FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Implementation of Optical Flow Algorithms in FPGA Platforms with embedded CPU

João Pedro Ramos de Oliveira Santos

Mestrado Integrado em Engenharia Electrotécnica e de Computadores

Supervisor: José Carlos dos Santos Alves (Professor Doutor)

June 2009

© João Pedro Ramos de Oliveira Santos, 2009

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

MIEEC - MESTRADO INTEGRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES 2008/2009

A Dissertação intitulada

"IMPLEMENTAÇÃO DE ALGORITMOS DE FLUXO ÓPTICO EM PLATAFORMAS FPGA"

foi aprovada em provas realizadas em 20/Julho/2009

o júri

Presidente Professor Doutor José Manuel Martins Ferreira
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

Professor Doutor Arnaldo Silva Rodrigues de Oliveira
Professor Auxiliar Convidado do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

Professor Doutor José Carlos dos Santos Alves
Professor Associado do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

O autor declara que a presente dissertação (ou relatório de projecto) é da sua exclusiva autoria e foi escrita sem qualquer apoio externo não explicitamente autorizado. Os resultados, ideias, parágrafos, ou outros extractos tomados de ou inspirados em trabalhos de outros autores, e demais referências bibliográficas usadas, são correctamente citados.

Autor - **JOÃO PEDRO RAMOS OLIVEIRA SANTOS**

Faculdade de Engenharia da Universidade do Porto

Abstract

This document aims at presenting the work developed in the field of visual motion characterisation. It describes the various existing techniques for the calculation of the Optical Flow parameter in a set of frames. Project decisions are also addressed and detailed information about the implemented hardware core architecture is provided. The implementation results are analysed to support the final conclusions.

The subject of visual perception is considerably vast and its characterization is undisputedly complex. It stands for the process of information gathering about the surrounding world, namely object identification, movement calculation, distance and time to contact estimation. The amount of data that is involved in this process is enormous and so the computational effort required to extract measurable data, in a real-time environment. Researchers have broken the problem into separate areas, one of them being the calculation of the Optical Flow in a scene, which is the main subject of this dissertation.

Artificial systems employ mathematical techniques to analyse sequences of frames and extract information about the movement that has occurred in a scene. The inconvenient of such techniques lies on the fact that the computational effort required to analyse a full image is too demanding to be implemented making use of conventional computer architectures. Therefore, these systems either try to reduce their vision field (reducing frame size) or risk not being able to cope with real-time requirements. However, dedicated hardware solutions have also been proposed in the meanwhile with alternative and faster architectures.

This dissertation explores the later developments of reconfigurable hardware platforms and altogether with the study of less-efficient implementations, presents an efficient architecture of an Optical Flow estimator and a C implemented framework to communicate with the designed IP core.

Acknowledgments

I would deeply like to thank my family for the support and care they have always shown and for gearing me up with the best tool one can truly crave for, an exemplary upbringing.

To my buddies with whom I shared the same laboratory and labour hours, Carlos Resende, João Pereira, João Rodrigues, Pedro Santos, Nuno Pinto, I thank them for their help and companionship and, no less often, patience.

The Author

Pinto a realidade por números,
Operações subjectivas,
Tudo no exponencial de uma folha.

Observo os movimentos rectos
Na irregularidade do caos.
Os padrões florescem nesta tela
Onde a traço grosso
Surge a fina percepção da vida.

Meço então as distâncias relativas,
Conto todos os pormenores,
Pois a cada esforçada aresta
Retoco o integral de uma existência.

The Author, 2005/02/13

Contents

List of Figures

List of Tables

Abbreviations and Symbols

DEEC	Departamento de Engenharia Electrotécnica e de Computadores
FEUP	Faculdade de Engenharia da Universidade do Porto
MIEEC	Mestrado Integrado em Engenharia Electrotécnica e de Computadores
IP	Intelectual Property
OF	Optical Flow
UAV	Unmanned Autonomous Vehicle
FOE	Focus of Expansion
OFCE	Optical Flow Constraint Equation
FPGA	Field Programmable Gate Array
SAD	Sum-of-all-differences
FSM	Finite State Machine
LUT	Look Up Table
FIFO	First In First Out

Chapter 1

Introduction

1.1 Extent

Eyesight is one the five essential senses the human being is equipped with. It is crucial for interacting with the surrounding world and perceiving it. Many simple actions such as, grabbing an object or moving, for instance, rely on this basic and yet very complex information channel, vision. However, the processes of retrieving information about the form of an object, position, colour or movement, conceal some of the most challenging image processing tasks.

The study of the biologic vision system over the last century has resulted in numerous attempts to mimic its natural analysis and processing characteristics, within different ranges of complexity. This process has brought general knowledge to a closer point of how natural systems work, however, nature remains unmatched even though technology is known to surpass the intrinsic potential of organic systems.

