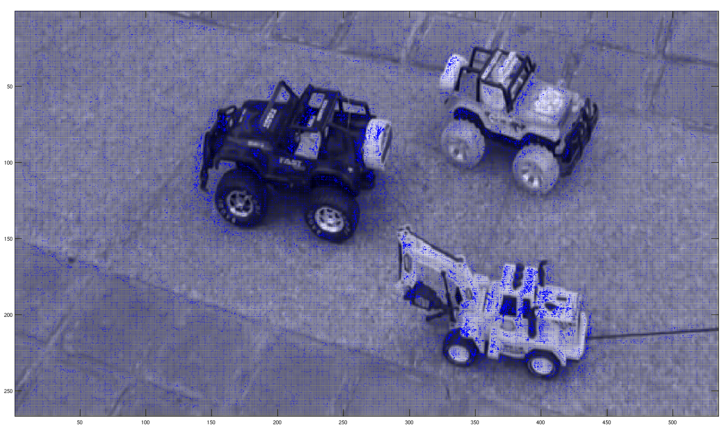

Figure 17: Normal magnified flow overlaid on the original image

With no points of comparison, and based on the issues we mentioned in the theoretical part regarding the aperture issue, this result can appear quite satisfying. Indeed, when we look at the global result, we succeed in capturing moving object and it also seems to show the correct motion. But once we zoom in the vector field we can notified that a lot of vectors are actually incorrect and sometimes definitely go in the opposite direction as it should. In fact, in this dataset the motion is close to the translation case we discussed in the previous section and we can clearly see that considering only one pixel is not enough and leads in most case to a wrong estimation due to the aperture issue.

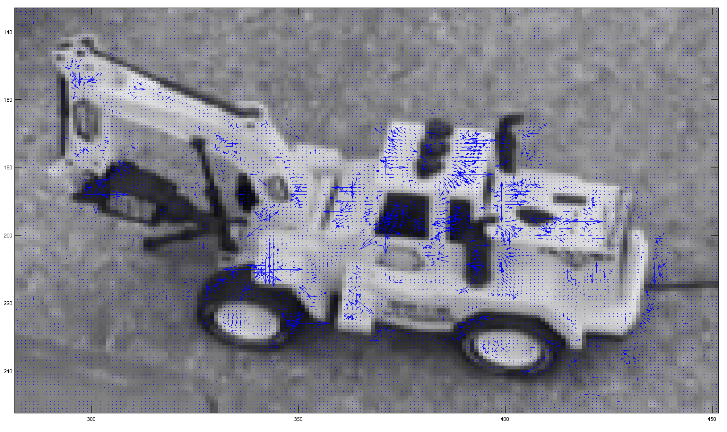

Figure 18: Normal magnified flow for one car showing aperture problem

3.1.3 Lucas and Kanade Method

Lucas and Kanade method is a differential method used to estimate the optical flow under the assumption that the flow is constant in a very small neighbourhood around the considered pixel using a Least Square Estimate to combine informations. The great advantage of this method is that it allows to solve the ambiguity introduced by the aperture problem. Here is a presentation of it.

Let's consider a small neighbourhood composed of pixel p_i with $i \in [\![1, n]\!]$ around each pixel where the estimated displacement vector has to satisfy:

$$\begin{cases} I_x(p_1)u + I_y(p_1)v = -I_t(p_1) \\ I_x(p_2)u + I_y(p_2)v = -I_t(p_2) \\ \qquad\qquad \vdots \\ I_x(p_n)u + I_y(p_n)v = -I_t(p_n) \end{cases} \tag{23}$$

As always, we can easily derive a matrix notation of the previous system and then use linear algebra properties and LSE to estimate our vector of unknowns.

$$A = \begin{pmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_n) & I_y(p_n) \end{pmatrix}, \qquad \overrightarrow{v} = \begin{pmatrix} u \\ v \end{pmatrix}, \qquad \text{and } b = \begin{pmatrix} -I_t(p_1) \\ -I_t(p_2) \\ \vdots \\ -I_t(p_n) \end{pmatrix} \tag{24}$$

Then we can formulate the Least Square Estimate problem :

$$\overrightarrow{v} = (A^T A)^{-1} A^T b \tag{25}$$

which leads to solve the following system:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n I_x(p_i)^2 & \sum_{i=1}^n I_x(p_i)I_y(p_i) \\ \sum_{i=1}^n I_x(p_i)I_y(p_i) & \sum_{i=1}^n I_y(p_i)^2 \end{pmatrix}^{-1} \begin{pmatrix} -\sum_{i=1}^n I_x(p_i)I_t(p_i) \\ -\sum_{i=1}^n I_y(p_i)I_t(p_i) \end{pmatrix} \tag{26}$$

3.1.4 Results and discussion

Here are some images that we used for this project.

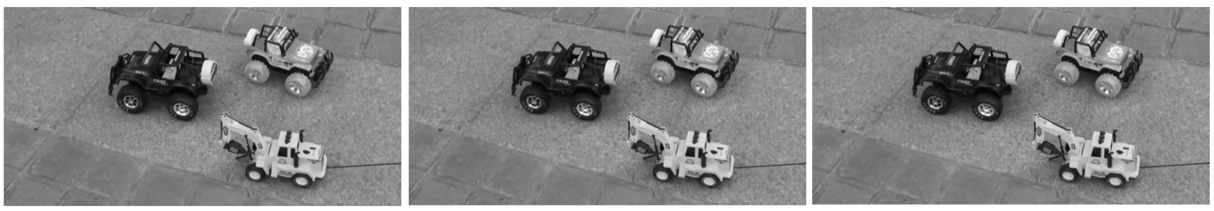
Figure 19: Subsection of the input images set

Let's now present some result of the computation of the gradient images :

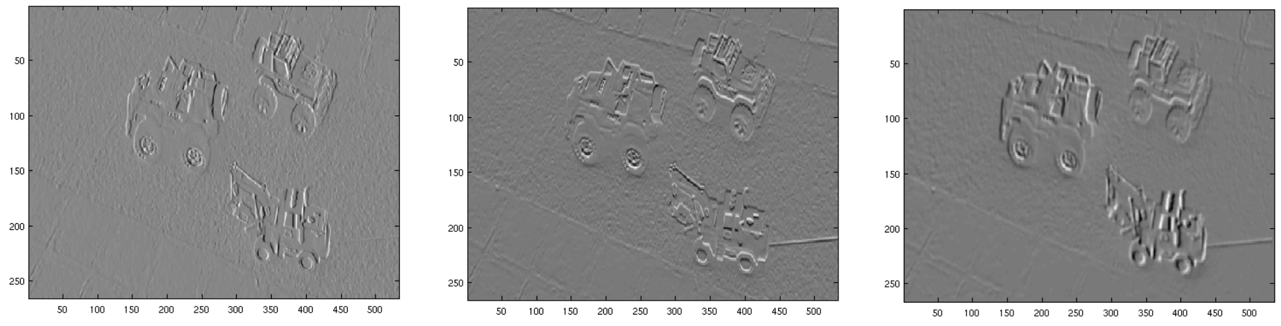

Figure 20: Spatial gradients according x, y and time gradient

Let's discuss briefly these results. Here we used our home-made implementation of gradient filtering made with convolution of derivative kernel with the image.
For the spatial gradients presented before we used the following mono dimensional kernels:

$$\frac{\partial I}{\partial x} \rightarrow (-1, 0, 1) \qquad \text{and} \qquad \frac{\partial I}{\partial y} \rightarrow \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix} \tag{27}$$

On the first image (on the left) we can clearly see that vertical edges are very well defined, that's the effect of the gradient on the horizontal direction, it enhance the differences on this direction and the stronger are created by vertical lines. The same thing is to be noticed for the second image (center) where we have the gradient according to the vertical direction where the strongest discontinuities enhanced are horizontal lines. Regarding the time gradient (on the right) it just a scalar difference of intensity values between two images.

