MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

LICENSE THESIS TITLE

LICENSE THESIS

Graduate: Firstname LASTNAME
Supervisor: scientific title Firstname LASTNAME

2015

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Firstname LASTNAME**

LICENSE THESIS TITLE

1. **Project proposal:** *Short description of the license thesis and initial data*
2. **Project contents:** *(enumerate the main component parts) Presentation page, advisor's evaluation, title of chapter 1, title of chapter 2, ..., title of chapter n, bibliography, appendices.*
3. **Place of documentation:** *Example*: Technical University of Cluj-Napoca, Computer Science Department
4. **Consultants:**
5. **Date of issue of the proposal:** November 1, 2014
6. **Date of delivery:** June 18, 2015 *(the date when the document is submitted)*

Graduate: _____

Supervisor: _____

MINISTRY OF EDUCATION AND SCIENTIFIC RESEARCH

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind autenticitatea lucrării de licenţă

Subsemnatul(a) _____, legitimat(ă) cu _____ seria _____ nr. _____
CNP _____, autorul lucrării _____
_____
_____
elaborată în vederea susţinerii examenului de finalizare a studiilor de licenţă la Facultatea de Automatică şi Calculatoare, Specializarea _____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _____ a anului universitar _____, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

În cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data _____

Nume, Prenume _____

Semnătura

De citit înainte (această pagină se va elimina din versiunea finală):

1. Cele trei pagini anterioare (foaie de capăt, foaie sumar, declaraţie) se vor lista pe foi separate (nu faţă-verso), fiind incluse în lucrarea listată. Foaia de sumar (a doua) necesită semnătura absolventului, respectiv a coordonatorului. Pe declaraţie se trece data când se predă lucrarea la secretarii de comisie.

2. Pe foaia de capăt, se va trece corect titulatura cadrului didactic îndrumător, în engleză (consultaţi pagina de unde aţi descărcat acest document pentru lista cadrelor didactice cu titulaturile lor).

3. Documentul curent **nu** a fost creat în MS Office. E posibil sa fie mici diferenţe de formatare.

4. Cuprinsul începe pe pagina nouă, impară (dacă se face listare faţă-verso), prima pagină din capitolul *Introducere* tot aşa, fiind numerotată cu 1.

5. E recomandat să vizualizaţi acest document şi în timpul editării lucrării.

6. Fiecare capitol începe pe pagină nouă.

7. Folosiţi stilurile predefinite (Headings, Figure, Table, Normal, etc.)

8. Marginile la pagini nu se modifică.

9. Respectaţi restul instrucţiunilor din fiecare capitol.

Contents

Chapter 1

Introduction

This chapter will present an introduction to the field of optical flow. General concepts from the domain will be described, as they are placed in a historical context.

1.1 Project Context

Optical flow can be defined as the change of light in an image, for example on the retina, that happens as a result of movement of the scene or the viewer.

The optical flow problem does not originate in computer vision, but in psychology, psychophysics. One of the most influential man in this field is J.J. Gibson, who, in 1940s describes the visual stimulus that animals use to navigate through the world around them.

Gibson stated the problem as "How do we see the world around us?". In his many publications around the 50's, he tries to explain this phenomena and describes some principles that stand at the foundation of modern day approaches to calculate the optical flow. In his book[1], Gibson explains how the stimuli are correlated to the perception of motion.

We perceive the motion in 2D as a projection of the 3D world. This projection is nothing else but the light reflected from the environment, hitting our retina, referred as the optical array. In a analytical context, he defines this movement as a perception. Each element is detected and a pair of coordinates are associated to it. Then each element receives a motion descriptor as either a pair of elements or a speed and direction component, in successive moments in time.

As the nature is the grand engineer, the same model is applied in the field of computer vision. Optical flow methods are concerned with calculating the motion between two consecutive images.

1.2 Project Domain

Optical flow applicability resides mostly in computer vision domain, where algorithms for detecting image depth are used to introduce depth in 2D scenes. Optical flow

computation aims to estimate motion vectors, which are used to describe how consecutive 2D image frames transforms in time .

Optical flow is also used in robotics, where problems like object detection and tracking, movement detection and robot navigation need to be solved.

Optical flow can be used in video compression to reduce redundancy in video data, by encoding the flow on consecutive frames.

In driver assisted systems, optical flow algorithms can be applied in collision detection, to detect dangerous situations, e.g. detecting pedestrians or other vehicles. Also the depth can be computed from optical flow, in a steroevision problem.

The wide range of applicability of optical flow techniques make this domain a continuous interest for the scientific community. However, the task of calculating optical flow is not a simple one and various approaches exist to this problem.

In the next chapter the fundamental algorithms used in optical flow will be presented and discussed. Then we will briefly discuss about various mathematical models and what are the main approaches for finding the flow.

1.3 Thesis structure

This section will shortly describe the chapters contained in the thesis. There are a total of eight chapters, as follows:

- **Chapter 1 Introduction - Project Context** In this chapter, a general introduction is offered and the project domain is described. Here, we define the context in which optical flow computation exist, motivations for such algorithms and their application.

- **Chapter 2 Project Objectives and Specifications** This chapter will establish the main objectives in computing the optical flow. Intuitive examples of what the optical flow is will be presented.

- **Chapter 3 Bibliographic research** Chapter three will present main aspects of the field and will shortly describe relevant approaches from related work. Some key features of the algorithm will be shortly discussed.

- **Chapter 4 Analysis and Theoretical Foundation** This chapter will analyse mathematical models of classic algorithms and other improvement features will be discussed in detail. We will look at different formulations for the optical flow, improvements and solving models.

- **Chapter 5 Detailed Design and Implementation** Here, we will present our detailed approach in solving the flow problem based on a cross correlation formulation and a LMN minimization technique.

- **Chapter 6 Detailed Design and Implementation** This chapter will present how can the key elements of the problem be implemented.
- **Chapter 7 Testing and Validation** Details of the output and testing of our implementation compared with others studied will be discussed here.
- **Chapter 8 User's manual**
- **Chapter 9 Conclusions** Conclusions regarding the project will be highlighted in this chapter. Further possibilities for development of the algorithm are presented.

Chapter 2

Project Objectives and Specifications

From the first methods published around 1980, the field advanced trough the decades little by little, up to these days when it can bee computed with high accuracy, and the computation time comes close to real time.

Optical flow is a big area of research form computer vision. It is given by a vector field of velocities which describe the movement of the elements of the projected environment. With a large range of applications, it is a rich source of information.

The changes in the optical array are stocked in a series of frames. The perceived image is a 2D array of pixels. Each pixel is a element in the movement. An accurate and robust flow estimation implies an approximation for each pixel of the frame. The result will be a vector field indicating the direction and that each pixel takes between frames. Further we will discuss dense flow algorithms. Unlike sparse algorithms that process only relevant features of the frame sequence, like corners or edges, dense methods compute the flow for each pixel of the image.

Optical flow is an old field of study. The first algorithms were published in 1980. They are the foundations of modern approaches.

Computing an energy function from the sequence of frames and the flow, and trying to minimize it is the short description for most of the algorithms. For example, the brightness constraint formulated by Horn and Schunk is still the starting point of many algorithms in describing the energy function. Also, the pyramidal implementation of Horn and Schunk's it is also present in modern approaches, solving the large displacement problem. This are some of the fundamental concepts stated by the fathers of optical flow. All this will be further discussed in the next chapters.

These core algorithms were for a long time considered the benchmark for other algorithms in dense optical flow. In time other researchers brought their contribution to the field of study by small changes in the previous algorithms. Changes vary from particularizing the weight of contribution of each pixel to the computation to adding new terms to the minimization problem, to even rewriting the energy function in a different penalty.

With decades of small steps to improvement, algorithms today reach a high accuracy,

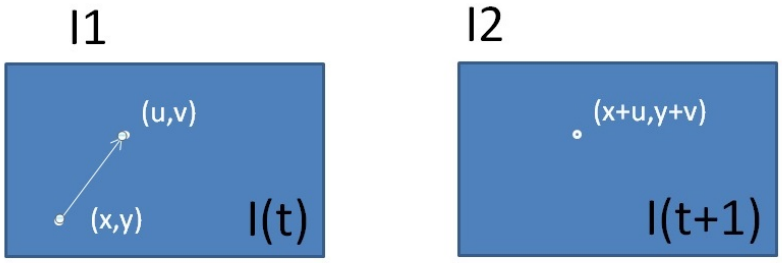


Figure 2.1: Moving Dot

increased with up to 13 times from the original Lucas-Kanade in terms of angular error as can be seen on Middlebury database, and can reach near real time speed.

2.1 Optical Flow Definitions and Examples

The optical flow is defined from a field of vectors (u, v) which define the velocities of the pixels in a image. Solving the optical flow means finding this (u, v) correspondence between the pixels of 2 consecutive frames.

Let us dfine the optical flow on the example in image 2.1. If we have the 2 frames, which are consecutive in time, we observe the white dot in the first frame (x, y). In the second frame it had moved up and to the right, reaching the position $(x + 1, y + v)$. Now, The flow is defined by the displacement of this dot. We say that the flow is defined by the vector (u, v). The assumption on which most of the dense models are formulated is that moving pixels do not change their intensity in time. In our example

$$I(x, y) = I(x + u, y + v)$$

.

Taking an more complex example, 2.1 the optical field is composed from $m \cdot n$ vectors corresponding to each pixel. This exemplifies the flow between image frames, which are matrices of pixels. The algorithms that compute such a structure for the flow are called dense, unlike sparse methods that compute the flow only for key features of the frames. The flow is formulated in a energy function, based on the brightness constraint and others, depending on the algorithm. This function should be ideally 0. A simple (but not very efficient) example of the energy function is $\sum |I(x, y) - I(x + u, y + v)|$. In 2.1 if we add all the difference between pixels in the first image and their corresponding from the next, we will find that the result is 0.

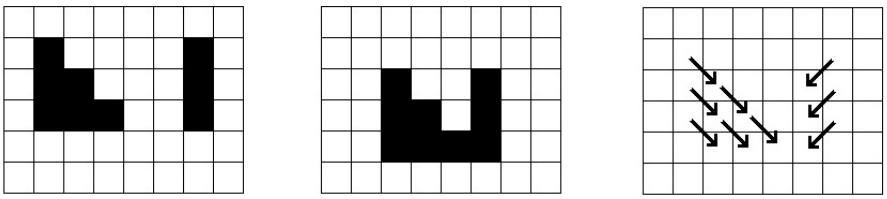


Figure 2.2: Moving Objects

2.2 Solving the Optical Flow System

Solving the system is the expensive part of the algorithm. As we have 2 equations for each pixel, this is the most critical part of the algorithm. The final system will have thousands of equations. The solution can be computed iteratively or it can be parallelized in sparse matrices.

The solution available is based on the chosen model. The classic formulation are based om a L^2 norm. With variational calculus, their derivatives are equated to 0. For this simple solvers are available. Gauss-Seidel and Jacobi iterations are the most handy solution. They are simple, fast, and memory inexpensive. This are classic approaches for the L^2 norm. We used them to implement the Horn and Schunk method and they are discussed in the next chapters.

In latest approaches there is a tendency to lose the L^2 formulation in favour of the L^1. Although it is more computationally expensive to solve then, this formulations are preferred because of it's robustness. We study the Proximal Projected Point and the Least Mixes Squares strategies.

2.3 Project Objectives

With a highly complex mathematical model and a large numbers of approaches, the main scope of this project is to familiarize with the basic concepts and methods of the optical flow computation. In the project we study some classical algorithms, Horn Schunk and Lucas Kanade at different levels.

We study and implement some new approaches based on mixed $L^2 - L^1$ norm optimization problems.

The main achievement is the implementation of an Least Mixed Norm (LMN) algorithm, which will be presented in section 4.5.2.

Such kind of algorithms are successfully used in image restoration and one of their strongest points is the adaptation to the overall problems ($L^2 - L^1, L^1 - L^1$) and tasks.

Horn and Schunk is a global method of computing the optical flow. Based on the brightness and the smoothness constraint, an error function is described. The function depends on thee intensity of the pixels and the flow. Taking a initial guess for the flow field, the scope of the algorithm is to minimize this error function by varying the flow vectors. The minimization problem transforms in a system of equations by differentiating with respect to the horizontal and vertical component of the flow. Using a Jacobi approximation, the flow is updated trough iterations, until the desired output is achieved.

Lucas Kanade is a local method. The error function is based this time on the brightness constraint only is constructed. It is also dependent of the intensity of the pixels and the flow. For each pixel the computation is done by searching in its near vicinity. The minimization problem it is again written as a system of equation by differentiation technique. For the system is solved by taking an initial guess and then computing the remaining flow.

We've implemented this algorithms and their pyramidal version.

As a staring point for our algorithm we have started from a correlation approach [2]. The energy function is formulated with the correlation transform of the intensity of the pixels. It is based on a sum of squared differences to which an L^1 smoothness term is added.

In 5.2, this energy function is solved with Projected Proximal Point(PPP). We've tried a different approach for minimization, Least Mixed Norm (LMN), for which we hope for a faster convergence.

The final purpose of this thesis is to study the results of both minimization techniques. We compare them in terms of computation time, angular error and end point error.

2.4 General Flow of events

In this section we present a flow of events which all our implementations will respect.

1. Basic Flow
   Use-Case Start
   This use-case starts when user launches the system.

   (a) System reads the settings from a file.

   (b) System loads the images from the provided location.

   (c) System prepares matrices, pyramids and other components

   (d) System solves linear system.

   (e) System displays the resulting vector field

   Use-Case End This use-case ends after the computed flow is displayed on the screen.