The robotic applications, in which this type of system is usually applied on, demand great ability to handle large amount of data as well as relative accuracy and speed. Hence, the set of requirements that such algorithms have to meet is very demanding. Therefore, they often require to be run on powerful machines, fact that represents a conditioning to them. This fact has turned the attention of research to the emerging necessity of developing new architectures and algorithms. In the attempt to contribute to the field of Optical Flow calculation, this thesis suggests a new hardware architecture able to meet real-time requirements, providing also that it can be used on a development environment such as Linux.

1.2 Motivation and Objectives

The human perceptive system is capable of identifying objects, distances, movement, velocities, time to collision, etc.. This is achieved through the analysis of images that come from the eyes. In this task, the visual system plays an important role, providing a constant sequence of frames which are computed by our complex neural network and translated into the referred perceptions.

In what concerns artificial systems, the existing techniques to model perception follow simple and yet challenging baseline concepts. These techniques are implemented in software programming languages (e.g. C, C++) running on powerful and dedicated machines. Each type of algorithm has a different theoretical background which results in different characteristics in terms of speed and accuracy.

However, the restriction to use software dedicated algorithms does not meet the necessary time constraints that real-time applications imply. This is due to the fact that computer architectures are designed to run instructions in sequential order, thus fitting general application requirements. Consequently, the calculation of complex data, such as Optical Flow, using these architectures, implies great loss of efficiency and waste of resources, resulting in failing to meet real-time requirements.

For relatively not many years, some hardware based estimator's architectures were introduced, successfully reducing the average processing-time and allowing a significant increase of processing rate. For real-time applications such as, security and multimedia systems or for autonomous vehicles, this research proved to be extremely useful.

The present state of the art in what concerns hardware implementations of Optical Flow estimators indicates that there is room for some further improvements. This is due mainly to the fact that most reference papers are based on technology which is, to the present time of this work, outdated. Besides this, the improvement of FPGA (Field Programmable Gate Array) platforms to greater reconfigurable areas and speeds, opens up the possibility to the implementation of alternative architectures. Such advancements allow the processing of larger frames in real-time applications and better accuracy. Presently, some of these platforms are also able to accommodate a 32-bit MicroBlaze CPU in their reconfigurable area. It is also possible to set specific distributions of Linux to run on these embedded CPUs. As a result, it is possible to create an interesting mix up of dedicated IPcores running in conjunction with Linux in the CPU, which is useful for robotic applications development, for instance.

The scope of this work is to use the previous knowledge acquired from successful implementations and develop a fully functional computational infrastructure. The final system should take the most advantage of the FPGA platform and its resources in order to be possible to develop software applications in a Linux environment. The elementary system functions should be grouped under a user-friendly application or framework.

The main goals set for this work are as following:

1. Study the concept of movement perception.
2. Research and evaluate Optical Flow (OF) techniques and algorithms.
3. Test relevant OF estimator implementations.
4. Develop and validate a HDL algorithm implementation, based on an existing Optical Flow algorithm.
5. Integrate with an existing digital image processing chain.

6. Devise a framework compatible with the FPGA embedded CPU.

7. Validate and examine the attained results.

1.3 Contributions

The main contributions of this thesis are described next:

1. Present a brief but systematic study of the OF algorithms and estimators.

2. Elaborate a real-time architecture for a 640x480 frame OF estimator.

3. Integrate the OF estimator core with a MicroBlaze CPU, running Linux.

4. Develop a stable and reliable framework to allow user-friendly interaction between the OF estimator core and software applications.

5. Produce a working example of a real-time video processing application, using the items referred before.

6. Contribute with a stable theoretical and implementation platform fit for further developments in the fields of perception modelling, algorithm improvement and software-hardware partioning.

1.4 Document Layout

This document is structured in five chapters which refer the following subjects: movement perception, OF estimators, software and hardware implementations of OF estimators, model and the associated framework specification, integration with a real-time demanding application, results' analysis, and lastly, conclusions and future work. This document is also supported by a webpage at [9].

In chapter 2, a summary of the theoretical background is presented. The main perception theories are referred in the attempt to contextualise the interest of artificial systems on Optical Flow calculation.

Chapter 3 describes the present state-of-art about the OF estimation techniques. It addresses the baseline of the different types of estimators and provides an insight about each ones relative advantages. Hardware implementations of the estimators are also analysed.

In chapter 4, the selected architecture of the OF estimator is laid in detail. The datapath of the implemented OF core is described through cross-analysis of the reference algorithm and the real-time performance requirements. The software application is described afterwards.

Chapter 5 takes on the discussion of the test results and subsequent examination of the performance of the implemented OF estimator. Additionally, some relevant project decisions and their outcome are reviewed.

Finally, in chapter 6 the conclusions of this document are presented altogether with some perspectives of future developments and additions to the present work.

Chapter 2

Visual Perception

2.1 Introduction

The study of perception is of the interest of a wide set of scientific areas which include psychology, cognitive science, neuroscience and others related with technology and artificial systems.