Then we directly implemented the Lucas and Kanade method with a neighbourhood kernel, so there are many aspects to discuss here. The first aspect is the influence of the size of the smoothing kernel we used to originally smooth the image. Indeed, the smoothing kernel has to be scaled to match the displacement between frames. Here is the result of the computed displacement field:

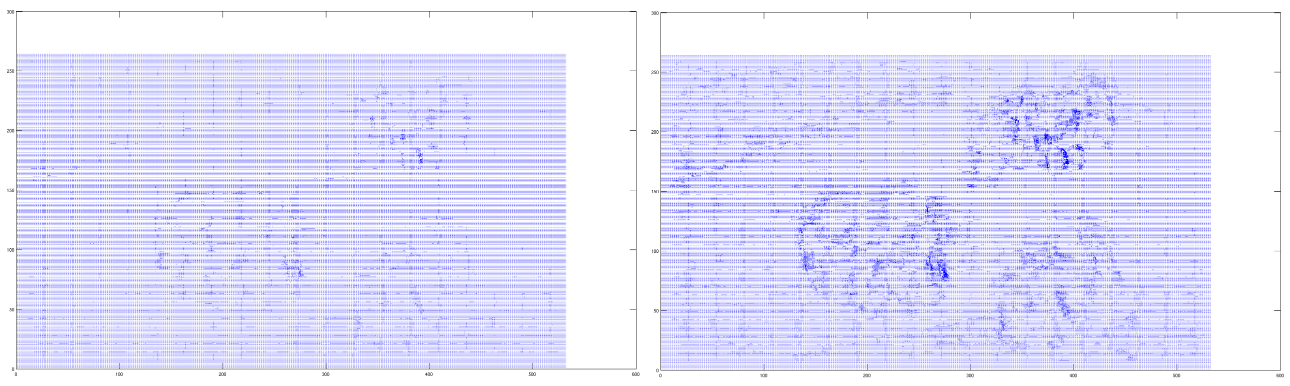
Figure 21: Original Flow field and scaled field with 4x magnification

As we can see on the previous picture, the original computed displacement field has a very low vector magnitude, so to have a more visible and analysable field we magnified it as shown in the second picture.
On both pictures, we can clearly identify three areas where vectors are showing a different vector field structure than most parts of the image. Thanks to our prior knowledge of the objects present in the image we can easily associate them to the three cars. Which appears to be exactly the case if we overlay the field on the original image.

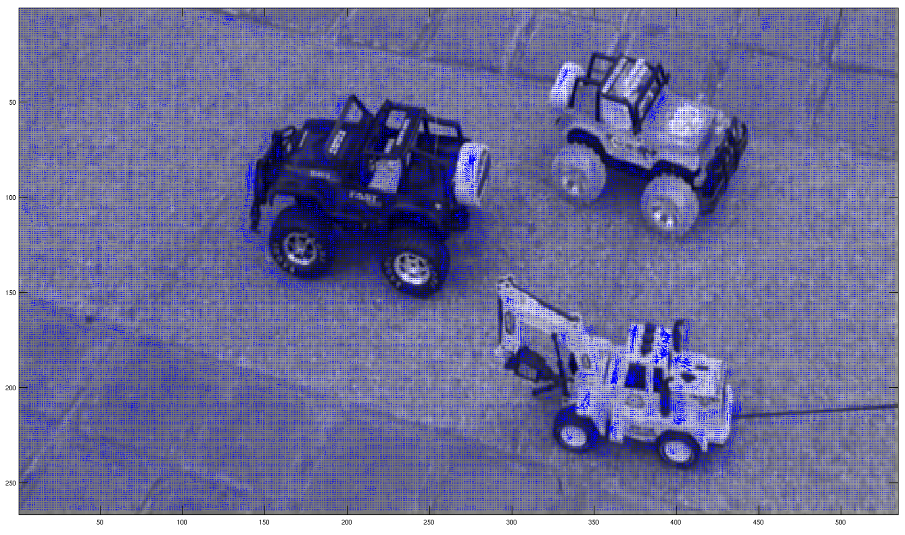
Figure 22: Magnified displacement field overlaid on the original image

Thanks to the previous image we can now confirm that the flow we computed is correctly associated to the motion of the three cars. Let's now look at the specific motion of those three cars:

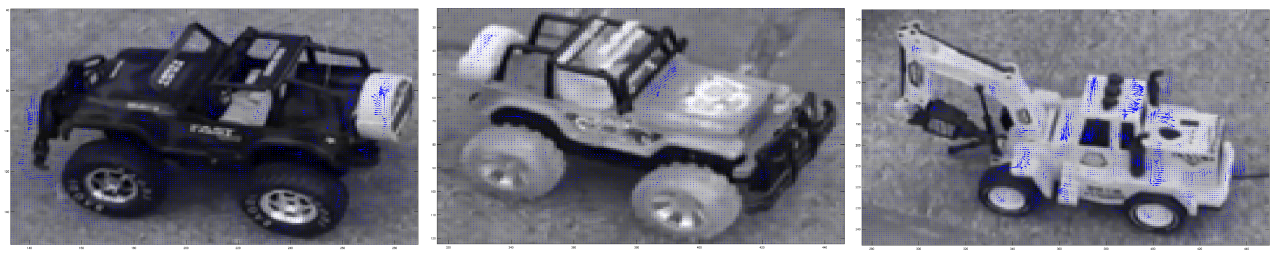
Figure 23: Magnified displacement field for the three cars

It might be complicated to see clearly the associated displacement field but for the first car, we can see that the majority of the displacement vectors are oriented to the left, and for the two others the majority of the arrows are pointing to the left. This seems to be a good start to show that we correctly reconstructed the motion field. So we computed a binary image which shows the orientation of the displacement according the horizontal axis to make sure that our result was correct.

