Optical Flow Practical Work

Marc Blanchon

***Abstract*—In this report, we will see the work flow for the implementation of basic Optical Flow Computation Algorithm. The code is written in matlab and the main idea is to compute the difference of direction between two cameras capturing the same scene. From this calculus we can then estimate the direction and the displacement of an image relative to the previous one.**

I. Introduction

According to the subject we can see which algorithm as well as which working philosophy has been used in order to realize such a task. The implementations of algorithm will be composed of the explanation and the development of Horn and Schunck algorithm and Lucas and Kanade algorithm. These two algorithm will allow us to compute the optical flow from any two images and give us a visual representation of the movement of cameras according to our original image (the first image used as reference). The next step will be to see the estimation of the displacement and show the image displacement according to camera and image differences.

A. Horn and Schunck

Horn and Schunck algorithm is based on a simple calculation. First it is needed to compute the derivative along X and Y axis as well as time between the two images. Just by using convolution of 2x2 patches oriented in the axis we want to extract feature. We are able to extract the desired features:

$$Ix = ((Image1 * \begin{bmatrix}-1 & 1\\-1 & 1\end{bmatrix}) + (Image2 * \begin{bmatrix}-1 & 1\\-1 & 1\end{bmatrix}))/2$$

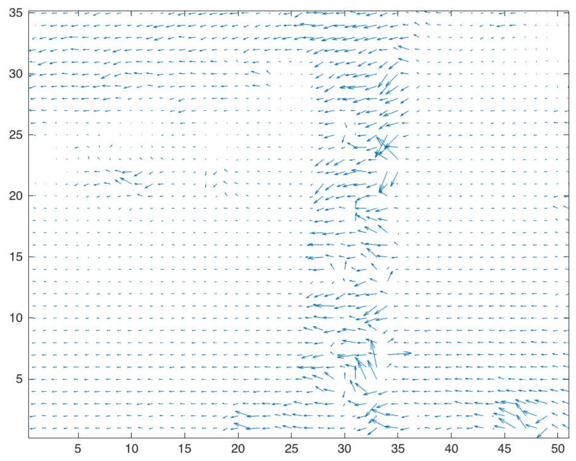
$$Iy = ((Image1 * \begin{bmatrix}-1 & -1\\1 & 1\end{bmatrix}) + (Image2 * \begin{bmatrix}-1 & -1\\1 & 1\end{bmatrix}))/2$$

$$It = ((Image1 * \begin{bmatrix}-1 & -1\\-1 & -1\end{bmatrix}) + (Image2 * \begin{bmatrix}1 & 1\\1 & 1\end{bmatrix}))/2$$

From here we are able to imply our penalization to the X displacement matrix (u matrix) by λ and after this we just apply the equation corresponding to the algorithm following the number of iterations.

$$u = \bar{u} - Ix(It + \bar{v} * Iy + Ix * \bar{u})/(\lambda^2 + Iy^2 + Ix^2)$$

$$u = \bar{v} - Iy(It + \bar{u} * Iy + Ix * \bar{v})/(\lambda^2 + Iy^2 + Ix^2)$$



As we can see, this algorithm is failing and giving bad information along the boundaries present in the images (in this case a tree).

Matlab Code 'HS.m'

B. Lucas and Kanade

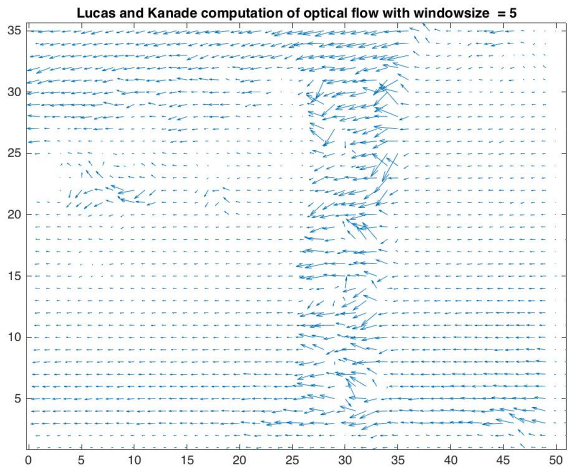
Lucas and Kanade process follow the same filtering process. The difference come from the algorithm equation and process but we can already start directly after the filtering process.

The main idea is to define a window size that will be used as a local patch to extract the optical flow and then combining these will gives us the result. These was the main lines.