Visual perception, in general, studies the processes of light detection and scene interpretation. Its study has been split up into several branches of investigation which are supported in known reference publications [10, 11, 12]. Consequently, it was concluded that the study of motion perception was of vital importance. Motion perception focuses on studying the acquisition of the speed and direction of the composing elements of a scene. Optical Flow (OF) is a parameter that quantifies movement speed and direction and can be used to achieve motion perception.

Robotic systems and multimedia applications are some examples which employ visual or motion perception techniques. In the first case, they are employed as aid systems for navigation and movement characterization, and scene structure extraction or prediction in the latter.

This chapter attempts to describe the theoretical background that led research to acknowledge the real importance of Optical Flow maps in the characterization and study of motion and visual perception.

2.2 From Visual to Motion perception

The concept of visual perception cannot be understood from the single study of human sight. As referred before, its achievement is the result of an undeterministic process such as our brain activity. What it actually viewed is highly affected by emotional states, context bias, psychological affinities, memories, etc.. These phenomena are difficult to measure and model in objective terms.

Besides this, it must be also taken into account that most of the perceptions which are generally associated to sight are, in fact, the result of a reconstructive process. As a matter of fact, eyes are only capable of capturing 2D images, as a brightness grid. The perception of depth, distance, movement and object recognition are all the outcome of the neural activity and brain processing

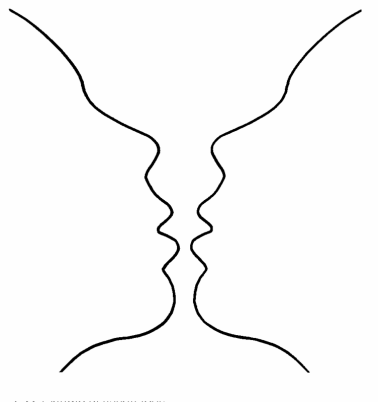


Figure 2.1: The vase-face perception is fully explained by the six Gestalt Principles.

capabilities, which draw over a 2D sketch, the calculated data. It is worth mentioning that the human brain is capable of remarkably doing all this in a real-time environment.

During the history of the study of Visual Perception there have been presented some revealing facts about how our sight, together with our brain, work to interpret reality. There are some reference theories on that matter which introduce the background concepts of perception. Though they are not essential to the comprehension of this work, they explain why modern artificial systems work with certain types of parameters in order to quantify visual perception.

In the next subsections the subjects will be addressed in a way that attempts to describe how the concept of visual perception evolved.

2.3 Gestalt Theory

The Gestalt theory is the theory of Perceptual Organization [13, 14]. It takes its name from the German philosophical term for shape and is best known for its theoretical approach and highly empirical principles. It supports that to human mind when parts are individually identified out of their context they do not acquire the same meaning as the whole. Taking a city landscape for instance, if one thinks of it in its elements alone such as a traffic sign or a passing car or a pedestrian, one realises that they have different meaning than when they observed altogether in the same picture. When the whole is observed, parts lose some of their significance to it and this happens in an unconscious way. A well known example is the illusion presented in figure 2.1, where the whole shows a vase while parts resemble two faces.

Furthermore, the theory also states that any of these phenomena can be fully explained by the six Gestalt Principles which are described next:

1. Proximity - things which are found closer together are pictured and *felt* as belonging together.

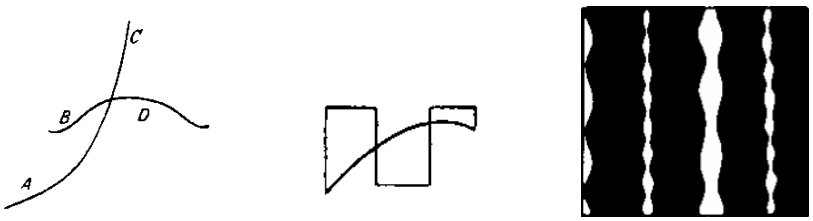
2. Similarity - things which share visual characteristics such as shape, size, colour, texture, value or orientation will be *viewed* as belonging together.

3. Common Fate - things that move together are seen as being together.

4. Good Continuation - there is a natural preference for continuous figures - there are four segments of line in figure 2.2(b) although only two have a tendency to be *viewed*.

5. Closure - as the last principle, there is also a natural tendency to view figures as closed shapes, independent of continuity or similarity, as in figure 2.2(a).

6. Area and Symmetry - the smaller of two overlapping figures is generally perceived as figure whilst the largest is perceived to be its background. The same happens with symmetrical contours where these tend to isolate closed shapes from them remnants of the picture, as in figure 2.2(c).



(a) Good Continuation. (b) Closure. (c) Area and Symmetry.

Figure 2.2: The images show examples in which the Gestalt Principles fully explain the visual illusions.

The Gestalt principles, some more evidently than others, explain the ability to perceive objects in OF maps. On specific occasions, objects can be separated from the background flow since they stand out of the scene as intense flow areas. Moreover, much part of the analysis of OF patterns employ these basic principles, as it will presented in 2.7.