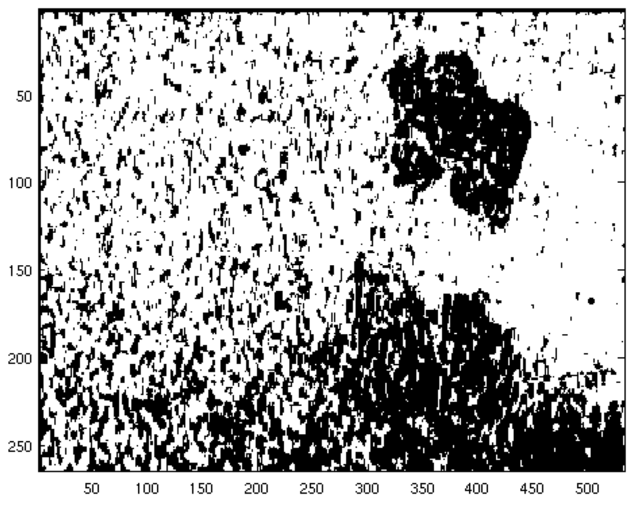
Figure 24: Binary image of horizontal displacement

On the previous image, the white color indicate that the test displacement inferior to 0 is true, which means that it's oriented from right to left. So black color is illustrating a displacement from left to right. So the observation we made before seems to be accurate because we can clearly see that cars 2 and 3 are effectively coloured black so that a left to right motion has been detected, while the first car is coloured white so has a right to left motion. So thanks to this image, it seems

that our implementation is working and that we succeed in reconstructing the motion field as we were required to.
But this image also rises another question, because the background of the image is also coloured in white and by the ways also seems to have a right to left motion. Due to the nature of the image acquisition as a ”on the fly video footage” there is a small motion of the camera between frames which is contained in the temporal gradient and which is taken in account for the displacement computation. So this may be an explanation of the motion associated to the background of the image and which could also be seen in the vector field. And in our model, this motion could sometimes have the same order of the moving object of the scene.

Now that we proved that our implementation seems to be successful, we can discuss some other cases and possible experiment.

The first thing that we need to discuss is the influence of the size of the smoothing kernel in the pre processing step. Indeed, the smoothing has to be big enough to capture the inter frame motion but not to big to avoid a too wide smoothing resulting in a smooth displacement field and a possible reduction of available accurate information.
Let’s firstly measure the displacement of the cars between frames. Here we are going to assume that there is no motion of the camera to simplify our analysis.

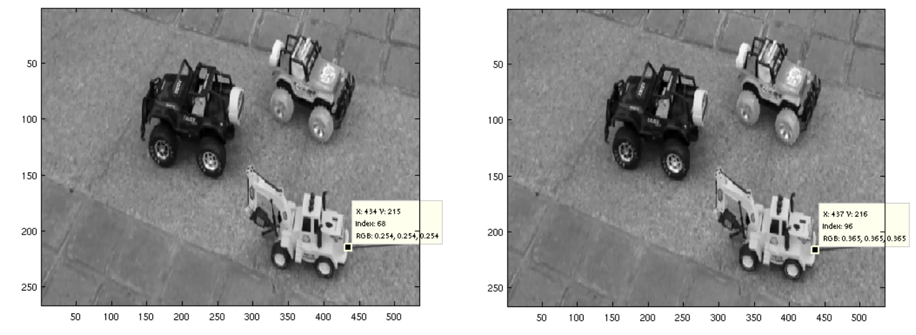

Figure 25: Motion between frames 2 and 3

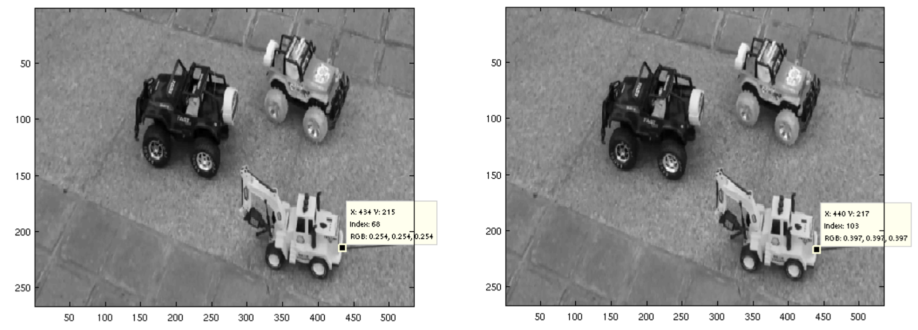

Figure 26: Motion between frames 2 and 4

As we can see on the first picture, we have an horizontal motion of 3 pixels between the two first frames and then 6 pixels between the first and the third frames. So the smoothing kernel has to match this displacement. We tried several experiments and we discovered that with a very small smoothing kernel (3x3) we can reconstruct motion for two consecutive frames but also with separated frames. The resulting field is very low and not very accurate but the reconstruction remains possible. That's an illustration that smoothing is contributing a lot in the accuracy of our reconstruction of displacement field. Also a too wide square deviation of the Gaussian smoothing kernel is jeopardizing reconstruction due firstly to the boundary conditions due to our implementation and also because smoothing too much is image leads to a more uniform displacement field.

Another aspect we explored is the use of the gradient function already implemented in MATLAB rather than our implementation. It uses a slightly different method because it seems to use a finite difference approach. The results are look very similar. This is why we chose to use the Matlab gradient function for more accurate results in latter applications such as Horn and Schunck method, because our current implementation of gradient is slightly reducing the size of the resulting image, and we need size consistency with the temporal gradient which is computed as a scalar image difference. The results are very similar, there is of course a difference because the two methods are different but they are close enough. So we hope that using this function is not violating implementation rules. Our original code for filtering images is presented in the implementation section and we can prove by this result that results are similar enough to allow us to use this function.

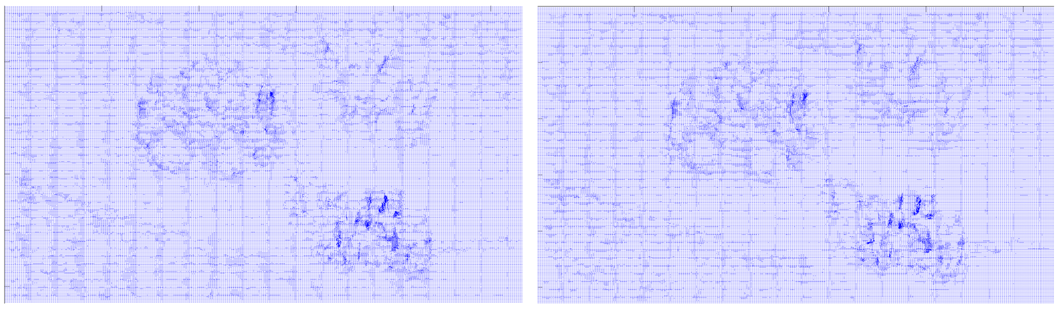

Figure 27: Magnified Optical Flow with our gradient computation and Matlab function

Then another experiment would be to use two different frames with a bigger motion of objects between them and try to see if we can come up using an appropriate smoothing and neighbourhood to reconstruct it.

3.1.5 Spatial Weighting in Lucas and Kanade Method

In the general solution of Optical Flow problem from Lucas and Kanade presented before, each pixel of the neighbourhood kernel is influencing the displacement equally. It means that in case of wide kernel, we give equal importance to a pixel at the extreme end of the kernel and at the closest neighbour. This is why the idea of spatial weighting according to pixel distance to the center of the kernel was introduced. The previous Least Square Equation is slightly modified by the introduction of a weighting matrix.

$$\overrightarrow{v} = (A^T W A)^{-1} A^T W b \tag{28}$$

which leads to solve the following system:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^n w_i I_x(p_i)^2 & \sum_{i=1}^n w_i I_x(p_i) I_y(p_i) \\ \sum_{i=1}^n w_i I_x(p_i) I_y(p_i) & \sum_{i=1}^n w_i I_y(p_i)^2 \end{pmatrix}^{-1} \begin{pmatrix} -\sum_{i=1}^n w_i I_x(p_i) I_t(p_i) \\ -\sum_{i=1}^n w_i I_y(p_i) I_t(p_i) \end{pmatrix} \tag{29}$$

The weighting Kernel is function of the distance between each pixel of the neighbourhood and the center. So a bi dimensional Gaussian weighting function is a suitable option because it takes in account the distance to the center. More details about the Gaussian smoothing and weighting in the implementation section.