Taking care of this window size we are able to compute the the u,v matrices just by simple computation such as resizing the vectors and computing a simple matrix calculus:

$$(A^{\mathrm{T}} * A)^{-1} * A^{\mathrm{T}} * B$$

With A corresponding to a concatenation of column transform of Ix and Iy, and B corresponding to column transformation of It.



In this algorithm we can also see that the boundaries are very bad because of the too fast change of intensity.

Matlab Code 'LK.m'

C. Hierarchical Lucas and Kanade

In this part, the coding part haven't been completed. Instead of explaining the process and equations, this part will describe how the code could be written in order to process the algorithm. First we have to consider that the Hierarchical Lucas and Kanade Algorithm is an expanded version of Lucas and Kanade algorithm including the dimension reduction process. Starting from the biggest patch size, the patch will decrease its size and the optical flow will be recomputed and at the end we will consider the final optical flow vector as the sum of all the vectors calculated.

In order to develop such a process, two ways are open, the descending or the ascending but to develop this algorithm the process chosen would be the descending. Like this we will be able to compute the u and v matrices by simple derivative and additions. So as a process the first task is to compute the Lucas and Kanade algorithm on the first images with the maximum box size that we input on the algorithm. And by iterations, we will update the optical flow at (i,j) coordinates as a combination of all the previous computed optical flow vectors. We will do this for all the view box looping from the maximum value of the view box going to a window box size of 1x1.

The main idea in this algorithm is to follow the optical flow in order to follow it and compute the optical flow at the new location. This is not a problem because as the rules of vector apply everywhere (even in images, comparing them to simple matrices), as we follow the optical flow vector and keep it in memory, we can say that the sum of the vectors from A to C passing by B is the same as having a unique vector from A to C without any 'B interruptions'.

As we follow this process, and as the process work flow is iterative, from the all sum from the full dimension of the images we are able to recover u,v vectors and this has the interest of having a better precision in the mean of we are able to take care of the boundaries without having full non-sense vectors along the boundaries.

D. Parametric motion estimation and application to image stabilization

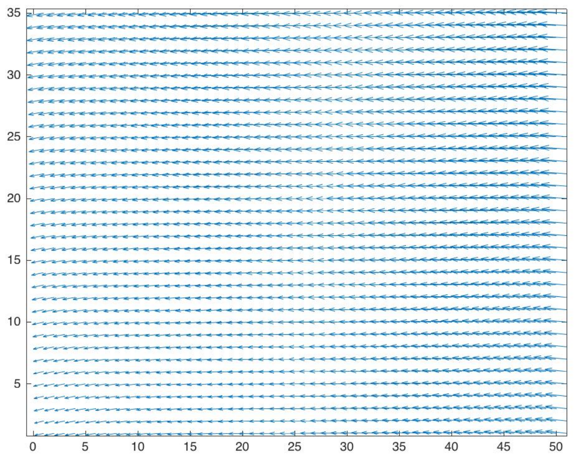
To avoid the formulas and trying to be concise, the calculation process has been written and commented in the matlab code.

This part is about 'How can we reconstruct the image displacement from one image to the next one?'. In this case we are simply using optical flow and some mathematical tricks that will provide us a solution. We can also see that the code is taking care only of two images but this can be extended to N images till we have to add one loop in the code to browse the file and to make the computation along the image and the directly next one.

In fact we will explain the concept, we are using optical flow to move the pixel from the base image by this displacement on a new image.

$$New\_Image(X,Y) = Prec\_Image(X\_Disp, Y\_Disp)$$

Which mean that (taking care of boundaries after all), we are moving the pixel from the previous image by the optical flow of the both images concerned. and this will lead us to an evolution of the image.



The first figure is the displacement that we will apply on the image to fit the view of the second camera. For the second image we can see some black border that mean that the image has moved and replaced the blank spaces by 0 rgb values or black color.

Matlab Code 'Lab.m'

E. Main Problems encountered

For the main problems encountered, we can talk about the dimensions of the matrices. In fact in Matlab we are dealing with dimension and indexes and reshaping, and this was the main problem. When we talk about the motion estimation, when we are mixing indexes, we finally lose the track and the image is reversed.

Another problem is the running time. For the LK.m Lucas and Kanade algorithm, we are dealing with concatenation of matrix without allocating memory, and despite a good CPU it can takes minutes to run.

II. Conclusion

At the end we can conclude that the project learned how we can compute difference between images differentiated by the point of view in another way and also lot of possible extension such as pattern recognition, motion estimation. The main idea of these projects is to show us new methods and different way of computing to success in our future challenges.