2.4 Gibson Theory

Focusing on an empirical approach to the problem of visual perception Gibson [11] presented a description of what is considered the theory of direct or ecological perception.

In opposition to other theories, movement plays the most significant role in the process of perception. In fact, it is supported by the author that the elements of a scene in conjunction with the occurrence of movement are what enable the viewer to capture the setting of the surroundings. However, it is also firmly rejected the possibility of any processing by the viewer, thus suggesting that perception occurs only through sensorial inputs. Moreover, the elements known as optic arrays, texture gradients and affordance are introduced to explain how the brain accomplishes the direct interpretation of the scene.

Optic arrays represent, in fact, the light information that reaches the eye retina and displays the layout of the scene and its elements. They are best pictured using vectored inputs coming from

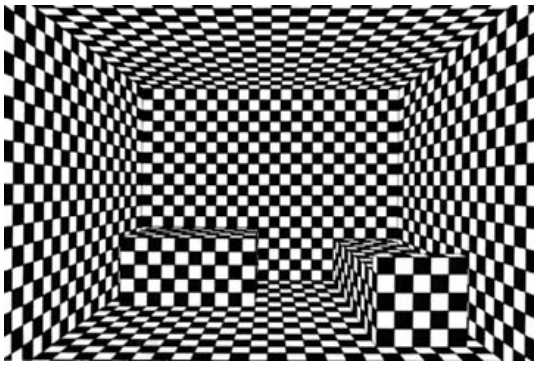


Figure 2.3: A simple texture mesh is altered in order to induce the perception of distance, position, perspective and size, from [1].

the different angles of the scene which are deviated due to any movement, of objects or viewer position.

The texture gradients designate the relative distortion of texture patterns that is observed either in static or dynamic scenes due to the spacing between objects and the viewer. Once again the perception is immediate and involves no processing. However, this parameter is described by two additional quantifiable references which are the point to which viewer is looking at or moving to (known as the pole), and, the horizon which is given by the height of the viewer.

2.5 Gregory Theory

In Gregory's reference publication [10], the author revisits some of Gibson's thoughts although he strongly disagrees with the assumption that perception derives directly from the visual information. In turn, it is stated that such data is frequently too impoverished in order to perception to take place. On this issue, Gregory makes a distinct contribution by supporting that perception is highly biased by motivation, emotion, past experience and expectations. Such is due to the fact that our brain identifies the simplest alternative of all possible interpretations, i.e., the most meaningful to us.

In addition, there were identified three types of memory, which were considered essential for perception. Each one of them stores different types of information and is characterized for having distinct physical properties. They are as described below:

- *Sensorial* Memory - highly volatile (fractions of second) used to store information that is directly inputted through eyesight, thus not subjected to any sort of interpretation.
- Short Term Memory - used store information about objects, namely colour.
- Long Term Memory - abstract knowledge which is the result of the human learning processes and intrinsic to thinking.

Figure 2.4 is an example of a visual illusion which is perceived differently according to the viewer. The explanation to this fact lies on the knowledge of shapes, sizes, colours (and movement

Figure 2.4: In the picture it can be seen either a young or an old woman, according to the viewer - this is due to certain affinities that are established with the shapes and colours of the picture.

patterns though it is a static figure) which are learned during our lifetime. This information is collected accordingly into each of the referred memory stores.

Although the author makes accurate observations about visual perception, he does not avert a *top-down* and rather empirical approach in his descriptions.

2.6 Marr Theory

Marr's approach [12] undertakes a structured and algorithmic description about the visual perceptive system. Evidences from neuropsychology and psychology are bridged to the laws of the design of efficient and robust applications in the field of artificial systems.

In fact, avoiding falling back in general purpose descriptions, Marr [14, 13], argues that a comprehensive theory of visual perception should be structured in three levels of abstraction, as described next:

- Computational Level - describes what tasks the visual system performs.
- Algorithmic Level - identifies and details the tasks involved in the perception.
- Hardware Level - lays out the neuronal mechanisms that perform the perceptual processing.

According to his theory, object recognition is an essential process to visual perception and it is, itself, comprised of a series of sub-processes. Each of these sub-processes, as illustrated in 2.5, draws on the representation produced by the one before, in an univocal flow.

The first step, known as the primal sketch, captures variations of light intensity in the scene, detecting mainly regions and boundaries. The $2\ 1/2$ D sketch interprets the layout of objects in the scene, in terms of their relative positions and distances to the viewer. The last step specifies the shapes of objects and is therefore known as the 3D Model Representation. The process of object identification is achieved by matching their shapes to representations of objects previously learnt by the brain and held in memory.