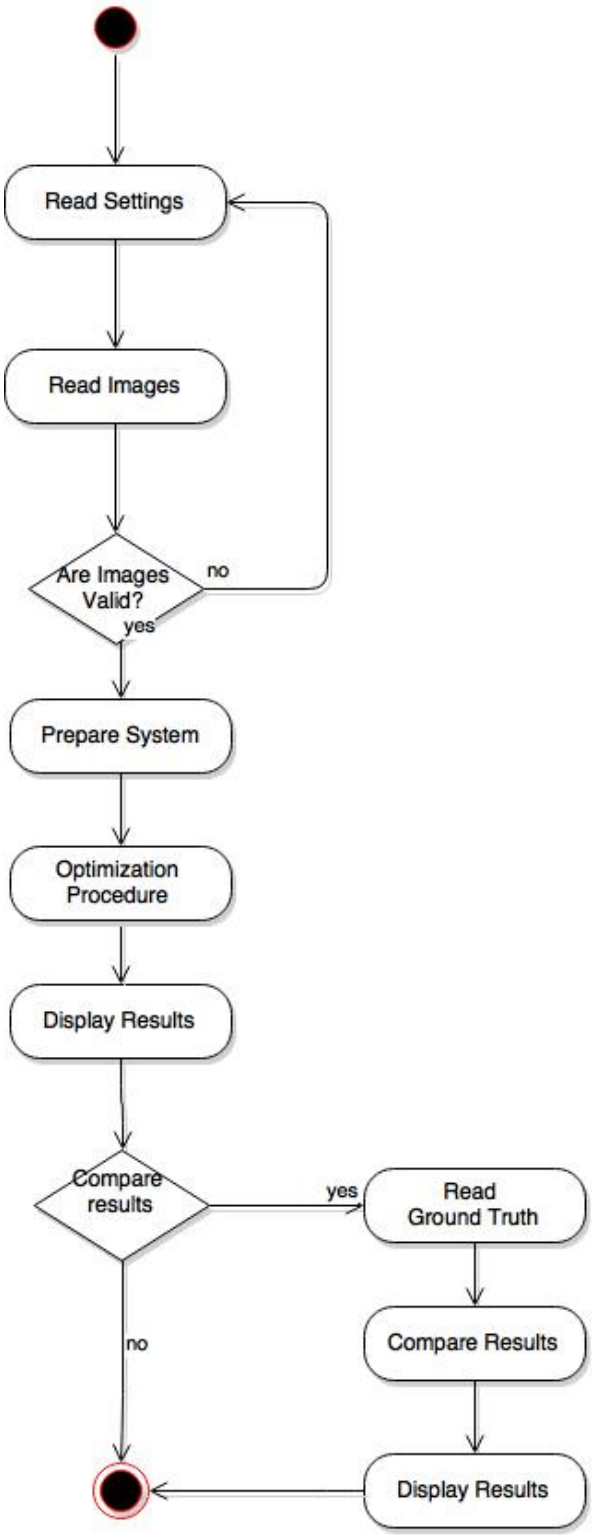


Figure 2.3: General Flow of the System

2. Alternative Flows

   (a) System encountered an error at reading the files.
   Can occur at step 1b

      i. Program displays error message and asks user to revise settings
      ii. Starts again the basic flow.

   (b) Comparing the output with the results.
   Occur after step 1d, if it is asked in settings.

      i. Ground truth data is read by the system.
      ii. System compares results.
      iii. System display statistics.

Chapter 3

Bibliographic research

Optical Flow problem is an old subject in computer vision. The fundamental method for robust optical flow, referenced in over 10000 articles up to the newest, was published in 1980, by Horn and Schunch [3]. They define the optical flow as

"the distribution of apparent velocities of movement in an image."

and state the problem as a minimization of a squared penalty function from the brightness constraint and the smoothness constraint.

Based on this approach, many other algorithms were published. All this formulations, spatially-discrete, derived from Horn-Scunk are referred in literature as "clasics" [4, 5]. They all combine a data term and a spatial term, and apply an optimization procedure to minimize the function.

Another heavily cited article was published in the same year by Lucas and Kanade [6]. This approach solves the problem as a sum of squared differences, and proposes a coarse-to-fine solution for large displacements. As stated in chapter 3 from [7] the main weakness of this algorithm compared to Horn Schunk 's is the lack of regularization.

3.1 Problem formulation

The optical flow problem is, generally speaking, an optimization problem applied to a function which consists of two terms. One of them is the function to be minimized and the other is a correlation term called *diffusion term* which could correct or prevent errors (outliers) carried by the data term.

From a mathematical perspective we should point from the beginning one should major aspect. There are two kinds of data and diffusion terms from the isotropy perspective. The L^2 problem generally gives isotropic terms, as opposed to? L^1 which gives rise to anisotropic ones. The anisotropic property is very important, geometrically it means a direction dependence of the terms. Also the approach is more robust.

Examples of the data term and the correction term are shown in the next tables, as they are presented in [8].

Data Term	Constancy Assumption	Motion Type
$\Psi\left((\tilde{u}^T\tilde{\nabla}I)^2\right)$	brightness	any
$\Psi\left((\sum_{i=1}^{2}(\tilde{u}^T\tilde{\nabla}I_{\chi_i})^2)\right)$	gradient	translational, divergent, slow rotational
$\Psi\left(\sum_{i=1}^{2}\sum_{j=1}^{2}\left(\tilde{u}^T\tilde{\nabla}I_{\chi_i\chi_j}\right)^2\right)$	Hessian	translational, divergent, slow rotational
$\Psi\left(\left(\tilde{u}^T\tilde{\nabla}\lvert\nabla I\rvert\right)^2\right)$	gradient magnitude	any
$\Psi\left(\left(\tilde{u}^T\tilde{\nabla}\left(\Delta I\right)\right)^2\right)$	Laplacian	any
$\Psi\left(\left(\tilde{u}^T\tilde{\nabla}det\left(\mathcal{H}\left(I\right)\right)\right)^2\right)$	Hessian determinant	any

Data Term	Correction term
$S_1 = \sum_{i=1}^{2}\lvert\nabla u_i\rvert^2$	Homogeneous
$S_2 = g\left(\lvert\nabla I\rvert^2\right)\sum_{i=1}^{2}\lvert\nabla u_i\rvert^2$	image-driven, isotropic
$S_3 = \sum_{i=1}^{2}\left(\nabla u_i\right)^T D\left(\nabla I\right)\nabla u_i$	image-driven, anisotropic
$S_4 = \Psi\left(\sum_{i=1}^{2}\lvert\nabla u_i\rvert^2\right)$	flow-driven, isotropic
$S_5 = trace\Psi\left(\sum_{i=1}^{2}\nabla u_i\left(\nabla u_i\right)^T\right)$	flow-driven, anisotropic

3.2 First Approaches

Differential techniques compute the optical flow trough spatio-temporal derivatives. Most algorithms start from the brightness constraint and from a Taylor expansion, obtaining the gradient constraint. Details of the numerical scheme of this chapter will be furthered discussed in 4.1.1

Then, for a complete formulation of the problem, a regularization term is needed. In the first chapter of [9], the algorithms are classified in two categories, depending on the regularization term, variational and feature-based, meaning it does or does not depend on neighbouring pixels' flow computations.

3.2.1 Horn Schunk

The first important formulation of te global optical flow was stated by Horn and Schunk in [3]. They started from the brightness constraint, a moving point does not

change its brightness in time. From this constraint results the temporal derivative of the brightness of a sequence is 0. Then, using multi-dimensional Euler-Lagrange equations, from the expansion of the derivative, optical flow can be formulated in one equation as the data term. But the optical flow has two components, the horizontal flow, and the vertical flow. To solve this problem, another constraint is introduced, the spatial smoothness constrained.

Smoothness constraint is a variational method, that means it takes into account flow solutions of neighbourhood pixels. By assuming a continuous flow, the neighbouring pixels should have similar velocities[3]. Further, by applying the Laplacian operator on the flow, the result should be 0. By minimizing the Laplacian, the flow propagates on texureless objects.

By combining the two constraints, the results a convex energy function:

$$\iint ((\nabla I u + I_t)^2 + \lambda(\|(\nabla u)\|_2^2 + \|(\nabla v)\|_2^2))dxdy \tag{3.1}$$

where λ is the smoothness term coefficient. Although, λ is set to 100 in the by Horn and Schunk, in the original version, some later analysis of the algorithm claim that if the coefficient ot set to values down to 0.5 [10], the algorithm yields better results. This proves that there is no optimal general value for λ, it should be approximated for each different set of images.

To solve the optical flow, the sum energy function 3.1 must be minimized. In order to obtain this minimization, variational calculus is applied. A set of two equations is obtained for the field. Further, each vector is estimated with Gauss-Seidel iterations. This means that each point is computed from the anterior estimation. Gauss-Seidel iterations will be further discussed in 4.3.1

3.2.2 Lucas Kanade

Published in the same year with Horn and Schunk [3], the Lucas Kanade method[6], offers a local approach for solving the optical flow problem.

It is also based on the brightness constraint, but the solution is local. It is based on the least square fit. It assumes that the flow is more or less constant in a small neighbourhood.

$$\iint_{x\in\Omega} W^2(x)[\nabla I(x)\cdot \boldsymbol{v} + I_t(x)]^2 \tag{3.2}$$

Only small motion can be detected because of the Euler-Lagrange expansions, but the accuracy is at subpixel level. To obtain satisfactory results, a multilevel approach is considered. Usualy Gaussian pyramids are used, as proposed in the original article, [6], solving first the low resolution levels, based on which the higher resolutions are build.

3.3 Coarse-to-Fine

Differential methods work well when the motion is small, about 1-2 pixels, but on greater speeds, the algorithm fails, because the partial derivatives will not capture the motion. The coarse to fine method, as described by Lucas and Kanade in [6] allows computation of greater displacements. For each image is computed a Gaussian Pyramid. Usually [4], the standard deviation used for the Gauss anti-aliasing filter is

$$\frac{1}{\sqrt{2d}} \tag{3.3}$$

where d is the downsamplig factor.

The pyramid consists of the image in lower and lower resolutions, obtained from unsampling and filtering. Staked, with the higher resolution at the bottom and the lowest on the top, they look like a pyramid.

The flow is estimated between each level, from the bottom, then warped to the next, finer level. At each finer scale, the residual motion is computed between the first image and the second image warped to the first.

Pyramid Height About the height of the pyramid, it can vary from case to case. A general solution is to downsample until the top level has about 20-30 pixels in height or width [4].

Downsamplig Also, they say the downsamplig factor should be 0.5. most of the solutions take this value, but are some implementations that set the downsamplig factor at 0.8. There should not be any difference in the result as the minimization function is convex. A good practice is considered to set it at 0.5 [4].

Warping In [4], various methods are tested for wrapping. Their results state that good number for the warping times is 3. The difference in accuracy between 3 warps an 10 is insignificant.

Interpolation technoques Further, they compare the interpolation technique used before warping. The diference between them is not significantly high, but they found that spline-based bicubic interpolation yields slightly better results.

Drawbacks In chapter 15 of [11] the pyramid method is discussed. The author draws attention over the drawbacks of the coarse to fine methods. Each level's flow depends on its predecessor. If at some point in the computation of the velocities is erroneous, for example if aliasing or occlusions occur, it will be propagated up to the finest level.

3.4 Solving the Minimization Problem

Most of the formulation of the optical flow are stated as a minimization problem. If the penalty function is squared the solution can be achieved trough variational calculus. This applies when the energy function is differentiable. Then, most important, the function must be convex. Because of the convexity of the function, the minimum can be found Euler-Lagrange.

This is the case of the differential methods like Horn and Schunk. They first computed the partial derivatives for u and v, then to solve the system of equations they apply Jacobi iterative method. In this algorithm, in a iteration k each flow vector is computed from the results from the precious iteration $k-1$

Gauss-Seidel and Jacobi iterations are methods for solving linear equations with many unknowns. They start from one initial guess from which they iteratively compute the solution. the Gauss-Seidel solves for an upper dominant coefficient matrix, while the Jacoby method is used when the linear system is more symmetric. In their paper, Horn and Schunk used the Jacobi method, because the coefficient matrix is a sparse bended matrix.

Squared penalties are very sensitive to noises and are also isotropic. because of this the L^1 norm was used in the last years approaches.

For the L1 the classic methods do not apply any more.

3.5 L^1 Techniques

L^1 penalty functions were introduced in the optical flow energy function in the need for robustness and for the non isotropy property. They are more computational expensive, but they are more resistant to noise.

A example of how the errors throw off a L^1 vs a L^2 function is exemplified in figure 3.5. After values greater than one the difference is quite noticeable. As the function is a sum of errors from each vector of an image, the error function builds up and quickly. L^2 functions have the tendency to spread the error. Imagine, in a minimization function, as most of the vectors are 0, but there is one that is erroneous. It is obvious that the L^2 norm, let's sat sum of squares, will be way bigger than the L^1 norm, let's say absolute-value norm. As noisy images are quite common, a more robust formulation is needed.

Some of the algorithms use a mixed $L^2 - L^1$ norm. The argument for this is that the smoothness constraint is th one predisposed to outliers.

Algorithms studied in this thesis involve such a norm. They were implemented either with Proximal Point Projection or Least Mixed Norm. The detailed mathematical scheme is presented in the next chapter.

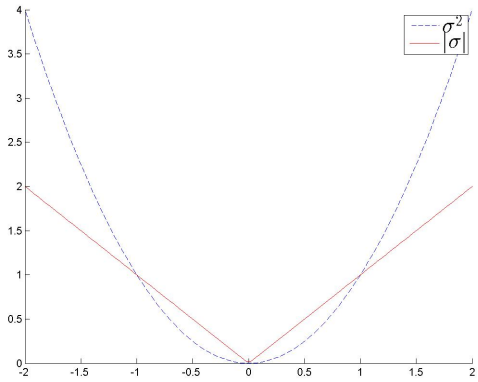


Figure 3.1: throw off of L^1 vs L^2 function

3.6 Improvements

Trough the decades, many algorithms were proposed with different ideas of solving the optical flow. Most of them are based a typical formulation like Horn and Schunks or Lucas Kanade, or a combinations of the two, adding a small improvement. The reliability and accuracy of the results increased in time, as did the time of computation, or the memory cost of the algorithms.

There are many suggested improvements, but there are some that proved to be very efficient and were adopted by most of the formulations.

They start by formulating the energy function based on th brightness constraint and a correcting term formulated with smoothness, rigidity constraint, or image segmentation.

Most of them solve the large displacement problem with a coarse to fine approach [12].

Median filters are us to increase robustness [13]. It may be used as a preprocessing technique, or in between the iteration. The second usage presents a particular interest in flow computation, especially in a coarse to fine approach. It is important to eliminate outliers as soon as possible. If not the propagate to the higher levels and spread in th neighbourhood with each iteration.

The penalty function tend to a L^1 norm [14, 15]. The classical squared penalty functions are replaced with the absolute value. The important properties of a penalty function is to have a small constant grow, to increase robustness of the algorithm, to be convex for easy calculations and to be non isotropic.

Some use a bilateral filter as weight for the correcting term[2]. The bilateral filter describe the pixel by its similarity with his neighbours. As the smoothness constraint propagates the flow to the neighbours, such a weight is important as when a pixel is similar to its neighbours we want the propagation, but on the other case, e.g. when a pixel is on the edge we want no influence from its neighbours, tu maintain edge accuracy.

Another improvement to the classical methods is expressing the data term. Some us a correlation transform, to increase robustness of the algorithm[16]. The term is described by the relation with the neighbours, not by its intensity. Another advantage in using the correlation transform is solving some of the occlusions, as new pixels can relay on their neighbours.

Census transform, [17] is also used in for describing the data term as a special type of correlation. The relation is encoded resulting in memory efficiency. Census methods are a little different of classical methods. They use special types of data structures on which they base their matching techniques. Despite the fact that Census transform does not have the best test results in the field it is used for their easy computation.

All of this improvements and their mathematical model will be discussed in detail in the next chapter.

Chapter 4

Analysis and Theoretical Foundation of existing algorithms

There are different approaches in computing the optical flow. From the fundamental methods, the field of study advanced tremendously in terms of speed and accuracy of solving the problem. Although highly improved results, the newer approaches are based on the same assumptions and start from the same point as the original formulation of Horn and Schunch, and Lucas and Kanade.

Although this algorithms are quite different, one being global formulation and the other a local one, modern approaches apply principles from both. The brightness consistency assumption is a good start point for many approaches, but it might be formulated in different form. Some algorithms use the Lukas Kanade approach style for a region based detection or feature tracking algorithms.