3.1.6 Results and discussion

The improvement provided to the previous result does not really appear to be significant compared to what we had before. Here is the resulting displacement field, then the overlaid field on the image and an analysis of car motion.

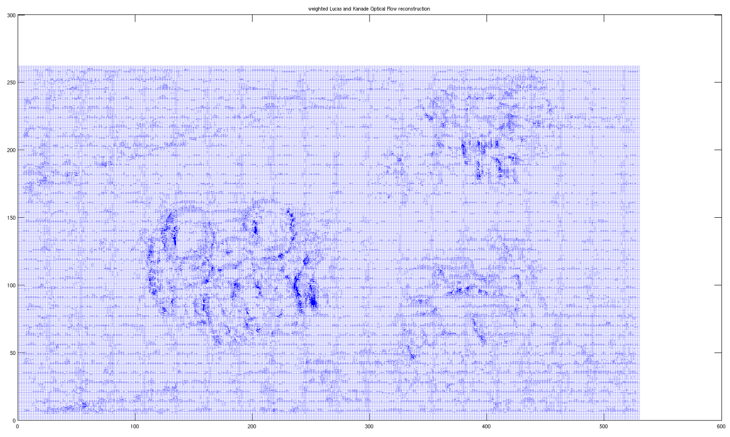

Figure 28: Magnified Optical Flow with weighted neighbourhood in Lucas and Kanade method

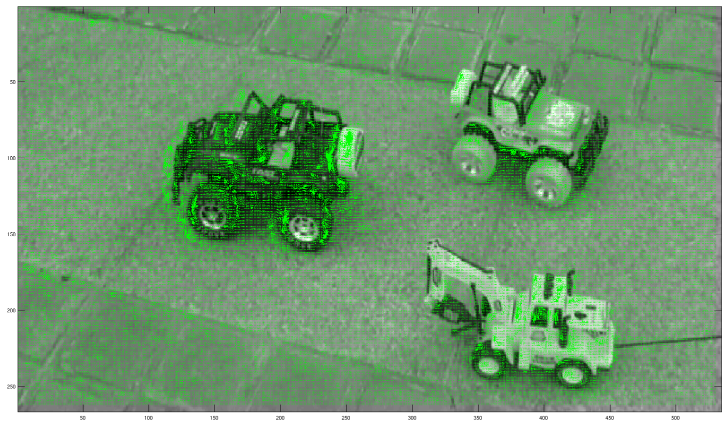

Figure 29: Magnified weighted Optical Flow overlaid on original picture

3.1.7 Other improvement

Due to another processing technique, we were able to produce an other kind of result with an estimated displacement field enhanced on the moving boundaries of objects. We think that these results seems to be quite better than before because they are enhanced on the moving object and reduce significantly the background noise.

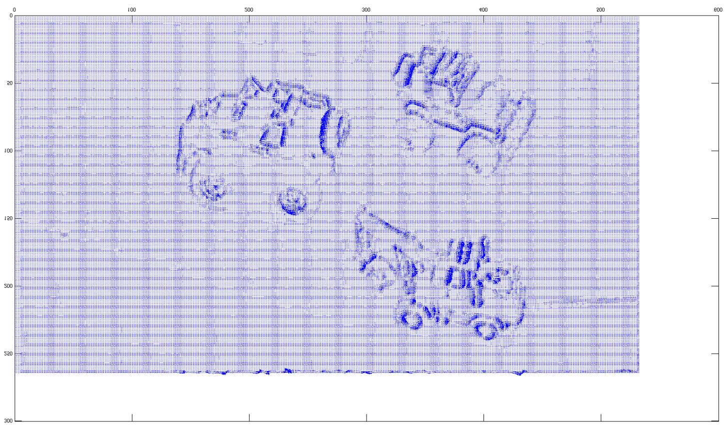

Figure 30: Magnified Optical Flow with an other method

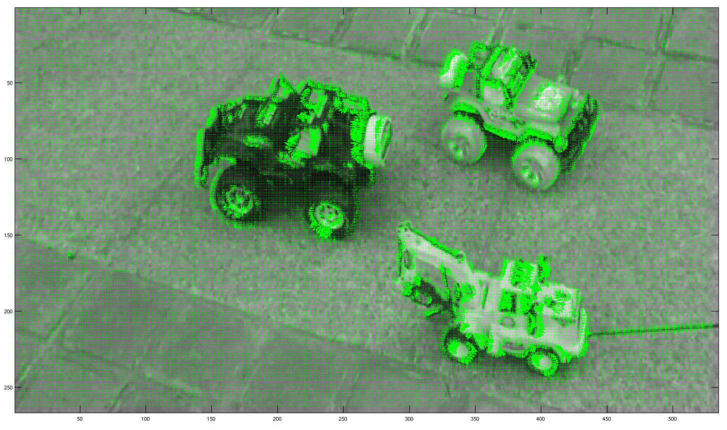

Figure 31: Magnified Optical Flow overlaid on original picture

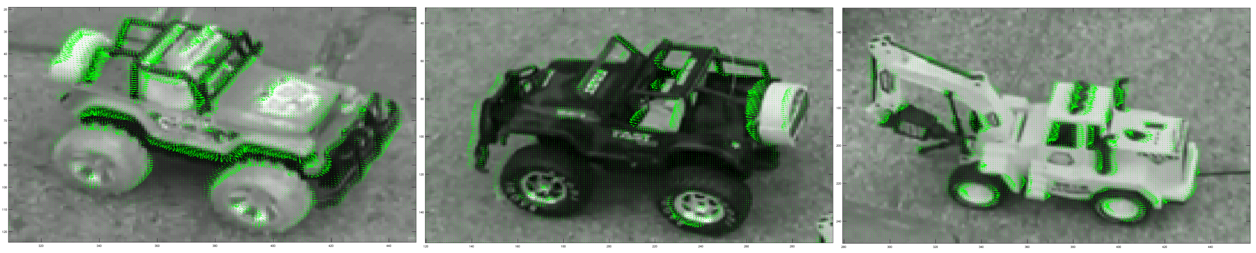

Figure 32: Magnified motion for each car moving in the image

With the previous results, we can clearly see that we clearly capture the moving objects. A great difference with previous results is that the reconstructed vector field appears to be more consistent and based only on the three cars. The background motion is significantly reduced and cars displacement is enhanced. Note that the magnifying factor used to scale the vector field is the same with previous results. So introducing a weighted kernel over the neighbourhood allow us to compute a locally scaled and displacement field. Indeed pixel that belong to the same objects are supposed to have a consistent motion between each other but according to the point of acquisition projective perspective equation car scale this motion also slightly within the moving object. This is why this result really appears to be better because it gather information from local environment with respect to the distance from which it is.