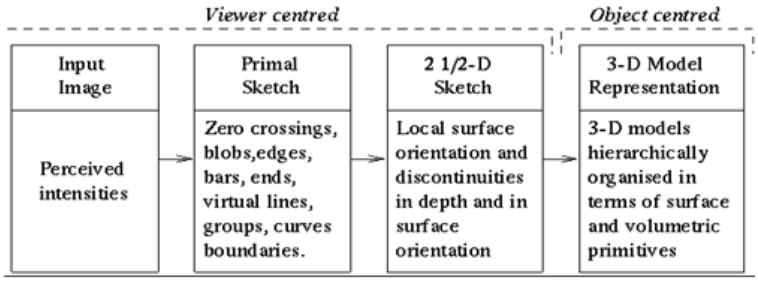
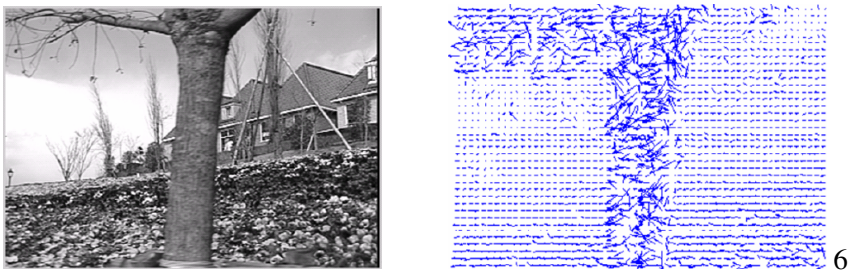


Figure 2.5: Marr's Representational Framework [2].

Finally, Marr's contributions were important to the clarification of visual perception and impelled the further development of his Computational Theory of Visual Perception.

2.7 Optical Flow

The concept of optical flow refers to an empirical phenomenon that occurs when we move. During motion, objects translate in the scene and we experience that those which seem closer to us move faster and backwards, while the more distant appear static or moving slower.



(a) 10th frame of the Flower Garden sequence.

(b) Corresponding optical flow.

Figure 2.6: Example of a measured Optical Flow field extracted from a test sequence known as the Flower Garden, represented with motion vectors, from [3].

Movement quantification is achieved by calculating, for each pixel, a vector indicating speed and direction. The representation of these two variables on flow maps as seen in figure 2.6 is very frequent. As an alternative, those variables are represented in an intensity map, as seen in figure 2.7.

The long study and experience acquired from the multiple implementations of OF estimators have made possible an empirical understanding of the generated patterns associated to the various types of movements. However, these patterns are not uniform and their matting forms up what is named the OF map of a scene. These maps represent the scene movement in relation to the viewer, as if it always moving around him and not otherwise. Moreover, regions with different intensities are usually distinguishable and can be used to warn of the presence of *moving* objects.

Figure 2.7: Example of an Optical Flow intensity extracted from a test sequence, from [4].

In the first place, it is relevant to state that, on an OF map referring to a scene in which the viewer is moving forward, it occurs that the peak of intensity is located in the border of the frames. This is due to the fact that they are situated nearer the orthogonal plane to which the observer is moving.

The information that can be derived out the flow map is, as referred throughout this chapter, not enough to accomplish a secure UAV (Unmanned Autonomous Vehicle) navigation. However, by knowing the meaning of specific patterns and some other system variables it is viable to make complex inferences about movement. Let's take the example of an UAV moving in a forward direction to better describe how OF maps are used. Hence, if the UAV OF estimator, for instance, detects an increasingly larger flow below and speed is known to be constant, then it could mean that the aircraft is dangerously losing altitude. The same applies if a region stands out of the background - it might be a hint that it is an object in motion and thus its direction and speed may be inferred by analysing its flow to determine if the object is an impact course. These conclusions are however highly influenced by the UAV current state, such as its speed, direction, previous position, etc..

Beyond obstacle detection, OF also provides a reliable way to measure and characterize motion. In figure 2.8(a), it is pictured an aircraft travelling forward and the corresponding OF pattern in a 360^{o} flat map. The Focus of Expansion (FOE) tells which direction it is flying. This way, the FOE is ideally described as the direction defined by the point where OF is null and the position of the viewer - in the referred picture, it is directly ahead. Thus, in this set, the OF diverges from the FOE and flows backwards where it converges in the opposing flying direction. The most intense flow occurs at 90^{o} and *up* and *down* directions.

On the other hand, figure 2.8(b) represents a left turn. The resulting pattern has the opposite movement. Flow intensity fades as the viewing direction nears *up* and *down* positions, but is constant horizontally. It is also interesting to observe that the peak intensity occurs, in this case, at eye-level and that the FOEs situate at top and bottom. The combination of the two types of OF patterns could reproduce more complex movements of the aircraft.