4.1 Differential Methods

The Horn and Schunk's solution is the basis of modern differential approaches. The optical flow is formulated variational problem. The critical points, minizors, must be found.

To avoid the aperture problem, the equation is formulated from two constraints. Originally, the brightness constraint and the smoothness constraint. In most algorithms from this category the brightness constraint is kept and the second constraint is modified.

4.1.1 The Brightness Constraint

The brightness constraint requires a Lambertian surface (ideally mate), is valid only under constant lightening, the main illumination source should be at such a distance that it will not change the brightness of the objects. Also secondary illumination should not exist, so no inter-object reflection can appear.

Also, a small motion is assumed from one frame to the next.

Although all its requirements which are almost never respected, at least not entirely, in real world, the brightness assumption has proven it's efficiency in may algorithms, being considered a good start point.

The brightness constraint assumes that the moving pixels of an image do not change their intensity in an image in time.

$$\frac{\partial I}{\partial t} = 0 \tag{4.1}$$

From this, using multi-dimensional Euler-Lagrange equations (see appendix B for full demonstration), we obtain

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0 \tag{4.2}$$

The optical flow is given by the x and y derivative in time. Considering this, let

$$u = \frac{dx}{dt} \quad , \quad v = \frac{dy}{dt} \quad and \quad \nabla I = \begin{bmatrix} I_x & I_y \end{bmatrix}^T \tag{4.3}$$

be the optical flow, $\boldsymbol{v}$ components. Denoting the temporal derivative with I_t, we obtain the gradient constraint.

$$\nabla I \cdot \boldsymbol{v} + I_t = 0 \tag{4.4}$$

4.1.2 The Aperture Problem

As a analogy, in motion perception, each retinal neuron captures only a small part of the visual field. This can be associated with seeing motion trough a small window, aperture. Because of the lack of context, uncertainty in speed, direction might appear. This uncertainty is called the aperture problem. The brain solves this ambiguity by composing a big picture from each neuron's perception, transforms a local problem in a global one, meaning that each computed velocity depends on the entire image.

As in our seeing mechanism, in optical flow detection, when a solution is computed locally, a uncertainty of the motion arises. Of course, it is more likely to have bigger uncertainties in special cases.

For example when there is no texture on the objects. As shown in figure 4.1.2, let the rectangle be the aperture, and the motion be given by the line moving from left to right. Let us consider the bolted point. the viewer can only say with certainty that the bolted dot will move from left to right, having no information about the movement on vertical, not the speed of the dot. The line can move either pure horizontally, slightly up, or slightly down. It is only when the viewer takes in account the end points of the line, he can state that the line is moving slightly up.

On the other hand, if the texture is to structured, it may also bring uncertainty to the solution. If we take for example a sinusoidal wave like in 4.1.2, there is no telling if the

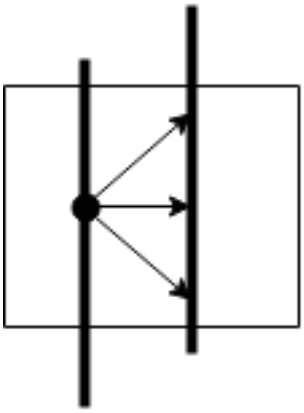


Figure 4.1: Example of aperture problem by the movement of a line.

Figure 4.2: Example of aperture problem by the movement of a sinusoidal signal.

signal moves up or down. As states in [10], the authors encounter problems when testing the solutions based on SSD on such input sequences. Because of the periodical signal, the algorithms found more than one solution. As they made the search window bigger, the more (ghost as they called it) local minimas the algorithm found.

In optical flow, the simplest solution is the brightness assumption, that neighbouring pixels have the same movement.

4.1.3 Lucas-Kanade

Lucas-Kanade method [6] solves the flow as a feature tracking algorithm. This differential method takes in account only the neighbourhood, solving for each pixel with last square estimation. From the brightness assumption, for a neighbourhood Ω, the flow vector $[u_i, v_i]$ must satisfy:

$$\begin{aligned} &I_x(x_1) + I_y(x_1) = -I_t \\ &I_x(x_2) + I_y(x_2) = -I_t \\ &\vdots \\ &I_x(x_n) + I_y(x_n) = -I_t \end{aligned} \tag{4.5}$$

where x_1, x_1, ..., x_n are in the neighbourhood of pixel i.

Using the least square method, for each pixel in the image the energy function can

be stated as:

$$E(x) = \sum_{x \in \Omega} W^2(x)[\nabla I(x) \cdot \boldsymbol{v} + I_t(x)]^2 \tag{4.6}$$

where W is a window function, Gaussian like, giving more weight to the canter pixel, rather than the neighbours.

To minimize $E(x)$, variational calculus is applied, and the derivatives of the equation 4.6 with respect to u an v, respectively will be 0.

$$\begin{aligned} \frac{\partial E}{\partial u} &= \sum_{x \in \Omega} W^2(x)[I_x^2 u + I_x I_y v + I_x I_t] = 0 \\ \frac{\partial E}{\partial v} &= \sum_{x \in \Omega} W^2(x)[I_x I_y u + I_y^2 v + I_y I_t] = 0 \end{aligned} \tag{4.7}$$

If we demote

$$\begin{aligned} A &= \Big[\nabla I(x_1),\ \ldots,\ \nabla I(x_n)\Big]^T \\ W &= diag\Big[W(x_1),\ \ldots,\ W(x_n)\Big] \\ b &= \Big[-I_t(x_1),\ \ldots,\ -I_t(x_n)\Big] \end{aligned} \tag{4.8}$$

for each n points $x_i \in \Omega$. Then, to minimize equation 4.6, we differentiate and the solution will be given by,

$$A^T W^2 A \boldsymbol{v} = A^T W^2 b \tag{4.9}$$

We multiplied the equation with A^T on the left hand side, make the matrix nonsingular and be able to compute it's inverse. The Flow vector will be equal with:

$$\boldsymbol{v} = [A^T W^2 A]^{-1} A^T W^2 b \tag{4.10}$$

where,

$$A^T W^2 A = \begin{bmatrix} \sum W^2(x) I_x^2(x) & \sum W^2(x) I_x(x) I_y(x) \\ \sum W^2(x) I_x(x) I_y(x) & \sum W^2(x) I_y^2(x) \end{bmatrix} \tag{4.11}$$

and

$$A^T W^2 b = \begin{bmatrix} -\sum W^2(x) I_x(x) I_t(x) \\ -\sum W^2(x) I_y(x) I_t(x) \end{bmatrix} \tag{4.12}$$

by simple calculations we obtain

$$\begin{aligned} u &= \frac{-\sum W^2 I_y^2 \ \sum W^2 I_x I_t + \sum W^2 I_x I_y \sum W^2 I_y I_t}{\sum W^2 I_x^2 \sum W^2 I_y^2 - (\sum W^2 I_x I_y)^2} \\ u &= \frac{\sum W^2 I_x I_t \ \sum W^2 I_x I_y - \sum W^2 I_x^2 \sum W^2 I_y I_t}{\sum W^2 I_x^2 \sum W^2 I_y^2 - (\sum W^2 I_x I_y)^2} \end{aligned} \tag{4.13}$$

One of the advantages of this technique is that the flow can be computed locally, without the need to rely on the entire image. In the case of a nonhomogeneous image, e.g. formed of small, textured objects, this algorithm gives good estimation.

On the other hand, when the elements of the image are big and textureless, when the image is homogeneous, this algorithm's performances will decrease. It will find the flow on the edges of the objects but it will fail to fill in homogeneous areas.

4.1.4 Correcting term

As in the Lucas Kanade method [6], the flow equation can be solved, neglecting the aperture problem, as a local solution. But, as the window is getting bigger, the flow vector will be influenced by more neighbours, and the aperture problem is felt more as the results are more noisy. Why would one try to enlarge the system of equations? Well, because put it in a context and everything can change. By connecting the pixels with their neighbours, all te pieces will be connected and the system can be viewed as an ensemble. The results get better by solving the aperture problems.

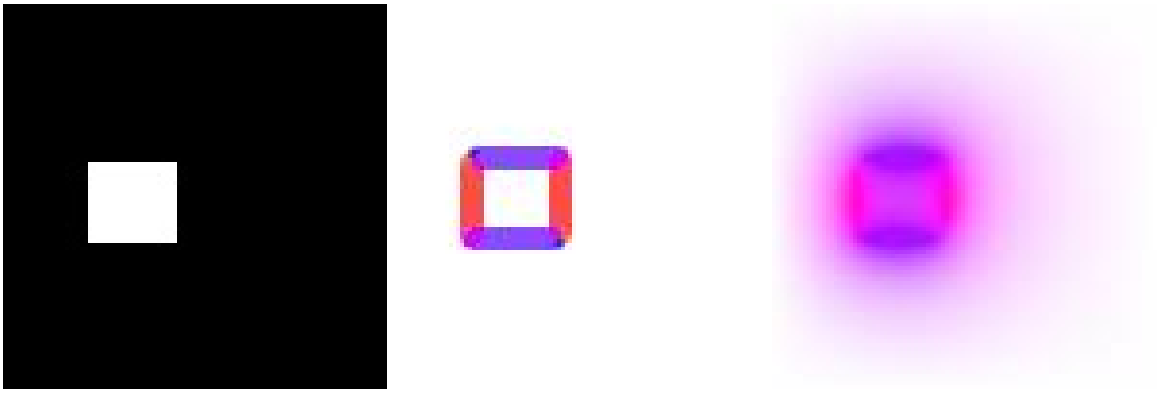
For example, as stated before, on surfaces with smooth, or no texture, the flow cannot be computed with a local method. But, if for a pixel in a textureless area, the flow is computed taking in account its neighbours, flow information will be propagated to it. In image 4.1.4, it is shown the flow output for a square moving right and upwards. It can be seen the propagation of the flow between iterations 3 and 100.

In addition to the brightness constraint, Horn and Schunk take a second term in equation, the smoothness constraint. The value of a pixel will be probably very close to its neighbours. It is probably the simplest regularizator.

$$E_{smooth} = \iint (\|\nabla u\|_2^2 + \|\nabla v\|_2^2) dx dy \tag{4.14}$$

where

$$\nabla = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{4.15}$$



(a) (b) (c)

Figure 4.3: In 4.3a is the square on which the propagation was tested. in 4.3b and 4.3c , example of flow propagation from iteration 3 to 100

which means if the flow differs from a pixel to another (on x and y directions), the smoothness equation, 4.14 will be high. As the algorithm tries to minimize the energy function3.1, the second term, 4.14, will also be minimized. If this term is closer to zero, the flow derivatives over x and y are smaller, which means the difference with the neighbours is also smaller, in other words the flow is smooth.

Nagel, in [18] proposes another smoothing term, in which he introduces also the gradient. It is based on the idea that errors often occur over the edges and the gradient has its greatest value on the edges. As it takes the gradient with the flow, the term is anisotropic.

$$\frac{\alpha^2}{\|\nabla I\|_2^2 \delta}[(u_x I_y - u_y I_x)^2 + (v_x I_y - v_y I_x)^2 + \delta(u_x^2 + u_y^2 + v_x^2 + v_y^2)] \tag{4.16}$$

The smooth term is more or less taken in account, depending on the nature of the input sequence. On some it works better with a higher weight, or in some, lower. Probably the best approach is to vary λ weight of the second term in the flow equation 3.1, and compare the results, like in a training stage of the program. Usually λ is less than 1 to avoid over smoothing.

$$E = E_{data} + \lambda E_{smooth} \tag{4.17}$$

To avoid this overall smoothing, as pixels may not be acting as a whole, λ could be a matrix instead of a term, so each pixel can be influenced in his own percent of the neighbours. An solution in computing this weight can be taken form the bilateral filter. The weight can be computed as explained in 4.4.3, so the smoothing term will influence each pixel differently. As when the pixel is on the edge, there should not be any influence from the neighbours, otherwise resulting in noisy results on the edges; but when the pixel is in the middle of an object, the velocity should be influenced by its neighbours.

4.2 Coarse-to-fine

Differential methods cannot approximate large displacements. This is due to the small motion assumption from the data term. The data term is expanded with Euler-Lagrange equation and then expressed as a sum of partial derivatives. To discretize this derivatives a small step is necessary, usually of one pixel. This further means that motion larger than the with of the derivative window will not be detected. Also the kernel cannot be wider than one pixel on each side, because the small step requirement of the derivative.

This problem can be solved with coarse-to-fine method. For example, in the original Horn Schunk formulation, the algorithm contains no Coarse to fine strategy. But, when this approach is added to the implementation the error of the results drop significantly. As it can be seen in the images in 4.2 When the flow is computed with the Coarse-to-fine approach the result are better. for this example in particular, the mean angular error without the pyramidal approach is 30.86 and after is decreased to 15.94

Firstly, Gaussian Pyramids are built by successively blurring and unsampling into images of smaller and smaller resolutions. The unsampeling is done until the smallest resolution is about 30 pixels width or height. Then, the flow is iteratively computed between each level of the pyramid, from the corest to the finest.

4.2.1 Gaussian Pyramids

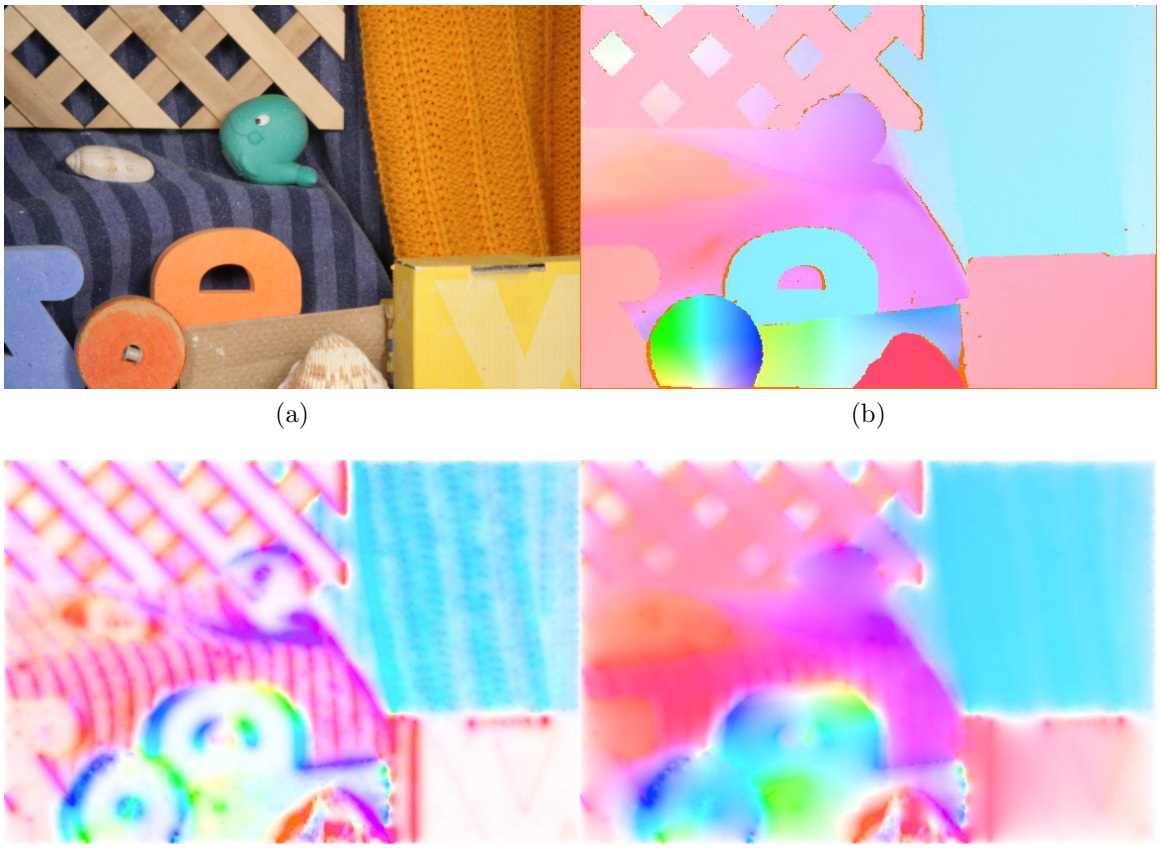
The pyramid of an image is a multi-scale representation of it. In order to obtain the pyramids, successive smoothing and subsampling techniques are applied. Each level is computed from the previous, recursively.