3.2 Horn and Schunck algorithm

The Horn and Schunck method is an other kind of technique used to solve the motion estimation of the Optical Flow problem. As studied in the Shape from Shading project, where Horn also proposed a solution, the use of partial differential equation and optimization technique is an efficient way to estimate unknowns.

3.2.1 Theoretical Presentation

This method relies on the use of partial differential equations and the use of optimization techniques to estimate motion parameters.

In this method we want to optimize the following energy minimisation problem. Let **E** be our energy function defined by a sum of an error constraint function and a smoothness constraint on integrability:

$$\mathbf{E} = E_c + \lambda E_s \qquad \text{with} \qquad \lambda \in \mathbb{R} \tag{30}$$

Where E_c is the error constraint energy defined by :

$$E_c = \int\int_\Omega (I_x u + I_y v + I_t)^2 dxdy \tag{31}$$

And E_s is the smoothness constraint defined by:

$$E_s = \int\int_\Omega ((U_x^2 + U_y^2) + (V_x^2 + V_y^2))dxdy \tag{32}$$

where I_x, I_y and I_t are the gradient of the image previously defined and $U_x = \frac{\partial U}{\partial x}$ and $U_y = \frac{\partial U}{\partial y}$ and respectively for V.

We are looking for an estimation of the parameters u and v which minimize $\mathbf{E}$ so we apply the classical gradient descent method. So we need to differentiate $\mathbf{E}$ with respect to those two parameters as follow:

$$\begin{cases} \frac{\partial E}{\partial u} = 2I_x(I_x u + I_y v + I_t) + \lambda(2U_{xx} + 2U_{yy}) \\ \frac{\partial E}{\partial v} = 2I_y(I_x u + I_y v + I_t) + \lambda(2V_{xx} + 2V_{yy}) \end{cases} \tag{33}$$

where $U_{xx} = \frac{\partial^2 U}{\partial x^2} = \frac{\partial}{\partial x}\frac{\partial U}{\partial x}$ and we can also identify the expression of the Laplacian operator in both equations: $\Delta = \nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$.

$$\begin{cases} \frac{\partial E}{\partial u} = 2I_x(I_x u + I_y v + I_t) + 2\lambda\Delta U \\ \frac{\partial E}{\partial v} = 2I_y(I_x u + I_y v + I_t) + 2\lambda\Delta V \end{cases} \tag{34}$$

So to implement and solve this problem here is the estimation of u and v parameters:

$$\begin{cases} u^{(n+1)} = \bar{u}^{(n)} - I_x \frac{I_x \bar{u}^{(n)} + I_y \bar{v}^{(n)} + I_t}{1 + \lambda(I_x^2 + I_y^2)} \\ v^{(n+1)} = \bar{v}^{(n)} - I_x \frac{I_y \bar{u}^{(n)} + I_y \bar{v}^{(n)} + I_t}{1 + \lambda(I_x^2 + I_y^2)} \end{cases} \tag{35}$$

In the previous set of equations, we used an approximation of the Laplacian Operator based on finite differences : $\Delta u(x, y) - \bar{u}(x, y) - u(x, y)$ where $\bar{u}$ is a weighted average of u computed for each neighbourhood.

3.2.2 Results and discussion

In this model, the computation of spatial and temporal gradient is not affected, this is an independent processing from the displacement field computation. Also, as mentioned before, in this section to benefit from image size consistency we will use the MATLAB *"gradient"* function. But we could again proved that the results would be very similar with our own previous implementation of gradient computation.

To be able to compare the results from this method with the previous we used the exact same smoothing parameter and the same frames and we obtained the following results.

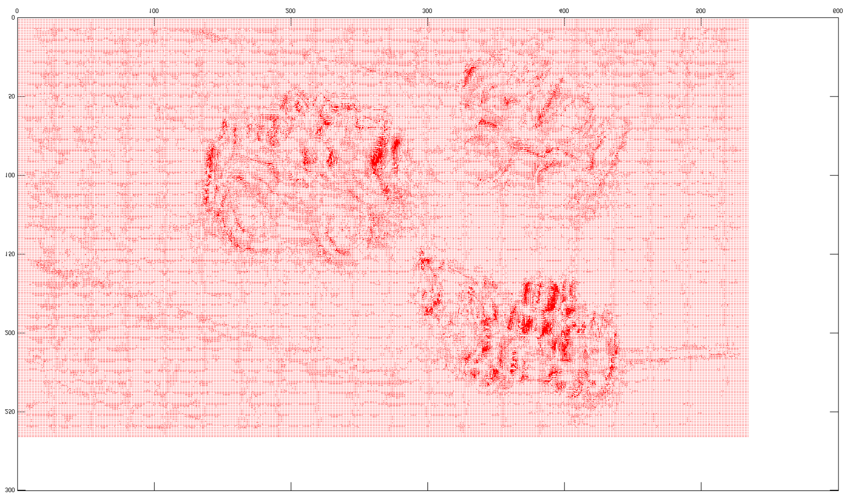

Figure 33: Magnified Optical Flow reconstructed with Horn and Schunck method

We can overlay it on the original image to show that it correctly estimates the displacement of the objects of the scene even if we can already see it on the raw field.

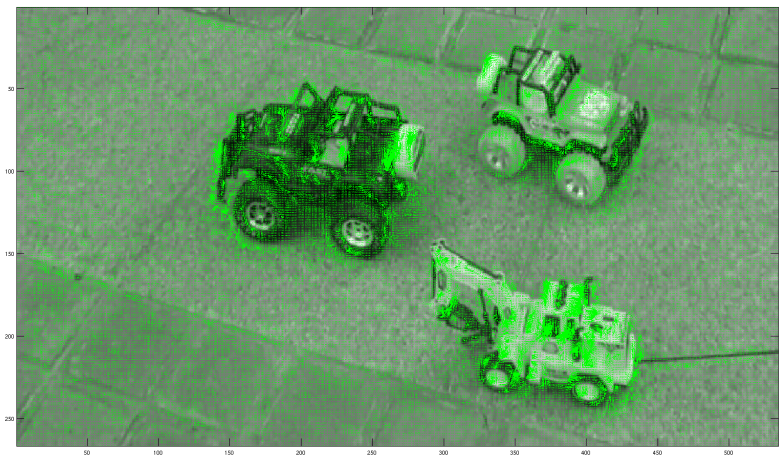

Figure 34: Magnified Optical Flow overlaid on the original image

And as we can see it also seems here that our Horn and Schunck implementation is also succeeding in reconstructing the optical flow for the motion of the three cars. Again, we plotted the horizontal displacement map to see the relative displacement of each car.