In a static environment, as a landscape with plenty of obstacles, the resulting OF pattern of the UAV would indicate alarmingly intense regions. An UAV could then act in order to reduce them by slowing down in order to increase a safer response time. This would cause the background flow

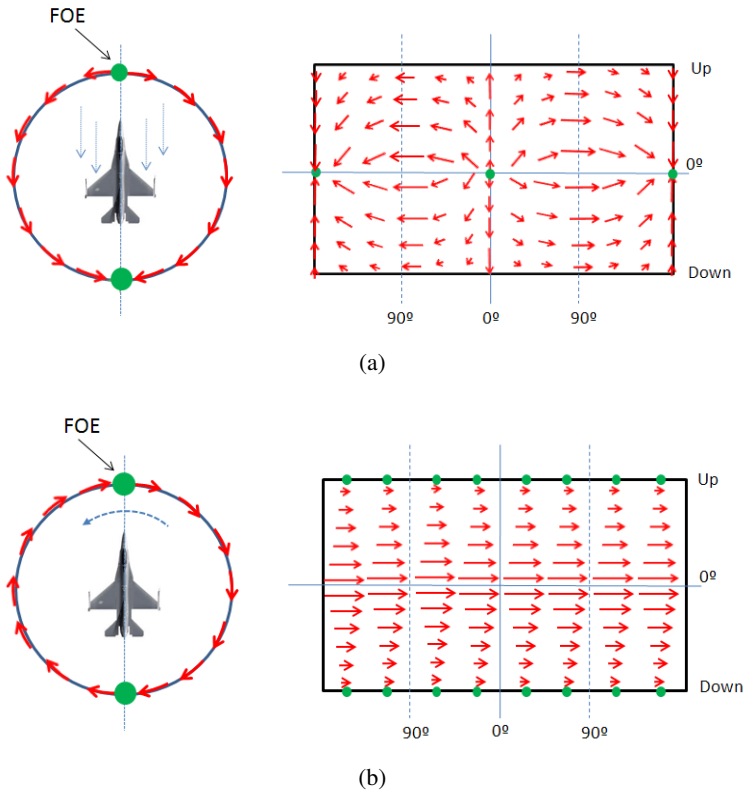


Figure 2.8: Two typical OF patterns represented in a 360º flat map.

to diminish, giving the navigation system a better perception of the surrounding environment, as we would do in a natural way. In addition, maintaining OF null in all directions would allow it to hover over a relative position to the surrounding scene objects.

The use of OF maps in real navigation and orientation systems has contributed to the knowledge of the different OF patterns associated to movement characteristics. The analysis of these patterns remains the simplest way to verify how much information they can offer.

2.8 Concluding Remarks

The theories that were alluded in the extent of this chapter picture a sketch of what visual perception is. As far as the purpose of this thesis is concerned, it is important to understand the theoretical and historical background of perception to realise why the estimation of Optical Flow is such a relevant source of information.

The concept of Optical Flow is nothing less than a confirmation of the utter importance of movement to visual perception, as Gibson and Gregory suggested. Moreover, it represents the division of the problem of perception into smaller subjects, being motion perception one of them.

Optical Flow maps derive from Gibson's concept of optic arrays added with speed significance. Throughout the years of experimentation many OF patterns have been studied and associated to

different types of movement, fact that has simplified the task of OF interpretation. To artificial systems calculating OF it presents a good trade-off between processing-time and density of outputted data that is simple enough to be post-processed.

However, one must have in mind that OF maps are not enough to emulate the human visual perceptive system. They do represent another step which will be drawn on.

Chapter 3

Optical Flow Estimation

3.1 Introduction

This chapter comprehends a considerable part of the work that has been developed in the extent of the state-of-the-art research. There is a vast amount of algorithms and implementations which relate to OF. The next subsections summarise the existing methods in their theoretical background as well as in their performance and relevancy to this thesis.

The implementations which were studied and tested will also be subject of analysis as well as benchmarking techniques which allowed differencing them.

It is the final purpose of this chapter to justify the design decisions, based on the alluded facts. This was considered a critical step in the project flow as much effort was spent to select a convenient algorithm which served the demanding features to stand out of other similar projects.

3.2 Algorithms' Hierarchy

The present amount of publications which introduce different techniques is immense. These are based on varied theoretical assumptions which difference them in their performance and practical results.

The following sub-sections attempt to explain how the different types of algorithms are conceptually organised and in what premises they stand on to calculate optical flow.

As figure 3.1 illustrates, there are four main general types of algorithms which are based on gradient, correlation, energy and phase calculation. In the course of the research, three specific techniques were studied, namely Horn & Schunck, Lucas & Kanade and Camus algorithms. In addition to this, some specific implementations of these algorithms were also studied and will be detailed in section 3.4.

3.2.1 Gradient Based Algorithms

The use of gradients to calculate optical flow requires some constraints to be met. These were first identified by Horn & Schunck [15] in what is considered the reference publication of the subject.