Building The pyramid Let us consider an image I of size $m \times n$. The first level of the pyramid, the base is the image, I_0 is the image I, itself. The next level, I_1, is computed from I_0. The image I_0 is convoluted with a Gauss kernel, than the filtered image is sampled. I_1 is obtained with the dimensions $(m_0/samplingfactor \times n_0/sampligfactor)$.

Similarly, the next levels are obtained, the I_2 is computed from I_1, I_3 from I_2, and so on, as illustrated in picture.

Usually a sampling factor of 0.5 is considered. If we consider L the current level of the image, than we can obtain the I_{L+1} image as fallows

$$\begin{aligned} I_{L+1}(x,y) =& \frac{1}{4} I_L(2x,2y) + \\ & \frac{1}{8}(I_L(2x-1,2y) + I_L(2x+1,2y) + I_L(2x,2y-1) + I_L(2x,2y+1)) + \\ & \frac{1}{16}(I_L(2x-1,2y-1) + I_L(2x+1,2y+1) \\ & + I_L(2x-1,2y+1) + I_L(2x+1,2y-1)) \end{aligned} \tag{4.18}$$



(a) (b)

(c) (d)

Figure 4.4: Example of the improvement the coarse to fine approach gives an algorithm. In picture 4.4a is one of the 2 images of the sequence on which the flow is to be computed. In 4.4b is the flow according to Middlebury. In 4.4c is the flow computed with classical Horn and Schunk algorithm. And in 4.4d is the same algorithm which was encapsulated in a coarse to fine strategy.

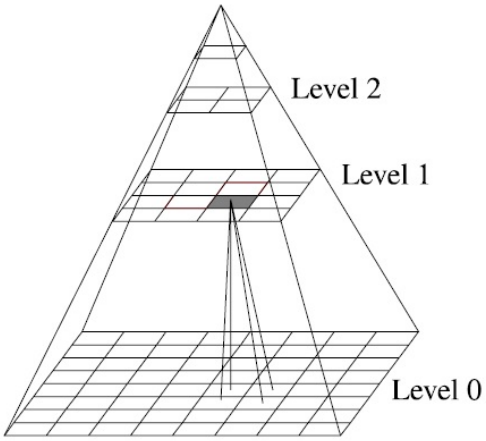


Figure 4.5: visual representation of a pyramid. Taken from [9]

The result of such computation is shown in figure 4.2.1.

Choosing the pyramid height Choosing the number of levels of aa pyramid depends of the nature of motion in the frame sequence and the downsampling factor. As stated in the chapter before 3.3, the height of the pyramids is usually chosen to have around 20-30 pixels on the height of the image, or width, respectively. The height is given by

$$pyramid_{height} = \frac{\log\left(\frac{p}{\min(ht,wt)}\right)}{\log(d)} \tag{4.19}$$

where p is the number of pixels on the top level on the minimum between width wt or the height ht.

4.2.2 Warping the flow on the image

At new level of the pyramid, the second image is warped to the first. The new computed flow is considered. Let us consider the current level l, and the computed flow $\boldsymbol{v}_l$, on the next level $l+1$, the second image is warped with the $\boldsymbol{v}_l$ velocities from the previous level. At a certain level $l+1$ the warped image is

$$I_w = interpolation(I_l, w_l + dw) \tag{4.20}$$

where w_l is the flow computed until the level l and dw the residual flow computed at level l. Some algorithms, like proximal point approximation, do not compute the residual flow, but update directly the new flow.

Figure 4.6: Example of a pyramid with 4 levels.

4.3 L^2 Solutions

First formulations were in the L^2 norm. For this some classical approaches are used. As the function is convex and derivable, the solution of the optimization problem is the solution of the associated Euler-Lagrang equations. First compute the partial derivatives, then equate them to 0. This are simple solutions, that run fast and don't consume to much memory. But this methods are very sensitive to noise.

4.3.1 Gauss-Seidel Iterations

Gauss Seidel is a iterative method for solving simultaneous equations **??**. If we consider the system

$$A\boldsymbol{x} = b \tag{4.21}$$

where x is the unknown, the matrix A can be decomposed in a sum of 2 matrices, the lower triangular and the upper.

$$\begin{aligned} A &= L_* + U \\ A &= \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ 0 & 0 & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 \end{bmatrix} \end{aligned} \tag{4.22}$$

The equation becomes:

$$\begin{aligned}(L_* + U)\boldsymbol{x} &= b \\ L_*\boldsymbol{x}^{k+1} &= b - U\boldsymbol{x}^k\end{aligned} \tag{4.23}$$

The algorithm takes a guess for the first step k of the iteration. The next steps are solved using the previous approximation and the solutions found up to the current point from the latest iteration. The algorithm stops when the solution converges, the iteration does not change the result significantly.

The Horn Schunk equation 3.1, can be minimized using the Gauss Seidel Equations. First, we need to derive by u and v:

$$\begin{aligned}\frac{\partial E}{\partial u} &= I_x(I_x u + I_y v + Iy) + \lambda(\bar{u} - u) \\ \frac{\partial E}{\partial v} &= I_y(I_x u + I_y v + Iy) + \lambda(\bar{v} - v)\end{aligned} \tag{4.24}$$

Notice the substitution of the term ∇u^2 with $(\bar{u}-u)$, this is done like a Laplacian transform by unwrapping the ∇ line in equation 4.15. A matrix will be: $A_{2*i+1,2*i+1} = I_{xi}^2 + \lambda$, $A_{2*i,2*i} = I_y^2 + \lambda$, $A_{2*i+1,2*j} = I_{xi}I_{yi}$, $A_{2*i,2*j+1} = I_{xi}I_{yi}$, $A_{2*i+1,2*j+1} = A_{2*i,2*j} = \lambda$, where $j \in N_i$, by solving the equation for $\boldsymbol{x}$, we obtain the iterations:

$$\begin{aligned}u_i^{k+1} = \frac{1}{I_{xi}^2 + I_{yi}^2 + \lambda}((I_{yi}^2 + \lambda)(&\sum_{j\in\mathcal{N}_i;j<i} u_j^{k+1} + \sum_{j\in\mathcal{N}_i;j>i} u_j^k) \\ &- I_{xi}I_{yi}(\sum_{j\in\mathcal{N}_i;j<i} v_j^{k+1} + \sum_{j\in\mathcal{N}_i;j>i} v_j^k) \\ &- I_{xi}I_{ti}) \\ u_i^{k+1} = \frac{1}{I_{xi}^2 + I_{yi}^2 + \lambda}(-I_{xi}I_{yi}(&\sum_{j\in\mathcal{N}_i;j<i} u_j^{k+1} + \sum_{j\in\mathcal{N}_i;j>i} u_j^k) \\ &+ (I_{yi}^2 + \lambda)(\sum_{j\in\mathcal{N}_i;j<i} v_j^{k+1} + \sum_{j\in\mathcal{N}_i;j<i} v_j^k) \\ &- I_{yi}I_{ti})\end{aligned} \tag{4.25}$$

4.3.2 Jacobi Iterations

Like Gauss Seidel, the Jacobi method solves a system of linear equations. It is generally used when the system is diagonally dominant.

If we consider the system

$$A\boldsymbol{x} = b \tag{4.26}$$

where x is the unknown, the matrix A can be decomposed in a sum of 2 matrices, the principal diagonal and the rest of the elements.

$$
\begin{aligned}
A &= D + E \\
A &= \begin{bmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{nn} \end{bmatrix} + \begin{bmatrix} 0 & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & 0 \end{bmatrix}
\end{aligned} \tag{4.27}
$$

The equation becomes:

$$
\begin{aligned}
(D + E)\boldsymbol{x} &= b \\
D\boldsymbol{x}^{k+1} &= b - E\boldsymbol{x}^{k}
\end{aligned} \tag{4.28}
$$

The Jacobi iterations are used by Horn and Schunk in solving the minimization of the energy function. As can be observed, the method finds each x from all it's previous guess but itself. Applied on Horn and Schunk's partial derivatives, the solution can be expressed as:

$$
\begin{aligned}
u_i^{k+1} &= \frac{1}{I_{xi}^2 + I_{yi}^2 + \lambda} \left((I_{yi}^2 + \lambda)(\sum_{j \in \mathcal{N}_i} u_j^k) - I_{xi} I_{yi} (\sum_{j \in \mathcal{N}_i} v_j^k) - I_{xi} I_{ti} \right) \\
u_i^{k+1} &= \frac{1}{I_{xi}^2 + I_{yi}^2 + \lambda} \left(-I_{xi} I_{yi} (\sum_{j \in \mathcal{N}_i} u_j^k) + (I_{yi}^2 + \lambda)(\sum_{j \in \mathcal{N}_i} v_j^k) - I_{yi} I_{ti} \right)
\end{aligned} \tag{4.29}
$$

Notice the difference between Gauss-Seidel and Jacobi method. The first one uses the solution for x as soon as is available, while the Jacobi iterations relay only on the previous approximation.

The math between equation 4.24 and the iteration equations 4.25 and B.8 can be found in Appendix B B.1.

4.4 Improvements

As the algorithms are based on the same formulations of Horn and Scunk and Lucas Kanade, algorithm's performance evolve in time by small transformations and changes added to the classical algorithms. Some of them have proven high efficiency, that were adopted as a standard by fallowing algorithms.

4.4.1 Low Pass Filter

As stated above, most of the algorithms use a coarse to fine estimation, and, for each level, an iterative approximation is performed. One of the downside of this approach is that if any outliers are present in the for at a lower level, this error will be propagated to all finer levels up to the final result, damaging the overall outcome of the algorithm.

In the papers comparing some of the existing algorithms, the implementations take a median filter somewhere in the algorithm.

In [11], the problem of aliasing is considered for bigger displacements, in the sampling procedure. The authors solve this problem by applying a blurring filer over the images, before computing the gradients of the images. From this they apply estimate the velocities for each level.

Also, in [4], by analysing the best practices in solving the optical flow problem, it is proven that, if a median filter applied after each warp, the robustness and the accuracy of the algorithms is increased. The robustness is a weak point of the differential methods. Outliers can completely throw off the energy function, especially in L^2 methods. Their results compare different sizes of the median filter, and no filer of different algorithms. The best results are obtained with a 5×5 kernel. Also the accuracy of the results when no filter was applied is significantly lower.

4.4.2 Cross Correlation

The cross correlation function measures the similarity between 2 signals. The normalized cross correlation takes values in $[0, 1]$, where 1 is a total match between the signals and 0 is a total mismatch between them.

It is used in computing the data term, as a function derived from the brightness constraint. The pixels are not described anymore by their intensity, but by their relation with the neighbours.

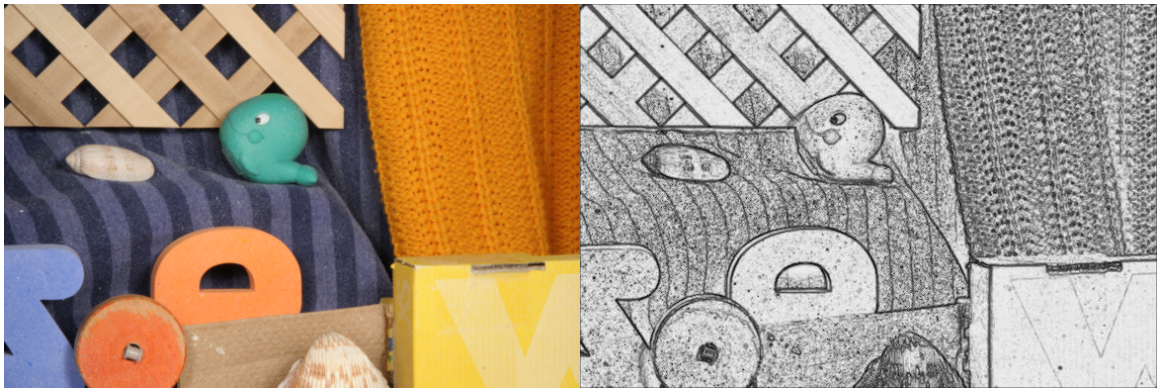
In [2], the classical sum of squared difference, the data term is expressed as a zero normal cross correlation, this being more discriminative.

$$C(i) = \frac{I(s) - \mu(i)}{\sigma(i)} \tag{4.30}$$

As shown above, each point of the image, if passed trough a cross correlation transom, from which results a matrix of the same size as the neighbourhood considered. to find the minimum, the data term is further expanded with classic differential methods it is added the smoothness term.

One of the advantages in using the cross correlation transform instead of the regular pixel intensity, is the steadiness to noise. As a pixel is described by it's neighbourhood it will not be as exposed to small noise.

Also occlusions in the image are resolved from the correlation transform. When parts of the environment are reviled from a frame to another, the pixels capturing this will



(a) (b)

Figure 4.7: Example of the bilateral filter transform. 4.7a is the original image, on which the formula 4.31 was applied. The result is shown on the right in 4.7b

be described by their neighbours and the flow will be normally computed in this areas.

4.4.3 Bilateral Filter

The bilateral filter is a smoothing filter, but the accuracy along edges is kept. As proposed in [19], the filter has two elements. The geometric distance, from the classical lowpass filter, let's take Gaussian for example. Each neighbour pixel will influence the output proportionally with its distance to the current pixel. On the other hand, the chromatic component measures the photochromacy between the neighbours and the central pixel. This can be expressed as the difference between the intensity of the neighbour pixel and the centre.

In the Gaussian case, one could get the normalization term, which is

$$e^{-\left(\frac{\Delta_c^2(i,s)}{2\sigma_c^2}+\frac{\Delta_d^2(i,s)}{2\sigma_d^2}\right)} \tag{4.31}$$

In a L^1 context, better results are yield by the a Laplacian formulation:

$$e^{-\left(\frac{|\Delta_c(i,s)|}{\sqrt{2}\sigma_c}+\frac{|\Delta_d(i,s)|}{\sqrt{2}\sigma_d}\right)} \tag{4.32}$$

This normalization term can measure the likelihood of being on the edge of a pixel. An exaple of the bilateral filter transform is given in 4.4.3. It can be observed in the left figure, 4.7b, the brighter areas on the planes of the object indicate a value near to 1, and the darker areas, indicating 0 are present on the edges of the objects.