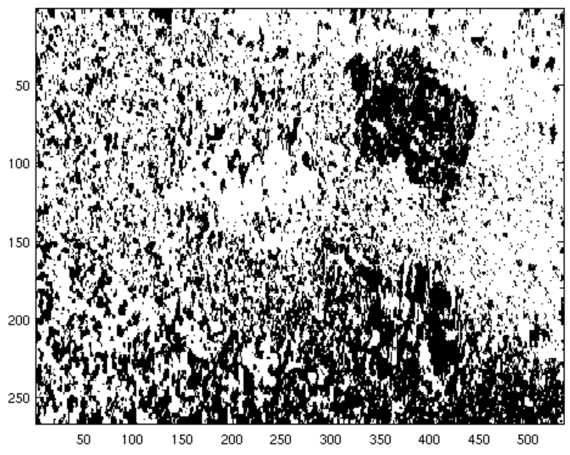

Figure 35: Binary image of horizontal direction

Here again, the displacement is correctly identified the car on the top and the truck on the bottom can be identified from having a left to right motion, while the other car appear white. If we compare this picture with the one we plotted for the Lucas and Kanade method, it appears more noisy but we can identify more details of the top car and the bottom truck.

Some improvement can then be discussed. Indeed, in this implementation we only considered a standard 4 connected neighbourhood, but we could also have a weighted 8 connected neighbourhood. This introduces some slight differences, we will not deeply cover it, we have tested it and an other kernel is available for test in our code.

An other aspect that could also be discussed would be the influence of the number of iteration in the optimization loop. We ran few different experiments and did not notice significant behaviour.

3.3 Motion between frames

One other aspect we can quickly discuss is the inter frames motion. Indeed, we explained that motion between frames as to be small enough to have a nice working solution. The pre processing smoothing also influenced a bit this aspect, because by smoothing more our images and according to the method we use to solve and reconstruct the displacement field, we are still able to reconstruct the displacement field even in case of larger displacement.
In this example we considered the car displacement between frames 2 and 9 and we were able to reconstruct a motion field with each of the presented method. But the most important point is that all of them make sense and present an accurate displacement with the real displacement.

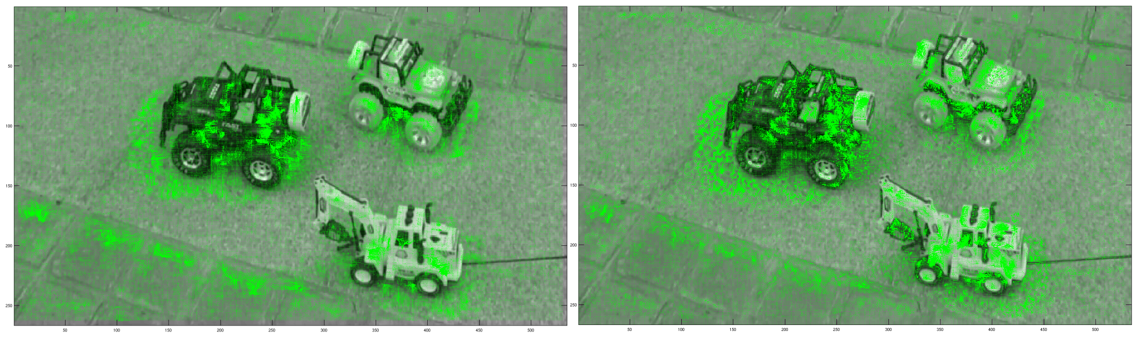
Figure 36: Magnified displacement field for larger displacement

3.4 Other Data sets

The website *http://vision.middlebury.edu/flow/data/* provides us other data sets. We tried a couple of them to see what we can obtain. Here are the results.

The first one is a zooming view into a synthetic representation of Yosemite :

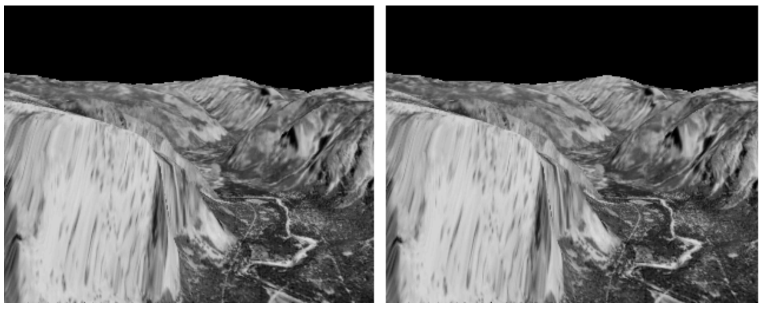
Figure 37: Input Yosemite images

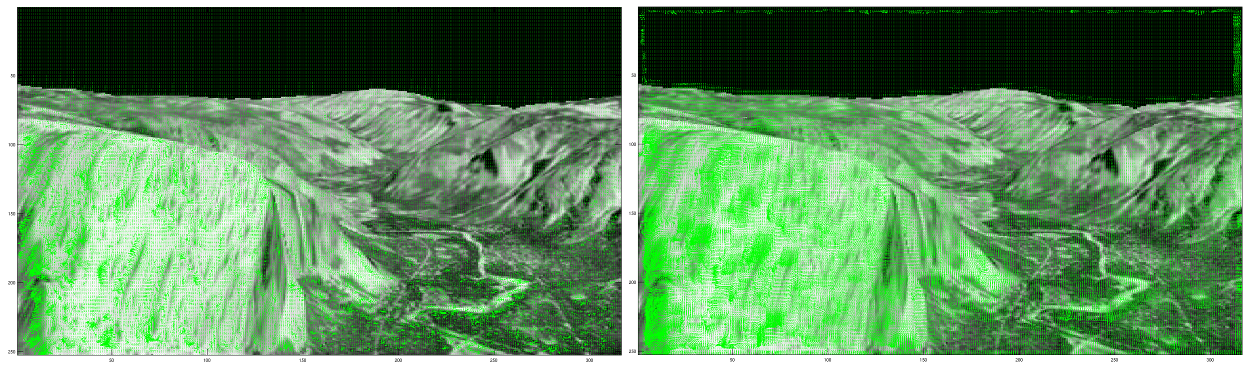

Figure 38: Magnified displacement field with Lucas and Kanade and Horn and Schunck methods

The second one is a moving tree branch acquired with a high speed camera:

Figure 39: Input tree images

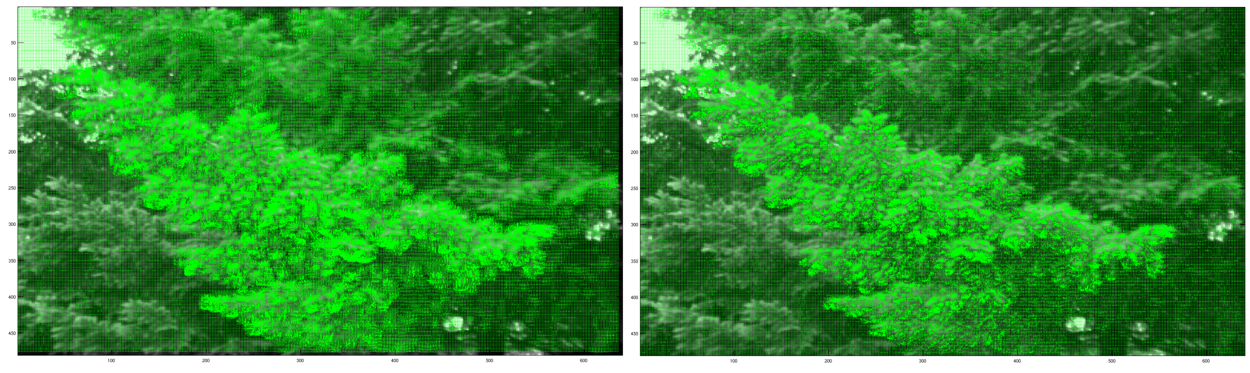

Figure 40: Magnified displacement field with Lucas and Kanade and Horn and Schunck methods

As we can see on the previous picture, we can obtain the displacement over several frames. The results seems consistent with respect to the orientation of the displacement and the shape obtained.