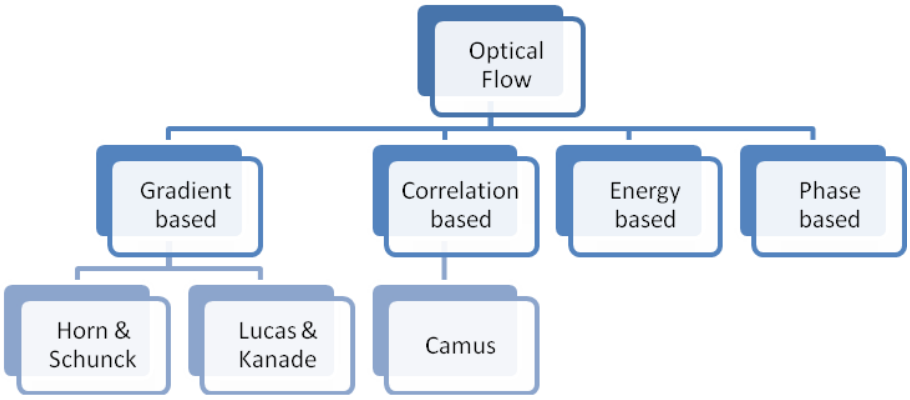


Figure 3.1: Optical Flow algorithm's hierarchy.

One of the main assumptions is that incident illumination is uniform across the surface of the elements of the scene, to prevent the existence of confusing shadows or reflections. In addition to this, it was established that object reflectance varies smoothly and has no spatial discontinuities. These conditions guarantee that image brightness is differentiable in every pixel, without the occurrence of any singularity which would have to be dealt in special manner.

These constraints are necessary because gradient calculation requires the brightness information of neighbouring points since any movement will cause the brightness patterns to alter locally. However, if a pattern occurs, in which brightness changes only with the x coordinate, then movement cannot be detected in the y coordinate. As a result, for movement to become detectable in its two components it is then required to establish two constraints.

3.2.1.1 Constraints

The first of the conditions states, as referred, that the brightness of a given pixel must remain constant in time, i.e., in motion. Hence, it is possible to conclude that:

$$\frac{dE}{dt} = 0 \text{ , where E stands for brightness.} \tag{3.1}$$

Denoting brightness as E(x, y, t), for a pixel (x, y), at instant t, the previous equation gives place to:

$$\frac{\partial E}{\partial x}\frac{dx}{dt} + \frac{\partial E}{\partial y}\frac{dy}{dt} + \frac{\partial E}{\partial t} = 0. \tag{3.2}$$

Considering that: $u = \frac{dx}{dt}$ and $v = \frac{dy}{dt}$, it is possible to relate brightness variations and movement, as the Optical Flow Conservation Equation (OFCE), by stating that:

$$E_x u + E_y v + E_t = 0 \,, \tag{3.3}$$

where E_x, E_y, E_t are the partial derivatives of image brightness to x, y, t, respectively.

Hence, rewriting equation 3.3 makes apparent how to find the components of the movement in the direction of (E_x, E_y) gradient:

$$\begin{aligned} (E_x, E_y).(u, v) &= -E_t \\ \Rightarrow (u, v) &= -\frac{E_t}{(E_x, E_y)} \\ \\ \|(u, v)\| &= -\frac{E_t}{\sqrt{(E_x^2 + E_y^2)}} \end{aligned} \tag{3.4}$$

The Smoothness Constraint, as it is known the second constraint, is necessary to infer about movement in contours in which brightness is constant, as previously explained. It assures that points do not move independently, rather translating in groups which represent, in fact, the various elements of the scene. The pixels that integrate these groups have similar velocities and movement directions, feature that originates the velocity field to vary smoothly along the image. Therefore, the presence of occluding edges should not pose additional difficulties.

In mathematical terms, the smoothness constraint is equivalent to minimizing the square of the magnitude of the optical flow velocity gradient:

$$\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \text{ and } \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 \tag{3.5}$$

An additional measure of the smoothness constraint is the sum of the square of the Laplacians of u and v equal to:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \text{ and } \nabla^2 v = \frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \tag{3.6}$$

Besides, it is also worth remarking that, in specific movements, these Laplacians will be cancelled, namely while moving parallel to a flat object, describing a circumference around an object or translating orthogonally to the surface.

3.2.1.2 Horn & Schunck

In sequence of what has been explained above, the Horn and Schunck method [15] makes direct use of the OFCE and Smoothness constraints (see 3.3 and 3.5) to calculate optical flow. As a matter of fact, this method calculates iteratively the best solution to both equations, which corresponds to the (u, v).

As discussed, the brightness gradient is essential to the calculation. To estimate E_x, E_y and E_t, the authors suggest using a sample cube formed by eight measurements, as in figure 3.2.

Hence, the estimation of E_x, E_y and E_t is achieved by averaging the four differences of adjacent measures in the cube. This is better understood if one has in mind that the variables represent

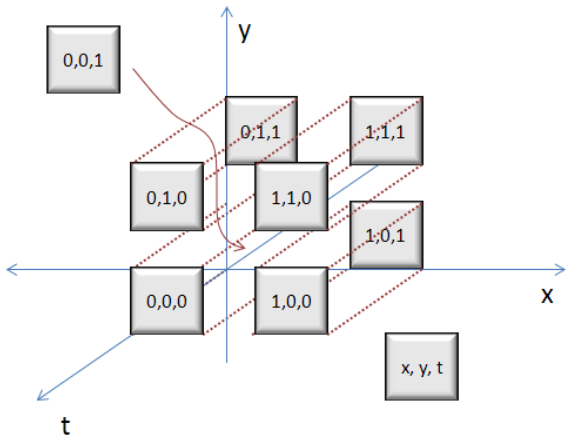


Figure 3.2: The sample cube which is required for the E_x, E_y, E_t gradients calculation.