In [2], this formulation of the bilateral filter is used as a component of smoothness term. The smoothness constraint says that, the velocity of the neighbour pixels is the same. This applies, of course only if the neighbour pixels belong to the same object. The

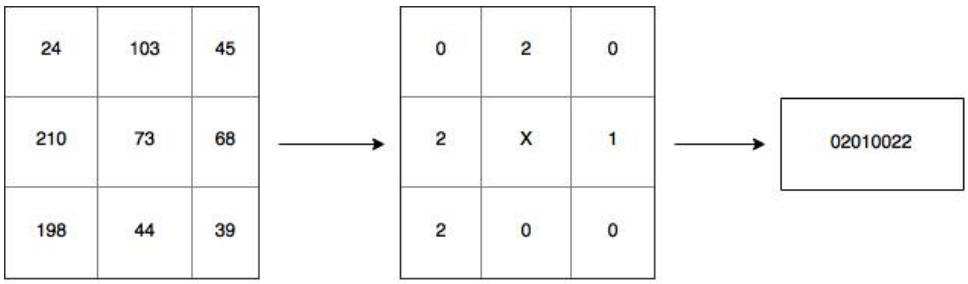


Figure 4.8: Census Transform example to signature vector

classic methods based on the smoothness constraint have large errors along the edges. By considering the bilateral filter term 4.31, one can easily reduce the smoothness penalty for velocities that do not belong to the same object, by simply applying the dot product between the bilateral filter term and the first norm of the difference.

4.4.4 Census transform

In Census transform [17], the intensities of pixels are mapped in a string of bits. The string is formed based on the relation of the current pixel to its neighbours. Considering a 3×3 window, the centre is compared with its neighbours after the function:

$$\mathcal{C}(x, x') = \begin{cases} 0 & \text{if } I(x) - I(x') > th \\ 1 & \text{if } |I(x) - I(x')| \leq th \\ 2 & \text{if } I(x) - I(x') < -th \end{cases} \tag{4.33}$$

where $I(x)$ is the intensity of the pixel x and x' is the neighbour. After the neighbour matrix was transformed, it is unwrapped, usually from the upper left corner, clockwise. Let us take an example. In figure 4.4.4, the Census transform is applied 4.33, with a threshold of 10.

There must be a balance when choosing the threshold, between a value that is to high and causes insensitivity to light chances and a value that is too small where the uniqueness of the signature bit is affected.

The transform is applied on each pixel of a image. The image will be described by a series of signature vectors. Each of this vectors are stored in a Hash-Table together with their coordinates. Then the second image is transformed too. A matching is done between the signatures of the second image and the signatures in the table, resulting in a list of possible correspondences for each entry or a null if the signature has no match. In case of multiple match, the pixels' intensities are compared or, if there is still no unique solution, the closest match, distance wise, is taken.

An advantage of this solution is that the algorithm can detect large displacements, like objects jumping from one side to another of the image, unlike classic differential methods.

Due to the nature of the algorithm, it can be implemented on parallel hardware, like FPGAs, allowing for real-time computation.

But because of the transform, and the threshold, ignoring pixels with low variance, there is important loss in accuracy.

4.5 L^1 approach

When the penalty function, $E(x)$ is of type L^2, like $\sigma^2(x)$, the minimization problem is simple. One can reach the solution trough variational calculus, by Euler-Lagrange equations. For a higher robustness, current algorithms tend to use a L^1 penalty function, like modulus, others use a combination of both L^2 and L^1 penalty functions, for example, one for the data energy and one for the smoothness energy.

For this kind of formulations, the minimization can not be done any more trough differential techniques, and other approaches are used.

The smoothness term is the on that is more likely to give erroneous results, as the most outliers are present in the edge areas. Further will be presented two methods studied for solving thee L^1 norm. Both of the have only the correction term in L^1, the first remaining in L^2.

4.5.1 Projected Proximal Point

Proximal point algorithms are used in minimizing a sum of a convex function and a L^2 norm [20]:

$$prox_p(x) = arg\min_y \frac{1}{2}\|x-y\|_2^2 + P(y)$$

The optimization method is presented as in [2], as an adaptation to the optical flow problem. First let us consider the function to be minimized:

$$\min_{\boldsymbol{v}}\{\lambda \cdot E_d(\boldsymbol{v}) + F(K\boldsymbol{v})\}$$

In this function, $\boldsymbol{v}$ is th unknown and $E_d(\boldsymbol{v})$ is the L^2 function. $F(K\boldsymbol{v})$ must be taken in such way that the function to be minimized stays convex in $\boldsymbol{v}$. Now to optimize this problem let us use a dual approach by replacing F with a concave function.

$$\min_{\boldsymbol{v}}\{\lambda \cdot E_d(\boldsymbol{v}) + [\max_q\{\langle K\boldsymbol{v}, qrangle - F^*(q)\}]\}$$

where

$$F(q) = \begin{cases} 0 & \text{if } q \in Q \\ \infty & , \ otherwise \end{cases} \tag{4.34}$$

and Q the definition domain for q is

$$Q = \begin{cases} q \in R^{2\times|\Omega|\times\mathcal{N}} : q_{i,s} = qu_{i,s}, qv_{i,s} \\ \|qu_{i,s}\| \leq b_{i,s}, \|qv_{i,s}\| \leq b_{i,s}, \forall i \in \Omega, \forall s \in \mathcal{N}_i \end{cases} \tag{4.35}$$

where $b_{i,s}$ is the coefficient of $F(y) = b_{i,s}y$. We can say that the 4.5.1 function is convex in $\boldsymbol{v}$ and concave in q. The algorithm can be written as a gradient descent for $\boldsymbol{v}$ and a gradient ascent for q. As the min and max functions can be combined and swapped, the 4.5.1 becomes

$$\min_{\boldsymbol{v}} \max_{q} \{\alpha E_d(\boldsymbol{v}) + \langle K\boldsymbol{v}, q\rangle - F^*(q)\}$$

we can now build the function

$$\Phi(\boldsymbol{v}, q) = \alpha E_d(\boldsymbol{v}) + \langle K\boldsymbol{v}, q\rangle - F^*(q) + \frac{1}{2\tau}\|\boldsymbol{v} - \boldsymbol{v}^k\|^2 - \frac{1}{2\eta}\|q - q^k\|^2$$

where $\boldsymbol{v}^k$ and q^k are values from the previous estimation, and τ and η ar control parameters. From this function we can compute iteratively the primal-dual variables, by considering one fixed and updating the other one.

$$\begin{cases} \boldsymbol{v}^{k+1} = arg\min_{\boldsymbol{v}} \Phi(\boldsymbol{v}, q^k) \\ q^{k+1} = arg\max_{q} \Phi(\boldsymbol{v}^{k+1}, q) \end{cases}$$

Φ is differentiable, therefore the gradient can be computed to tu find the function's minimum.

$$\begin{cases} \frac{\partial\Phi}{\partial\boldsymbol{v}}(\boldsymbol{v}, q^k) = \alpha\frac{\partial E_d}{\partial\boldsymbol{v}} + K^T q + \tau^{-1}(\boldsymbol{v} - \boldsymbol{v}^k) \\ \frac{\partial\Phi}{\partial\boldsymbol{v}}(\boldsymbol{v}^{k+1}, q^k) = K\boldsymbol{v}^{k+1} - \eta^{-1}(q - q^k) \end{cases}$$

Now, to satisfy the constraints from 4.35, we need to project the q solution into its domain.

$$Pr(q) = (\frac{qu}{\max(|qu|, b)} \cdot b, \frac{qv}{\max(|qv|, b)} \cdot b)$$

To solve the system the algorithm goes trough a series of iterations. In each iteration,

it updates the $\boldsymbol{v}$ and q variables one at a time from 4.5.1. After each update of q, it is projected into its domain.

4.5.2 Least Mixed Norm

The Least Mixed Norm (LMN) solution was also approached image restoration problems, belonging from the family of interior point algorithms. The objective is to minimize a $L^2 - L^1$ function:

$$\min_{\boldsymbol{f}} \|g - H\boldsymbol{f}\|_2^2 + \alpha\|R\boldsymbol{f}\|_1 \tag{4.36}$$

Basically this approach uses the interior point method. One of th main benefits of this approach is that it can be applied on a large number of algorithms. Other benefit is that, despite of its great dimension, the problems are sparse and banded, so many fast solving procedures can be applied.

In [21], the problem is explained as a image restoration solution, but as the general function, 4.36, matches the optical flow total energy function, this solution can be applied in the optical flow computation, considering $\boldsymbol{f}$ the pair of field vectors (u, v).

Let us now discuss the solution presented in [21].

First le us write the data term, as a sun of it's positive and negative values: $\alpha R\boldsymbol{f} = \boldsymbol{v}^+ + \boldsymbol{v}^-$, where $\boldsymbol{v}^+ = \max(\alpha R\boldsymbol{f}, 0)$, and $\boldsymbol{v}^- = \max(-\alpha R\boldsymbol{f}, 0)$. Considering $\mathbf{1}$ a column of ones, the function becomes:

$$\min_{\boldsymbol{f}} \|g - H\boldsymbol{f}\|_2^2 + \mathbf{1}^T\boldsymbol{v}^+ + \mathbf{1}^T\boldsymbol{v}^- \tag{4.37}$$

The function becomes:

$$\min_{\boldsymbol{x}} \frac{1}{2}\boldsymbol{x}^T G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} \quad \text{subject to} \quad A\boldsymbol{x} = \boldsymbol{b} \tag{4.38}$$

where

$$G = \begin{bmatrix} 2H^TH & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad A = \begin{bmatrix} \alpha R & -I & I \end{bmatrix}$$
$$\boldsymbol{b} = \mathbf{0}, \quad \boldsymbol{x} = \begin{bmatrix} \boldsymbol{f} \\ \boldsymbol{v}^+ \\ \boldsymbol{v}^- \end{bmatrix}, \quad \text{and} \quad \boldsymbol{c} = \begin{bmatrix} -2H^T\boldsymbol{g} \\ \mathbf{1} \\ \mathbf{1} \end{bmatrix} \tag{4.39}$$

We can now write the Lagrange equation:

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s}) = \frac{1}{2}\boldsymbol{x}G\boldsymbol{x} + \boldsymbol{c}^T\boldsymbol{x} - \boldsymbol{\lambda}^T(A\boldsymbol{x} - b) - \boldsymbol{s}^T\boldsymbol{x} \tag{4.40}$$

where λ is the generalized Lagrange multiplier for the $A\boldsymbol{x} = \boldsymbol{b}$ constraint. Let X and S be sparse matrices with the diagonal elements from $\boldsymbol{x}$ and $\boldsymbol{s}$ unwrapped. The equation can be rewritten as:

$$F(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{s}) = \begin{bmatrix} G\boldsymbol{x} + \boldsymbol{c} - A^T\boldsymbol{\lambda} - \boldsymbol{s} \\ A\boldsymbol{x} - \boldsymbol{b} \\ XS\mathbf{1} \end{bmatrix} = 0 \tag{4.41}$$

where the central path is

$$F(\boldsymbol{x}_{\sigma\mu}, \boldsymbol{\lambda}_{\sigma\mu}, \boldsymbol{s}_{\sigma\mu}) = \begin{bmatrix} 0 \\ 0 \\ \sigma\mu\mathbf{1} \end{bmatrix} \tag{4.42}$$

where $\sigma \in (0, 1)$ and μ is the arithmetic mean of $\boldsymbol{f}$. Rewriting using Newton step, the system becomes:

$$\begin{bmatrix} G & -A^T & -I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{\lambda} \\ \Delta \boldsymbol{s} \end{bmatrix} = \begin{bmatrix} -\boldsymbol{r_c} \\ -\boldsymbol{r_b} \\ -\boldsymbol{r_a} \end{bmatrix} \tag{4.43}$$

where

$$\boldsymbol{r}_c = G\boldsymbol{x} + \boldsymbol{c} - A^T\boldsymbol{\lambda} - \boldsymbol{s},\ \boldsymbol{r}_b = A\boldsymbol{x} - \boldsymbol{b} \text{ and } \boldsymbol{r}_a = XS\mathbf{1} - \sigma\mu\mathbf{1} \tag{4.44}$$

to reduce by Δs

$$\begin{bmatrix} G + X^{-1}S & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ \Delta \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\hat{\boldsymbol{r}}_c \\ -\boldsymbol{r_b} \end{bmatrix} \tag{4.45}$$

where $\hat{\boldsymbol{r}}_c = \boldsymbol{r}_c + X^{-1}\boldsymbol{r}_a$. Making the system symmetric:

$$\begin{bmatrix} G + X^{-1}S & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{x} \\ -\Delta \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\hat{\boldsymbol{r}}_c \\ -\boldsymbol{r_b} \end{bmatrix} \tag{4.46}$$

Let $D = S^{-1/2}X^{1/2}$, the let D be decomposed in equal partitions over the diagonal. $D = diag(D_1, D_2, D_3)$. Then the system can be unwrapped as:

$$\begin{bmatrix} 2H^TH + D_1^2 & 0 & 0 & \alpha R^T \\ 0 & D_2^2 & 0 & 0 \\ 0 & 0 & D_3^2 & 0 \\ \alpha R & -I & I & 0 \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{f} \\ \Delta \boldsymbol{v}^+ \\ \Delta \boldsymbol{v}^- \\ \Delta \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\hat{\boldsymbol{r}}_{c1} \\ -\hat{\boldsymbol{r}}_{c2} \\ -\hat{\boldsymbol{r}}_{c3} \\ -\boldsymbol{r_b} \end{bmatrix} \tag{4.47}$$

where $-\hat{\boldsymbol{r}}_c1$, $-\hat{\boldsymbol{r}}_c2$, and $-\hat{\boldsymbol{r}}_c3$ are the appropriate vectors of $-\hat{\boldsymbol{r}}_c$, by removing $\Delta \boldsymbol{v}^+$ and $\Delta \boldsymbol{v}^-$ we obtain:

$$\begin{bmatrix} 2H^TH + D_1^2 & \alpha R^T \\ \alpha R & -D_2^2 + -D_3^2 \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{f} \\ \Delta \boldsymbol{\lambda} \end{bmatrix} = \begin{bmatrix} -\hat{\boldsymbol{r}}_{c1} \\ -\hat{\boldsymbol{r}}_b \end{bmatrix} \tag{4.48}$$

where $\hat{\boldsymbol{r}}_b = -\boldsymbol{r_b} + D_2^2\hat{\boldsymbol{r}}_{c2} - D_3^2\hat{\boldsymbol{r}}_{c3}$ and finally, eliminating $\Delta\boldsymbol{\lambda}$,

$$\left[2H^TH + D_1^{-2} + \alpha R^T(D_2^2 + D_3^2)^{-1}R\right]\Delta\boldsymbol{f} = -\tilde{\boldsymbol{r}}_{c1} \tag{4.49}$$

where $\tilde{\boldsymbol{r}}_{c1} = \hat{\boldsymbol{r}}_{c1} + \alpha R^T(D_2^2 + D_3^2)^{-1}\hat{\boldsymbol{r}}_b$

The system will be solved for $\boldsymbol{f}$ with equation 4.49, and then the other unknowns can be computed:

$$\begin{aligned}\Delta\boldsymbol{\lambda} &= (D_2^2 + D_3^2)^{-1}(-\hat{\boldsymbol{r}}_b - \alpha R\boldsymbol{f})\\ \Delta\boldsymbol{v}^+ &= D_2^2(-\hat{\boldsymbol{r}}_{c2} - \boldsymbol{\lambda})\\ \Delta\boldsymbol{v}^- &= D_3^2(-\hat{\boldsymbol{r}}_{c3} + \boldsymbol{\lambda})\\ \Delta\boldsymbol{s} &= G\boldsymbol{x} - A^T\boldsymbol{\lambda} + \boldsymbol{r}_c\end{aligned} \tag{4.50}$$

Now the system will be approximated iteratively. A initial guess is taken from the condition $A\boldsymbol{x} = b$

4.6 Error Measurement

In this chapter we focus towards the task of error measurement. We look at several techniques for finding the computation errors, briefly describing each approach.