4 Implementation

Here is our implementation for the Optical Flow method to estimate a displacement field.

4.1 Pre processing and Gradients computation

Here is the implementation of the smoothing pre processing step and the various gradient computation.

```
I = imread('toy_formatted2.png');
I2=double(I(:,:,1));
I3 = imread('toy_formatted3.png');.
I4=double(I3(:,:,1));
% smooth images
sigma = 1;
SI2=separable_gaussian('toy_formatted2.png',sigma,0);
SI4=separable_gaussian('toy_formatted3.png',sigma,0);

time_gradient = (SI4-SI2);

weight2 = [-1,0,1];
weight1 = weight2';

for i = ceil(size(weight1,1)/2) :1: size(I,1)-size(weight1,1)+ceil(size(weight1,1)/2)
     for j = ceil(size(weight2,2)/2) :1: size(I,2)-size(weight2,2)+ceil(size(weight2,2)/2)
         convol=0;
         % compute convolution for the neighbourhood associated to the vector
         for b = 1:size(weight1,1)
             convol = convol + (weight1(b)*I2(i-b+ceil(size(weight2,2)/2),j));
         end
         Ix2(i,j)=convol;
     end
end

for i = ceil(size(weight1,1)/2) :1: size(I,1)-size(weight1,1)+ceil(size(weight1,1)/2)
     for j = ceil(size(weight2,2)/2) :1: size(I,2)-size(weight2,2)+ceil(size(weight2,2)/2)
         convol2=0;
             %convolution with vector on the image generated before
             for a = 1:size(weight2,2)
                 convol2 = convol2 + weight2(a)*I2(i,j-a+ceil(size(weight2,2)/2));
             end
         Iy2(i,j)=convol2;
     end
end
```

4.2 Lucas and Kanade Method

Here is the standard implementation of Lucas and Kanade algorithm to estimate Optical Flow.

```
%% solving optical flow

% Lucas and Kanade Method
% define size of neighbourhood considered
kernel = ones(3);

% solve v = (A^T*A)^(-1) * A^T * B associated to the neighbouhood of each
% pixel of the image

for i = ceil(size(kernel,1)/2)+1 :1: size(I2,1)-size(kernel,1)+ceil(size(kernel,1)/2)-1
     for j = ceil(size(kernel,2)/2)+1 :1: size(I2,2)-size(kernel,2)+ceil(size(kernel,2)/2)-1
        t=1;
          for(a=-ceil(size(kernel,1)/2):1:ceil(size(kernel,1)/2))
                    for(b=-ceil(size(kernel,1)/2):1:ceil(size(kernel,1)/2))
                         A(t,1) = Ix(i+a,j+b);
                         A(t,2) = Iy(i+a,j+b);
                         bb(t) = -time_gradient(i+a,j+b);
                         t=t+1;
                    end
                 end
         vecteur(i,j,:) = inv(A'*A)*A'*bb';
     end
end

figure(12465);quiver(vecteur(1:1:264,1:1:532,1),vecteur(1:1:264,1:1:532,2),4)
```

4.3 Gaussian weights

4.3.1 Using formula

To implement our Gaussian filter we use the following formula and few tricks to make it work in MATLAB.

$$\begin{cases} f : \mathbb{R} \to \mathbb{R} \\ x \to f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{(x-\mu)}{\sigma}\right)^2} \end{cases} \tag{36}$$

In out program, the input parameter is the standard deviation σ that is conditioning all the kernel distribution. Indeed, by specifying a value of sigma and knowing that we want to be in the range $\pm 3\sigma$ it's then easy to derive the mean and the size of the vector. The mean is going to be 0 and according to the example, a σ of 2 should give a kernel of dimension 13.

```
for l=-(ceil(3*sigma)):1:ceil(3*sigma)
     g(l+(ceil(3*sigma))+1)=(1/(sqrt(2*pi)*sigma))*(exp(-0.5*((l-0)/sigma)^2));
```

```
end
```

We can of course check few things in this implementation, the first thing being the size of the filter if $\sigma = 2$. In this case we have a 13 elements signal. Then we can check the sum which is indeed equal to one, due to the use of the pdf's definition. The interesting thing to notice in this implementation is that σ is the only parameter conditioning the filter. For instance if we consider the G_3 filter we defined in the theoretical part we are not going to have it here. Indeed, we need to set a value of 0.25 to have a 3×3 kernel. And we have :

$$\begin{matrix} 0.00000011238 & 0.00033501293 & 0.00000011238 \\ 0.00033501293 & 0.99865949870 & 0.00033501293 \\ 0.00000011238 & 0.00033501293 & 0.00000011238 \end{matrix} \neq \begin{matrix} 0.0625 & 0.1250 & 0.0625 \\ 0.1250 & 0.2500 & 0.1250 \\ 0.0625 & 0.1250 & 0.0625 \end{matrix} \quad (37)$$

The difference is that in the second case, it's a 3×3 kernel which approximate a real Gaussian distribution, while in our case it's the real Gaussian kernel for the given standard deviation.

The call for this function is :

$$\boxed{OutputIm = separable\_gaussian(InputImage, sigma, display)}$$

This function is implemented based on the separable convolution presented before.

4.3.2 Using convolution

As presented in the theoretical section, there is an easy mathematical way to approximate the Gaussian distribution. This distribution relies on the convolution of the standard vector $[1, 1]$ to generate Newton's Binomial coefficients. The implementation just consists in a loop of convolution. And then based on separability properties of such a kernel, we can compute multidimensional Gaussian kernels using matrix product.

The call for this function is :

$$\boxed{OutputIm = gaussian\_weigths(n, display)}$$

Here is the implementation to get the Gaussian normalized coefficients:

```
init = [1,1];
gauss = [1,1];
for i=1:100
     gauss = conv(gauss,init)/sum(2*gauss);
end
```

And it also proves the fact that convolving box function will provide a Gaussian because [1,1] is the simple definition of a box function.

Then, to have a 2D kernel, we just need to use separability property associated with a Gaussian kernel :

```
gaussian2d = gauss'*gauss; % vector multiplication V^T * V = Matrix
```

Here are some results:

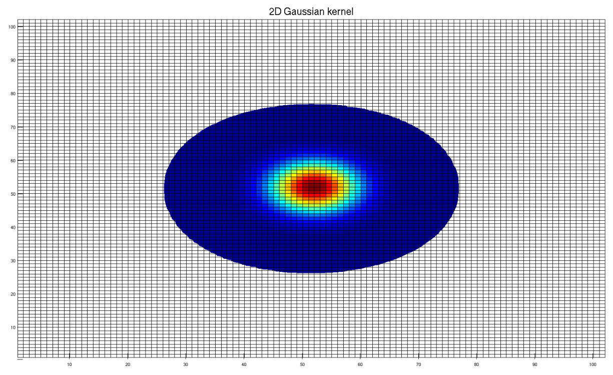

Figure 41: Multiplication of the previous 1D Gaussian to get a 2D Gaussian Distribution

The 3D view of the previous kernel.