Now that the problem is stated, in order to get an idea of which behaves better on what conditions, we can compare the flow results with a ground truth.

Middlebury's website[22] proposes a challenging data sheet containing some of the top algorithms in optical flow, as they state in [23], by which they hope to encourage improvement. In [22], a large database of the existing algorithms and their performance can be found. They are classified by accuracy. They have available a set of images on which the algorithms can be tested and measure their performance. Both realistic high speed camera and synthetic images can be downloaded from their database. For the artificially generated also the exact flow is available. This stands as a reference for the output of the algorithms. The error can be computed between this ground truth output and the result from the algorithms.

Being referenced in most of the latest articles, it seems that this database is a good reference to modern approaches and their metrics.

4.6.1 Cross correlation

Cross correlation is a simple way to measure the difference between 2 signals. In image processing it is used as a normalized form.

$$\frac{1}{n}\sum_{i,j}\frac{(f(i,j)-\bar{f})(f_{GT}(i,j)-\bar{f_{GT}})}{\sigma_f\sigma_{GT}} \tag{4.51}$$

The results of this function will vary in $[0,1]$, 1 meaning a total match and 0 a total dismatch.

4.6.2 Endpoint error

For measuring this error, the length of the vectors is taken in account. This error is expressed as the sum of differences both on the x and y directions, stated in a L^2 or L^1 norm.

$$\begin{array}{c}\sum\sqrt{(u-u_{gt})^2+(v-v_{gt})^2}\\ \sum|u-u_{gt}|+|v-v_{gt}|\end{array} \tag{4.52}$$

Although the endpoint error can provide useful information about the vector's length, it contains no measurement of the vector's orientation. To obtain this, the angular measurement is used.

4.6.3 Angular error

As we are talking about vectors, we need a complementary measurements over the magnitude error presented above.

$$\sum\arccos\left(\frac{u^T\cdot u_{gt}}{|u||u_{gt}|}\right) \tag{4.53}$$

This error measurement is frequently used, as it evaluates both the magnitude and the direction of the flow. Of course, it has its downsides. By taking the magnitude in equation, the higher speeds have a greater impact on the final result then the lower speeds having the same angular error.

Chapter 5

Proposed algorithm

In this chapter we will present our algorithm. We start from the cross correlation transform formulation of the optical flow from [2], and use the minimization technique described in chapter 4.5.2.

5.1 Energy Function Formulation

Our solution starts from the optical flow formulation in [2]. The flow equation is composed from a data term and a smoothness term.

The data term we choose a cross correlation transform to increase robustness and tolerance to occlusions.

We start from the normalized cross correlation.

$$\frac{1}{|\mathcal{N}|} \cdot \frac{\langle f - \mu_f, g - \mu_g \rangle}{\sigma_f \sigma_g}$$

The cross correlation measures the similarity between 2 signals. The results for the normalized cross correlation vary in $[0, 1]$, where 1 is a total match.

If we compute the cross correlation between the frame 1 warped with the flow and the second frame, the result should be 1. **As the normalized cross correlation is linear**, our data minimization function can be:

$$E_d = 1 - \frac{1}{|\mathcal{N}|} \cdot \sum_i \frac{\langle \tilde{I_2}(i + dw) - \mu_1, I1(i) - \mu_1 \rangle}{\sigma_1 \sigma_2}$$

where $\tilde{I2}$ is the image 2 warped with the flow and $I2$ is the first image. If we say that the cross transform is

$$C(i, k) = \frac{f_k - \mu}{\sigma} \in [0, 1]$$

where k is the vicinity of i. Than we can rewrite 5.1 as a sum of square differences.

$$\frac{1}{|\mathcal{N}|} \cdot \sum_i (\frac{\tilde{I}_2(i+dw) - \mu_2}{\sigma_2} - \frac{I_1(i) - \mu_1}{\sigma_1})^2$$

Considering that $\tilde{C}_2(i+dw) = \tilde{C}_2(i) + \nabla\tilde{(C)}_2(i) \cdot dw_i$, then $\tilde{C}_2(i) + \nabla\tilde{(C)}_2(i) \cdot dw_i + C_1(i) = 0$. Further more, if $\tilde{C}_2(i) - C_1(i) = C_t(i)$, the data function becomes

$$E_d = \sum_{i \in \omega} \sum_k (C_t(i,k) + \nabla C(i,k) \cdot (w_i - w_{0,i}))^2$$

Now, as the data term is in L^2, to increase the robustness of the formulation, the correction term will be in L^1. It will subject to the smoothness constraint as $\|w - w_i\|_1$. As we don't want the moistness constraint to affect all the pixels in the same way, we will formulate the weight of each pixel as a normal bilateral filter transform. The weight will be described by the similarity of the pixel with his neighbours. Unlike the classic bilateral filter who uses a Gaussian function, we will us a Laplace function:

$$e^{-\left(\frac{|\Delta_c(i,s)|}{\sqrt{2}\sigma_c} + \frac{|\Delta_d(i,s)|}{\sqrt{2}\sigma_d} \right)} \tag{5.1}$$

The energy function becomes:

$$E = \sum_{i \in \omega} \sum_k (C_t(i,k) + \nabla C(i,k) \cdot (w_i - w_{0,i}))^2 + \lambda \sum_k bf_{i,k} \cdot \|w_k - w_i\|_1 \tag{5.2}$$

5.2 Minimization procedure

To minimize this convex function, we will use the LMN method described in 4.5.2. We chose this optimization method because the generality of its formulation and its potential for improvement. It is more expensive than the proximal point approach, memory and computation wise, but it should have a faster convergence towards the result.

To save computational time and memory, we compute directly the matrices from 4.39.Finally the whole system will be determined by large sparse matrices. Let us take them one by on.

To compute G we need $H^T H$, where H the coefficient of the flow in the L^2 norm.

From equation 5.2, with the flow coefficient $\nabla C(i,k)$ we get:

$$H^T H = \begin{bmatrix} C_{x1}^2 & C_{x1}C_{y1} & 0 & 0 & \dots & 0 & 0 \\ C_{x1}C_{y1} & C_{y1}^2 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & C_{x2}^2 & C_{x2}C_{y2} & \dots & 0 & 0 \\ 0 & 0 & C_{x2}C_{y2} & C_{y2}^2 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & C_{x\Omega}^2 & C_{x\Omega}C_{y\Omega} \\ 0 & 0 & 0 & 0 & \dots & C_{x\Omega}C_{y\Omega} & C_{y\Omega}^2 \end{bmatrix} \tag{5.3}$$

From this G matrix will be computed:

$$G = \begin{bmatrix} 2H^T H & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.4}$$

Regarding dimension, if $\Omega = m \times n$, is the image dimension, the matrix $H^T H$ will be $2\Omega \times 2\Omega$ and G will be $6\Omega \times 6\Omega$. It is important that all the matrices are sparse, and bended.

Next, let us consider matrix $\boldsymbol{c}$; for this we need $H^T \boldsymbol{g}$. We now H, and ne notice the free term from 5.2 Note $(C_{t1} - C_{xn}u_{old} - C_{yn}v_{old})$ with e_n. Then, $\boldsymbol{g}$ is:

$$H^T \boldsymbol{g} = \begin{bmatrix} C_{x1}e_1 & 0 & \dots & 0 & 0 \\ 0 & C_{y1}e1 & \dots & 0 & 0 \\ \vdots & & & & \\ 0 & 0 & \dots & C_{x\Omega}e_{t\Omega} & 0 \\ 0 & 0 & \dots & 0 & C_{y\Omega}e_{t\Omega} \end{bmatrix}$$

Now, to compute A from the L^1 norm, we need αR. We identify α as the bf coefficient in 5.2 the α will be:

$$\alpha = \begin{bmatrix} bf_1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & bf_1 & 0 & 0 & \dots & 0 & 0 \\ 0 & 0 & bf_2 & 0 & \dots & 0 & 0 \\ 0 & 0 & 0 & bf_2 & \dots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & 0 & 0 & \dots & bf_\Omega & 0 \\ 0 & 0 & 0 & 0 & \dots & 0 & bf_\Omega \end{bmatrix} \tag{5.5}$$

and K

$$K = \begin{bmatrix} 9 & 0 & 1 & 0 & 0 & 0 & \dots & 0 & 0 \\ 0 & 9 & 0 & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & 0 & 9 & 0 & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & 0 & 9 & 0 & 1 & \dots & 0 & 0 \\ \vdots & & & & & & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \dots & 0 & 9 \end{bmatrix} \tag{5.6}$$

The K matrix has elements on the principal diagonal 9, on the diagonal shifted up and down it has groups of 6 ones and 2 zeros, on the diagonal shifted up and down with $2 * column_{number} - 2$ we have sequences of 2 zeros and 6 ones, on the diagonal shifted up and down with $2 * column_{number}$ we have full ones, and on $2 * column_{number} + 2$ shift we have groups of 6 ones and 2 zeros.

A matrix will be defined as in 4.39, with the size $2\Omega \times 6\Omega$

Now that we have all the big matrices are determined, we can find the vector $\boldsymbol{x}$ and the other unknowns from equations 4.49 and 4.50.

Algorithm 1 LMN approach

Set up pyramids I_1 I_2
If the coarsest level initialize f^0, s^0, λ^0
If not, unsample f^0, s^0, λ^0
for wr = 0, 1, 2 ... **do**
    Warp I2 towards I1
    Compute A
    for k = 0, 1, 2 ... **do**
        Compute G and $\boldsymbol{c}$ as they are dependent on $\boldsymbol{x}$
        Solve for Δx 4.49
        Recover f^0, s^0, λ^0 from 4.50
    end for
end for

Chapter 6

Detailed Design and Implementation

This offers a brief description of the tool used to develop our system, motivating why this solution was chosen. We present an overview over te system, and it's components.

Some of the key concepts are detailed from the implementation point of view. We present the concept of derivative discretization, the sparse matrix and their advantages are explained. Downsampling concept is explained in (more) detail, as we show the implementation details of this (algorithm?). Finally, the MATLAB output and color space is

6.1 Implementation Environment

MATrix LABoratory (MATLAB) is a high-level software tool and a programming language used in many disciplines, mostly for performing complex mathematical calculations, but also for signal and image processing, communications, control systems and computational finance.

It is mostly used to build prototype systems, the aim of which is to demonstrate certain concepts or working principles.

MATLAB's basic data element is the matrix, as the system was originally built for factoring matrices and solving linear equations. It offers a large number of built-in functions and various extensions, like Simulink, which is a graphical tool for modeling multidomain dynamic systems.

We have chosen MATLAB because of to the scientific nature of our project. The mathematical model and the proposed algorithms are implemented using this tool, as we focus on demonstrating their theoretical values and validate the system. However, because MATLAB uses mostly matrices for performing calculations, we had to adapt our implementation to match these requirements. In this sense, we often had to make modifications to the algorithms.

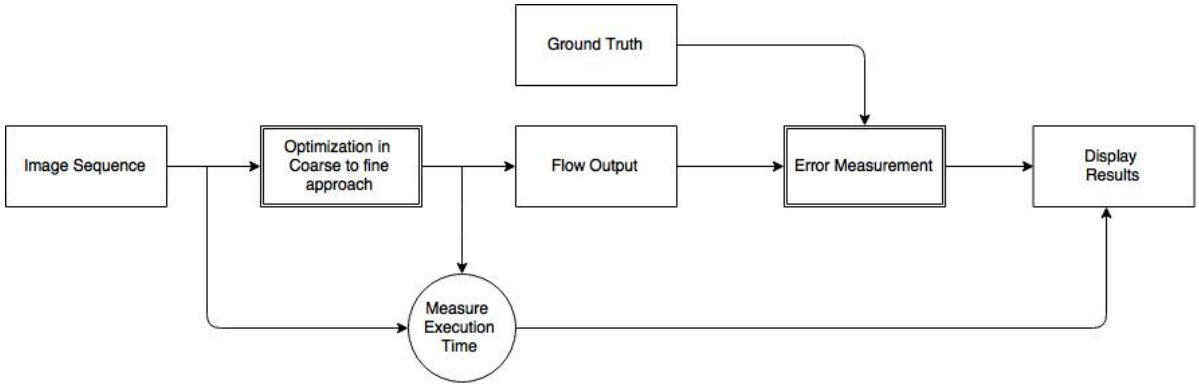


Figure 6.1: Conceptual architecture of the system.

6.2 General Overview

The system is represented in figure 6.2. The two most representative components of the system are the optimization procedure and the error measurement.

The optimization procedure contains the minimisation algorithm wrapped in coarse to fine approach. The implementation of this coarse to fine method will be discussed in detail in 6.4. This component takes as an input two frames and computes the optical flow between them and sends it to the error measurement component. A timer is started at the beginning of this process and stopped right after the output is computed.

The other component is represented by the error measurement part. This takes the output of the algorithm from the previous component and reads a .flo file with thee ground truth. The comparison between the two is made by computing the mean angular error and the endpoint error.

Finally, the results from the error measurement, the timer are displayed. Also a graphic representation of optical field, encoded in MATLAB's colour space, is displayed.

6.3 Proposed algorithm implementation

In this section we present the main functions of the system placed in a hierarchy. Also, we will detail each of these functions' behaviour and usage. In 6.3, we present the hierarchical representations of our system's functions. In the diagram we note the types Image a two dimensional array and flow a 3 dimensional array and flowStructure, the flow vector and his auxiliary variables, $\boldsymbol{\lambda}$, $\boldsymbol{s}$, $\boldsymbol{x}$, described in 5.2.

Main This is the function which starts the program. it can be run by itself or can be accessed from a GUI function. Its tasks are reading the images, and calling the 2 sub functions, *CoarseToFine* and *ComputeError*. After the flow and the error are computed, it displays the results and the encoded flow result.

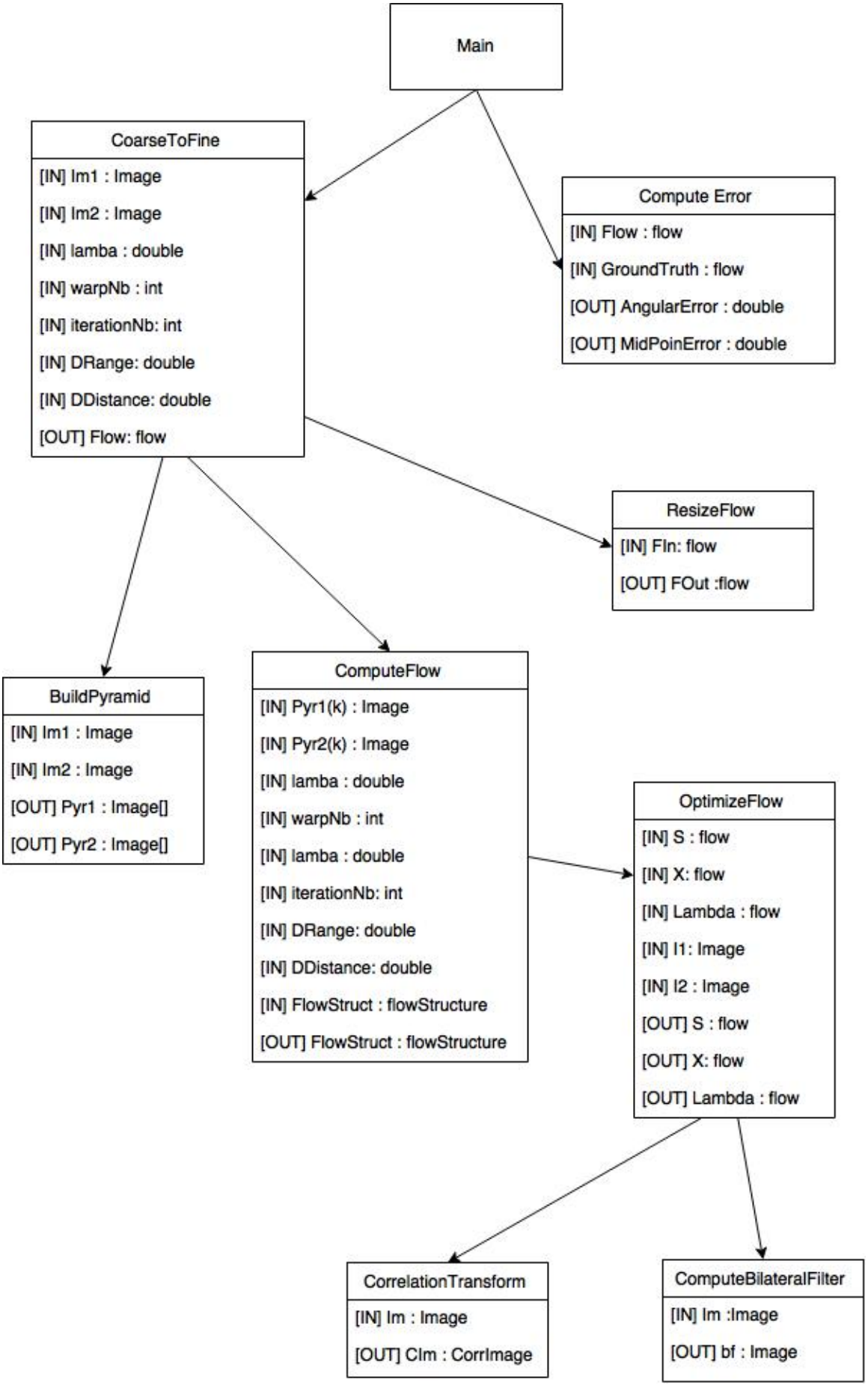


Figure 6.2: Hierarchical representation of the system functions.

CoarseToFine(Im1, Im2, params) This function holds the implementation for the coarse-to-fine method.

the method takes as parameters two images, *Im1* and *Im2* on which the flow will be computed, and a list of input parameters. The parameters are passed to the next function, so we will describe them on the function in which they are used.

The flow of this function is represented in figure 6.4. First the maximum height is computed with formula from 4.19. Then, the *BuildPyramid* function is called for each image. After the pyramids are built, in an iteration trough each level of the pyramid, starting from the coarsest, first the flow parameters are initialized if they are not, otherwise an up sampling is done with *resize*, then the *ComputeFlow* function is called for each level.

At the initialization level, a flow structure is build, from the flow and its auxiliary variables needed in the next step.

BuildPyramid(Im) This function has the task of building the pyramid. The input parameter is the image and the number of levels that it should have. The pyramid is build by repentantly downsizing the input image. It uses MATLAB's *resize* function for a with a sampling factor of 0.5. The function returns a cell structure of images with the higher level corresponding to the coarsest.

ResizeFlow(Flow, size) This function is used for up sampling. The input parameter is a flow vector and the desired size of the output flow. The sampling factor used for resizing the flow is the ratio between the new size and the old one.

Compute flow(Im1, Im2, wrs, maxIt, sigmaR, sigmaD) This function is called on every level of the pyramid. The *Im1* and *Im2* inputs are the images that correspond to the current level of the pyramid. *wrs* is the number of warps the system will performed, and *maxIt* is maximum number of iterations. *sigmaR* and *sigmaD* are used in computing the bilateral filter.

In this function are computed matrices of the system that do not change trough iterations. The structure of the function is: in an outer while loop, first the I2 is warped with the flow towards I1. ThenMatrices that do not change trough the iteration are computed : [see notations 5.2] G, c , K, $bf \cdot K$,$bf^2 \cdot K$, A. After the system in prepared, after the matrices are computed, the systm starts to iterate trough *OptimizationFlow* function.

OptimizeFlow(X, S, Lambda) In this function, the optical flow is optimized. After the flow and its parameters took an initial value in *CoarseToFine* function, they are updated using the formulas in section 4.5.2.

We first compute D^2 matrix from $D = S^{-1/2}X^{1/2}$, then as the matrix is single banded on the main diagonal we compute its inverse as $D^{-1} = S^{1/2}X^{-1/2}$, to save computational time. We need X and S matrices. This are the diagonal matrices from $\boldsymbol{x}$ and $\boldsymbol{s}$.

We use MATLAB's *spdiags* function to create this matrices. We also need X^{-1}. It will be computed in the same manner. Elements may appear *NAN*, from division with 0 in this matrices. This entries are replaced with 0.

Now we can compute the left hand variables from the equations, namely

$$\begin{aligned} \boldsymbol{r}_c &= G\boldsymbol{x} + \boldsymbol{c} - A^T\boldsymbol{\lambda} - \boldsymbol{s} \\ \boldsymbol{r}_b &= A\boldsymbol{x} - \boldsymbol{b} \\ \boldsymbol{r}_a &= XS\mathbf{1} - \sigma\mu\mathbf{1} \\ \hat{\boldsymbol{r}}_b &= -\boldsymbol{r_b} + D_2^2\hat{\boldsymbol{r}}_{c2} - D_3^2\hat{\boldsymbol{r}}_{c3} \\ \hat{\boldsymbol{r}}_c &= \boldsymbol{r}_c + X^{-1}\boldsymbol{r}_a \\ \tilde{\boldsymbol{r}}_{c1} &= \hat{\boldsymbol{r}}_{c1} + \alpha R^T(D_2^2 + D_3^2)^{-1}\hat{\boldsymbol{r}}_b \end{aligned} \tag{6.1}$$

We access D1, D2 and D3 by splitting D in 3 equal parts on the diagonal.

We can now compute solve the system for $\boldsymbol{x}$.

We use the formula

$$\tilde{\boldsymbol{r}}_{c1} = \hat{\boldsymbol{r}}_{c1} + \alpha R^T(D_2^2 + D_3^2)^{-1}\hat{\boldsymbol{r}}_b$$

. and the auxiliary variables.

$$\begin{aligned} \Delta\boldsymbol{\lambda} &= (D_2^2 + D_3^2)^{-1}(-\hat{\boldsymbol{r}}_b - \alpha R\boldsymbol{f}) \\ \Delta\boldsymbol{v}^+ &= D_2^2(-\hat{\boldsymbol{r}}_{c2} - \boldsymbol{\lambda}) \\ \Delta\boldsymbol{v}^- &= D_3^2(-\hat{\boldsymbol{r}}_{c3} + \boldsymbol{\lambda}) \\ \Delta\boldsymbol{s} &= G\boldsymbol{x} - A^T\boldsymbol{\lambda} + \boldsymbol{r}_c \end{aligned} \tag{6.2}$$

Thee function returns $\boldsymbol{x}$, $\boldsymbol{s}$,$\boldsymbol{\lambda}$, which will be the input at the next iteration. The code is available in Appendix A.

CorrelationTransforn(Im) The correlation transform function, used in expressing the data term transforms an image vector into its cross correlate form. The formula in computing the correlation transform is described at 4.30.

6.4 Pyramidal Implementation

The pyramidal approach wraps every implementation we did.

For the pyramid constriction, the image needed to be repeatedly unsampled. For this we used MATLAB's resizing operation, *imresize*. This function takes as input the image and the desired output size, and returns the matrix resized.

The method of the resizing is an optional parameter. By default is bicubic, where the interpolation is done a weighted average of the 4 by 4 vicinity. Another option is

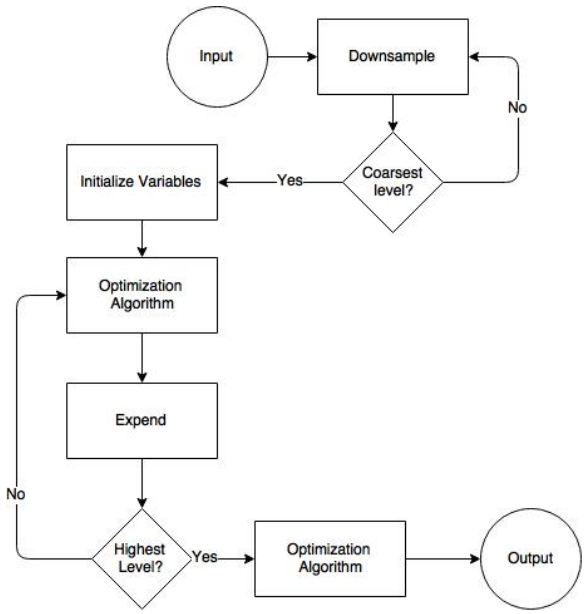


Figure 6.3: Pyramidal Flowchart

bilinear, in which the 2 by 2 vicinity is considered. Because out sampling factor is 2, we have chosen the bilinear interpolation. Also bilinear interpolation yield slightly better results in our optical flow computation.

We consider the lowest level, when the minimum between the height and the width is at least 20 pixels.

$$\min\left(\frac{\log(20/N)}{\log(0.5)}, \frac{\log(20/M)}{\log(0.5)}\right)$$

At the coarsest level, the algorithm starts optimizing the initial guess. After at most 5 iterations, it goes to the next level, rescaling the flow. The same *imresize* function is used.

Al this steps repat until the algorithm reaches the finest level, where the last optimization is done.

6.5 Derivative Discretization

For obtaining the derivatives of the image and also the gradient we used the discretized formula:

$$\frac{\partial f(x,y)}{\partial x} = \frac{f(x-h,y) - f(x+h,y)}{2h} \tag{6.3}$$

By taking the small step $h = 1$ we can compute the derivatives and the gradient of the image. For achieving this, the a Prewitt kernel convolution is used. The convolution process implies multiplying each pixel of a image and its neighbours by an kernel, cho-

sen depending on the operation. Perwitt is one of the kernels used for derivatives along Laplacian, Sobel and Roberts. The Perwitt filter is $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$ for the horizontal axis, vertical respectively. It is relatively easy and inexpensive to compute. Generally it is used in the context of edge detection, but in our implementations, was used in the energy function, especially in the brightness constraint.

When computing only the gradient, and the partial derivatives are not further used, the kernel applied is the Laplacian, also found in the context of edge detection. The general formula is:

$$f(x,y) = \frac{f(x-h,y) + f(x+h,y) + f(x,y-h) + f(x+h,y) - 4f(x,y)}{(2h)^2} \tag{6.4}$$

In 4.3.1 when computing the derivative, the gradient ∇u was replaces with$\bar{u} - u$). This is because the gradient was computed with a Laplace Kernel:

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & -9 & 1 \\ 1 & 1 & 1 \end{bmatrix} \tag{6.5}$$

Notice the property of the differentiative krnels, both Perwitt and Laplce, the sum of the elements must alwais be 0.

6.6 Sparse Matrix

Sparse matrices are usually large matrices with most of the elements 0. They are popular with solving large linear systems.

The memory advantage of sparse over full matrices is that in sparse only the non zero values and their indexes are stored.

An example was run in MATLAB. We created a 5000×5000 matrix, one full *M_full* and one sparse *M_sparse*. In both of them we inserted 50 elements non null. When we check the properties of the matrices, we observe the difference in the memory. Even if the matrices are identical, by keeping a matrix sparse, only the non zero elements are kept in the memory.

Name	Size	Bytes	Class	Attributes
M_full	5000×5000	200000000	double	
M_sparse	5000×5000	41608	double	sparse

Computational wise it skips redundant operations with 0 such as addition and multiplication.

Let us take another example. In our flow algorithms we have to solve equations, so considering $A\boldsymbol{x} = b$, A takes first full then sparse, with the size 5000×5000 and with 60 elements that are not 0. b will be a column vector with 5000 random elements. For our experiment we use the MATLAB function *mldivide* to solve the system for x. The time for each operation is shown in the table below.

Name	Size	Elapsed time	Attributes
M_full	5000×5000	8.389874	
M_sparse	5000×5000	0.103146	sparse

A special type of sparse are band matrices. This are regular sparse, the non-zero elements are grouped diagonally. This allows even more efficient memory use and a lower computation time. Our system works with such matrices.

First, in solving the system we used the *mldivide* function, but the program run very slow and on a 4GB of RAM it did not support other programs to run in parallel. The maximum default memory given by MATLAB for stocking matrices in 1 GB, and *mldivide* took most of it. We've switched to the *gmres* which is specialized in banded matrices and the memory consumption decreased to almost a half and the execution time shorten to about 10

To proov this in a simple way we have chosen to test on a 500000×500000 sparse matrix with three five-columns bands. It is important that the matrix is symmetric. We've tested and timed both *mldivide* and *gmres*. The results are shown in the table below.

Function	Size	Elapsed time
mldivide	500000×500000	28.587425
gmres	500000×500000	2.625163

Sparse matrix don't perform this well only in solving systems, but also in simpler algebraic operations. The function are optimised to improve metrics of such implementation and ar worth using as much as it may be.

6.7 Output and MATLAB Colorspace

For the flow output and visualization, the functions available on Middlebury ware used. As their ground truth flow was encoded in .flo format, we kept the same convention. The .flo is the unwrapped flow matrix in a long stream. values are red one by one and then the flow is rearranged in a $m \times n \times 2$ matrix, where m and n are the flow field's height and width and 2, every vector's x and y components.