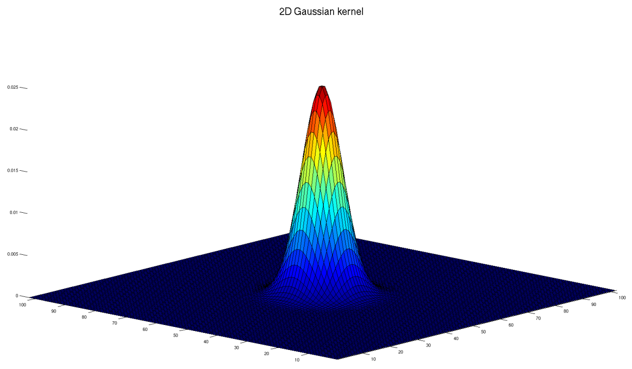

Figure 42: 3D view of the Gaussian kernel

Then to apply it on the image, we just need to use the resulting matrix *gaussian2d*, but the previous kernel is really big so we will use smaller ones and we will need to use the standard deviation as a parameter of the kernel.

4.4 Weighted Lucas and Kanade Method

Here is the implementation which introduces the local weighting as function of the distance.

```
%% Weighted Lucas and Kanade
sigma=1.2
%generate Gaussian Kernel

for l=-(ceil(3*sigma)):1:ceil(3*sigma)
     g(l+(ceil(3*sigma))+1)=(1/(sqrt(2*pi)*sigma))*(exp(-0.5*((l-0)/sigma)^2));
end

G=g/sum(g);
weight1=G';
weight2=G;
gaussian_kernel = weight1*weight2;
%gaussian_kernel = reshape(gaussian_kernel,1,49);

for i = ceil(size(kernel,1)/2)+1 :1: size(I2,1)-size(kernel,1)+ceil(size(kernel,1)/2)-1
     for j = ceil(size(kernel,2)/2)+1 :1: size(I2,2)-size(kernel,2)+ceil(size(kernel,2)/2)-1
        t=1;
          for(a=-ceil(size(kernel,1)/2)+1:1:ceil(size(kernel,1)/2))-1
                    for(b=-ceil(size(kernel,1)/2)+1:1:ceil(size(kernel,1)/2))-1
                         A(t,1) = gaussian_kernel(a+4,b+4)*Ix(i+a,j+b);
                         A(t,2) = gaussian_kernel(a+4,b+4)*Iy(i+a,j+b);
                         bb(t) = -gaussian_kernel(a+4,b+4)*time_gradient(i+a,j+b);
                         t=t+1;
                    end
                 end
         vecteur2(i,j,:) = inv(A'*(A))*A'*bb';
     end
end
```

4.5 Horn and Schunck

Here is our implementation of the Horn and Schunck method for displacement field reconstruction. Our implementation is based on the following algorithm and is quite simple and based on what was previously studied for Horn method for Shape from Shading.

```
Data: two frames with moving objects
Result: displacement vector field $\overrightarrow{u} = (u, v)$
initialization;
compute spatial gradient $I_x$ and $I_y$ and time gradient $I_t$;
while iteration $n \leq n_0$ do
    $\bar{u} = \frac{1}{4}$(u(i-1,j)+u(i+1,j)+u(i,j-1)+u(i,j+1)) ;
    $\bar{v} = \frac{1}{4}$(v(i-1,j)+v(i+1,j)+v(i,j-1)+v(i,j+1)) ;
    $\alpha = \lambda \frac{I_x \bar{u}^{(n)} + I_y \bar{v}^{(n)} + I_t}{1 + \lambda(I_x^2 + I_y^2)}$ ;
    $u^{(n+1)} = \bar{u}^{(n)} - \alpha I_x$ ;
    $v^{(n+1)} = \bar{v}^{(n)} - \alpha I_y$ ;
    $n = n + 1$;
end
```

Algorithm 1: Horn and Shunck Optical Flow method

Here is our MATLAB Implementation:

```
% Averaging kernel
kernel_1 = 0.25*[0 1 0;1 0 1;0 1 0];

% initialization of vector components to zero
u =zeros(size(I2,1),size(I2,2));
v=zeros(size(I2,1),size(I2,2));
alpha=2;
% Optimization loop
for i=1:25
    % Averages of the flow vectors for a given neighbourhood
      mean_u=conv2(u,kernel_1,'same');
      mean_v=conv2(v,kernel_1,'same');
     % Update vector flows based on finite differences Laplacian
     u= mean_u - (Ix.*((Ix.*mean_u) + (Iy.*mean_v) + time_gradient))./(alpha^2 + Ix.^2 + Iy.^
     v= mean_v - (Iy.*((Ix.*mean_u) + (Iy.*mean_v) + time_gradient))./(alpha^2 + Ix.^2 + Iy.^
end
```

5 Conclusion

In this project, we could study and implement an interesting computer vision technique to reconstruct the motion between to image acquisition. The reconstruction of the Optical Flow allow us to estimate the displacement of moving objects of a scene over time.

In this project, we studied two majors methods to perform this motion reconstruction the first one being Lucas and Kanade method which uses the neighbourhood information and the second being Horn and Schunck method which rely on numerical optimization techniques and partial differential equations.

In both method we used a magnifying factor to produce more visible and understandable displacement field and to allow us to make meaningful analysis and comparisons. Indeed, the inter frame motion is small, that's one of the fundamental property used by the previous method to allow it's reconstruction. So the reconstructed vector field we obtained was most of the time to small to be correctly displayed on this document and sometimes on a screen without zooming. This is why we applied uniform scaling of the field.

We showed that introducing a spatial local weighting with a Gaussian kernel in Lucas and Kanade methode was producing a some improvement in the result of the computed displacement field. Our other method provided an enhancement of the computed field on the boundaries of moving objects.

The Horn and Shunck method introduce also a good improvement of the computed displacement field with a completely different method.

In this project we covered two main methods to compute the displacement field in the framework of Optical Flow. All these methods integrate a large number of parameters that can be modified to produce slightly different results. We did not covered the influence of all these parameters, we focused on major modifications that can be obtained. It was interesting to discuss the aperture issue in the theoretical questions because we could clearly see that in our implementations : Lucas and Kanade and Horn and Schunck, we could avoid this phenomenon.

An aspect which could be interested would be to compute a time evolving displacement field, which will evolve according to the time evolution encoded by the different frames. We computed an animated picture of the field between frames that could give a good idea of a result, and thanks to this result, we can see that real time Optical Flow is possible to implement and use for detection.

References

[1] R. Klette, K. Schluns, A. Koschan, Computer Vision Three-Dimensional Data from Images, Springer, 1998.

[2] B. D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision. Proceedings of Imaging Understanding Workshop, pages 121–130, 1981

[3] Wikipedia contributors. Wikipedia, The Free Encyclopaedia, 2013. Available at: http://en.wikipedia.org, Accessed March, 2013.

[4] D. A. Forsyth, J. Ponce, Computer Vision: A modern approach, Second Edition, Pearson, 2011.

List of Figures