Middlebury[22] provides functions for writing and reading this files.

For visualizing the output each vector is encoded to the MATLAB's colour space. As shown in 6.7, the vectors on the write, expressed as arrows, are encoded trough colour code. The colour notation is used more that the vectors, because the changes in direction are more obvious. Also, the magnitude of the vector is encoded by the level of saturation of the colour, white representing no movement, and a full, saturated colour represents the widest movement of the flow.

The pixel range between white and full colours vary, depending on the flow output. The function adapts this range to the needs. As can be observed, the encodings for the main directions are: indigo for up, red for right, yellow for down and cyan for left.

On Middlebury's site, thy provide a function for the colour transform in the same package as the read and write functions mentioned earlier. They can be found on the datasets section as "flow-code".

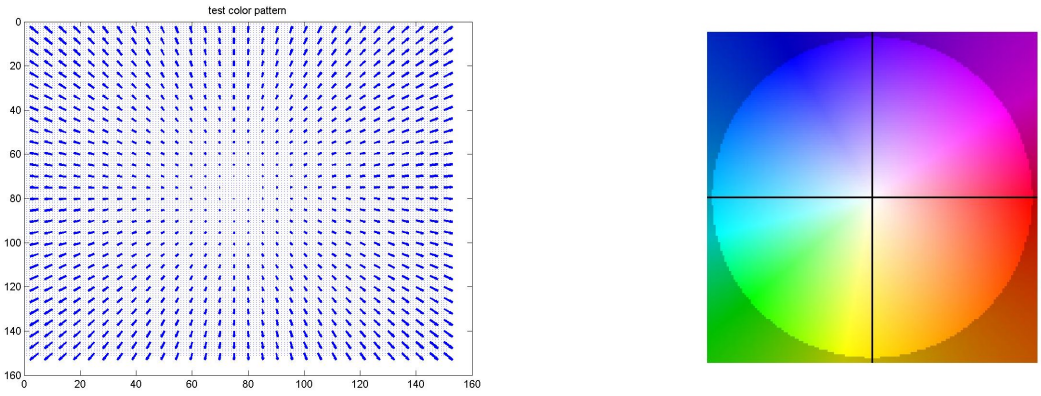


Figure 6.4: Colour encoding.

Chapter 7

Testing and Validation

About 5% of the paper

7.1 Title

7.2 Other title

Chapter 8

User's manual

In this chapter we specify the resources needed to run our application and the setup steps towards lunching the application. We also describe the input parameters.

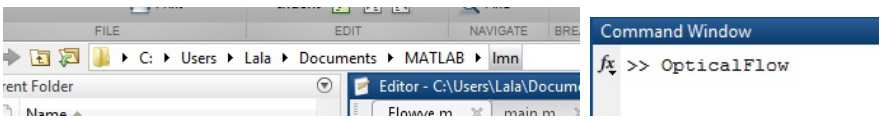
8.1 Required resources

For setting up the application, any operating system capable of running MATLAB can be used. In this presentation, we used Windows 7 OS. For a faster computation a 8GB of RAM and high speed processor are recommended.

The implementation and testing were made on a Intel i3 processor and a 4 GB of RAM, but the execution times were relatively high.

8.2 Application Setup

1. Copy the contents of the CD to `C:\` or other local disk and extract the archive.
2. Start MATLAB.
3. Set MATLAB working directory to `C:\...\lmn\`. See figure a
4. In the MATLAB command window, type the following command: *OpticalFlow*. See figure b



(a) (b)

Figure 8.1: Path Selection

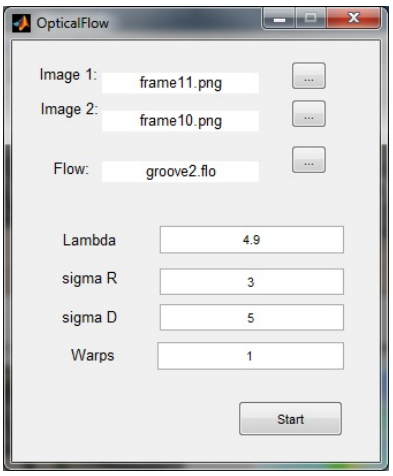


Figure 8.2: Path Selection

8.3 Running the application

After the *OpticalFlow* command was initiated, another window will pop up with th settings.

First the user should provide 2 images in the *Image 1* and *Image 2* fields.

The *flow* input allows user to specify the ground truth to which the flow will be computed. Expects a .flo fie. this field is optional.

Then the parameters are requested. All of them affect th smoothness of the flow.

Lambda is indirect proportionally with the flow smoothness.

Chapter 9

Conclusions

In this thesis we discussed the main aspects of optical flow computation. We have studied and implemented the classical algorithms from this field. Also we have studied ways of improvement for the algorithm. We've implemented a Least Mixed Norm solution, which is a new approach in solving the optical flow.

9.1 Research

The scope of this thesis is accommodation with the field of optical flow. With a large number of approaches and a highly complex mathematical model, we have started with the classical models and their implementation.

We have seen the problem has two parts, the formulation of the optical flow system and the optimization method.

The formulation may be local or global. Although some later works included a local approach, the global formulation achieves more attention in dense computation. This is due the good results and potential for development.

The norm of the formulation ca be in L^2 or L^1. L^2 is present in the first approaches of the optical flow, but the last solutions incline to a L^1 formulation, as the robustness it gives to the algorithm counterbalances the extra computational cost.

9.2 Proposed Solution

After we have studied some L^1 techniques for optical flow computation we have implemented a new approach in this domain, the Least Mixed Norm optimization.

The L^1 technique is preferred over $L2$. This is due to the robustness given to the algorithm, and the non-isotropy property. Although is more computational expensive, it can be implemented on modern computers with good run time.

We've used as a formulation model the cross correlation formulation. We've compared our results with the Projection of Proximal Point method, which was recognized for

great results.

AT our first implementation of the two algorithms, Projected Proximal Point and Least Mixed Norm, we have observed how fast the LMN techniques converges compared to the PPP.

At th first implementation our results from the LMN optimization are comparable with those from PPP implementation.

9.3 Further development

The LMN implementation will be improved in the near future, as it gives satisfying results even with very little optimization.

The bottleneck of our implantation is the sparse system solver, because is the most costly part with respect to execution time and memory used. Because the system matrices are very well structured as banded symmetric sparse, a more specialized solver is needed.

Because the big applicability of the sparse problems there should be implemented specialized sparse solvers for this kind of matrices, and especially in c, where the this kind of problems are highly parallelizable.

A pre and post processing phase should fallow, to improve the accuracy of the results.

After the algorithm is properly set and no optimization can be done in the near future, it can be implemented for parallelizable platform like CUDA, using specific libraries like openGL. This should bring a considerate increase in execution time.

Bibliography

[1] J. J. Gibson, "The perception of the visual world." 1950.

[2] M. Drulea and S. Nedevschi, "Motion estimation using the correlation transform," *Image Processing, IEEE Transactions on*, vol. 22, no. 8, pp. 3260–3270, 2013.

[3] B. K. Horn and B. Schunck, "Determining optical flow," in *1981 Technical Symposium East*. International Society for Optics and Photonics, 1981, pp. 319–331.

[4] D. Sun, S. Roth, and M. J. Black, "Secrets of optical flow estimation and their principles," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2432–2439.

[5] D. Sun, S. Roth, and M. Black, "A quantitative analysis of current practices in optical flow estimation and the principles behind them," *International Journal of Computer Vision*, vol. 106, no. 2, pp. 115–137, 2014.

[6] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision." in *IJCAI*, vol. 81, 1981, pp. 674–679.

[7] A. Mitiche and J. K. Aggarwal, *Computer Vision Analysis of Image Motion by Variational Methods*. Springer, 2014.

[8] W. Trobin, "Local, semi-global, and global optimization for motion estimation," 2009.

[9] A. Wedel and D. Cremers, *Stereo scene flow for 3D motion analysis*. Springer Science & Business Media, 2011.

[10] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, "Performance of optical flow techniques," *International journal of computer vision*, vol. 12, no. 1, pp. 43–77, 1994.

[11] D. Fleet and Y. Weiss, "Optical flow estimation," in *Handbook of Mathematical Models in Computer Vision*. Springer, 2006, pp. 237–257.

[12] J. R. Bergen, P. Anandan, K. J. Hanna, and R. Hingorani, "Hierarchical model-based motion estimation," in *Computer VisionâĂŤECCV'92*. Springer, 1992, pp. 237–252.

[13] A. Bab-Hadiashar and D. Suter, "Robust optic flow computation," *International Journal of Computer Vision*, vol. 29, no. 1, pp. 59–77, 1998.

[14] A. Wedel, T. Pock, J. Braun, U. Franke, and D. Cremers, "Duality tv-l1 flow with fundamental matrix prior," in *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*. IEEE, 2008, pp. 1–6.

[15] A. Wedel, T. Pock, C. Zach, H. Bischof, and D. Cremers, "An improved algorithm for tv-l 1 optical flow," in *Statistical and Geometrical Approaches to Visual Motion Analysis*. Springer, 2009, pp. 23–45.

[16] J. Molnár, D. Chetverikov, and S. Fazekas, "Illumination-robust variational optical flow using cross-correlation," *Computer Vision and Image Understanding*, vol. 114, no. 10, pp. 1104–1114, 2010.

[17] F. Stein, "Efficient computation of optical flow using the census transform," in *Pattern Recognition*. Springer, 2004, pp. 79–86.

[18] H.-H. Nagel *et al.*, "Constraints for the estimation of displacement vector fields from image sequences." in *IJCAI*. Citeseer, 1983, pp. 945–951.

[19] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Computer Vision, 1998. Sixth International Conference on*. IEEE, 1998, pp. 839–846.

[20] N. Parikh and S. Boyd, "Proximal algorithms," *Foundations and Trends in optimization*, vol. 1, no. 3, pp. 123–231, 2013.

[21] H. Fu, M. K. Ng, M. Nikolova, and J. L. Barlow, "Efficient minimization methods of mixed l2-l1 and l1-l1 norms for image restoration," *SIAM Journal on Scientific computing*, vol. 27, no. 6, pp. 1881–1902, 2006.

[22] [Online]. Available: http://flow.middlebury.com

[23] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International Journal of Computer Vision*, vol. 92, no. 1, pp. 1–31, 2011.

Appendix A

Relevant code

```
Optimize (x,s,lambda)

D23SqInv = spdiags(1./ ...
( ...
x(1+dim:2*dim)./s(1+dim:2*dim)+ ...
x(1+2*dim:3*dim)./s(1+2*dim:3*dim)  ...
) ...
, 0, dim, dim);
D23SqInv(isnan(D23SqInv)) = 0;

D3 = spdiags(x(1:dim)./s(1:dim), 0, dim, dim);
D2 = spdiags(x(1:dim)./s(1:dim), 0, dim, dim);
D1Inv  = spdiags(s(1:dim)./x(1:dim), 0, dim, dim);
XInv = spdiags(1./x(:),0,3*dim, 3*dim);

D3(isnan(D3)) = 0;
D2(isnan(D2)) = 0;
D1Inv(isnan(D1Inv)) = 0;
XInv(isnan(XInv)) = 0;

miu = mean(x(:).*s(:));
sigma = std2(x(:).*s(:));
ra = x(:).* s(:) - miu*sigma*ones(3*dim, 1);
rc =  G*x+c-s-A'*l ;
rb = A*x;
rch =rc + XInv* ra;

rch1 = rch(1:dim);
rch2 = rch(1+dim:2*dim);
```

```
rch3 = rch(1+2*dim:3*dim);

rbh = rb + D2 * rch2 - D3 * rch3;
rcTilda =rch1 +bfK'*(D23SqInv)*rbh;

BigA = 2*G(1:dim, 1:dim)+D1Inv+bf2K'*D23SqInv*K;

deltaF = gmres(BigA, -rcTilda);


deltaL = D23SqInv*(-rbh-bfK*deltaF);
deltaVp = D2*(-rch2-deltaL);
deltaVm = D3*(-rch3-deltaL);
deltaX = [deltaF; deltaVp; deltaVm];
deltaS = G * deltaX- A'*deltaL+ rc;

x = x + deltaX;
s = s + deltaS;
l = l + deltaL;
```

Appendix B

Other relevant information (demonstrations, etc.)

B.1 Horn Scunk equation to Jacobi Iteration

First, let us define the staring equation.

$$E = (I_x u + I_y v + I_t)^2 + \lambda(\|(\nabla u)\|_2^2 + \|(\nabla v)\|_2^2) \tag{B.1}$$

by discretizing $\|(\nabla u)\|_2^2$ with a Laplace filter, it can be rewritten as $(\bar{u} - u)^2$. The equation then becomes:

$$E = (I_x u + I_y v + I_t)^2 + \lambda((\bar{u} - u)^2 + (\bar{v} - v)^2) \tag{B.2}$$

The we apply the partial derivatives and eqaual then to 0 in order to find the minimum, as E being convex,

$$\begin{aligned} \frac{\partial E}{\partial u} &= I_x(I_x u + I_y v + Iy) + \lambda(\bar{u} - u) \\ \frac{\partial E}{\partial v} &= I_y(I_x u + I_y v + Iy) + \lambda(\bar{v} - v) \end{aligned} \tag{B.3}$$

Now to rearrange the terms

$$\begin{aligned} (I_x^2 + \lambda)u + I_y I_x v &= -I_x I_t + \lambda\bar{u} \\ I_y I_x u + (I_y^2 + \lambda)v &= -I_y I_t + \lambda\bar{v} \end{aligned} \tag{B.4}$$

and in matrix form , $A\boldsymbol{x} = b$:

$$\begin{bmatrix} I_x^2 + \lambda & I_y I_x \\ I_y I_x & (I_y^2 + \lambda) \end{bmatrix} \cdot \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_x I_t + \lambda \bar{u} \\ -I_y I_t + \lambda \bar{v} \end{bmatrix} \tag{B.5}$$

The equation can be solved as $\boldsymbol{x} = A^{-1}b$. First to compute A's determinant

$$\frac{1}{\lambda(\lambda + I_x^2 + I_y^2)} \tag{B.6}$$

the equation becomes:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\lambda(\lambda + I_x^2 + I_y^2)} \begin{bmatrix} I_y^2 + \lambda & -I_y I_x \\ -I_y I_x & (I_x^2 + \lambda) \end{bmatrix} \cdot \begin{bmatrix} -I_x I_t + \lambda \bar{u} \\ -I_y I_t + \lambda \bar{v} \end{bmatrix} \tag{B.7}$$

from this, on can easily find u and v as:

$$\begin{aligned} u^{k+1} &= \frac{(I_y^2 + \lambda)\bar{u} - I_x I_y \bar{v} - I_x I_t}{I_x^2 + I_y^2 + \lambda} \\ u^{k+1} &= \frac{-I_x I_y \bar{u} + (I_y^2 + \lambda)\bar{v} - I_y I_t}{I_x^2 + I_y^2 + \lambda} \end{aligned} \tag{B.8}$$

Appendix C

Published papers