derivatives in order to x, y and t which are being estimated by discrete points that are directly adjacent in the same direction. Thus the gradient can be calculated as:

$$\begin{aligned} E_x &\approx \tfrac{1}{4}\{E_{i,j+1,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i+1,j,k}\} + \\ &\quad + \tfrac{1}{4}\{E_{i,j+1,k+1} - E_{i,j,k+1} + E_{i+1,j+1,k+1} - E_{i+1,j,k+1}\} \\ E_y &\approx \tfrac{1}{4}\{E_{i+1,j,k} - E_{i,j,k} + E_{i+1,j+1,k} - E_{i,j+1,k}\} + \\ &\quad + \tfrac{1}{4}\{+E_{i+1,j,k+1} - E_{i,j,k+1} + E_{i+1,j,k+1} - E_{i+1,j,k+1}\} \\ E_t &\approx \tfrac{1}{4}\{E_{i,j,k+1} - E_{i,j,k} + E_{i+1,j,k+1} - E_{i+1,j,k}\} + \\ &\quad + \tfrac{1}{4}\{+E_{i,j+1,k+1} - E_{i,j+1,k+1} + E_{i+1,j+1,k+1} - E_{i+1,j+1,k}\} \end{aligned} \tag{3.7}$$

Furthermore, the authors propose an approximation to the Laplacians of u and v which is:

$$\begin{aligned} \nabla^2 u &\approx \kappa\left(\bar{u}i,j,k - u_{i,j,k}\right), \\ \nabla^2 v &\approx \kappa\left(\bar{v}_{i,j,k} - v_{i,j,k}\right), \end{aligned} \tag{3.8}$$

where κ is a proportional factor and $\bar{u}$ and $\bar{v}$ are local averages calculated as follows:

$$\begin{aligned} \bar{u}_{i,j,k} &= \tfrac{1}{6}\{u_{i-1,j,k} + u_{i,j+1,k} + u_{i+1,j,k} + u_{i,j-1,k}\} + \\ &\quad + \tfrac{1}{12}\{u_{i-1,j-1,k} + u_{i-1,j+1,k} + u_{i+1,j+1,k} + u_{i+1,j-1,k}\}, \\ \bar{v}_{i,j,k} &= \tfrac{1}{6}\{v_{i-1,j,k} + v_{i,j+1,k} + v_{i+1,j,k} + v_{i,j-1,k}\} + \\ &\quad + \tfrac{1}{12}\{v_{i-1,j-1,k} + v_{i-1,j+1,k} + v_{i+1,j+1,k} + v_{i+1,j-1,k}\}. \end{aligned} \tag{3.9}$$

The task of calculating the gradient is reduced to minimizing the error for the constraint equations:

$$\varepsilon_b = E_x u + E_y v + E_t, \tag{3.10}$$

$$\varepsilon_c^2 = \left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial^2 u}{\partial y^2}\right)^2 + \left(\frac{\partial^2 v}{\partial x^2}\right)^2 + \left(\frac{\partial^2 v}{\partial y^2}\right)^2 \tag{3.11}$$

In practice, ε_b becomes very hard to cancel due to intrinsic errors of quantization and noise originating in the frame capturing device. As a consequence, α^2 is introduced to act as a weighting factor and after some manipulation (for more detail refer to [15]) it is possible to attain:

$$\begin{cases} \left(\alpha^2 + E_x^2 + E_y^2\right)(u-\bar{u}) = -E_x\left[E_x\bar{u} + E_y\bar{v} + E_t\right] \\ \left(\alpha^2 + E_x^2 + E_y^2\right)(v-\bar{v}) = -E_y\left[E_x\bar{u} + E_y\bar{v} + E_t\right] \end{cases} \tag{3.12}$$

This equation system supports the fact that the final velocity $(u-v)$ will take the same direction as the brightness gradient, as it depends only of it.

Finally, the distance between the final and local average $(v-\bar{v})$ is concluded to be proportional to the error attained from the OFCE equation 3.3. Hence, this algorithm is prone to be implemented using an iterative approach. The weighting factor α^2, which was previously introduced, only becomes relevant in areas to which E_x and E_y gradients are small. The iterative solution takes the form of:

$$\begin{aligned} u &= \bar{u} - E_x\frac{\left[E_x\bar{u} + E_y\bar{v} + E_t\right]}{\left(\alpha^2 + E_x^2 + E_y^2\right)}, \\ v &= \bar{v} - E_y\frac{\left[E_x\bar{u} + E_y\bar{v} + E_t\right]}{\left(\alpha^2 + E_x^2 + E_y^2\right)}. \end{aligned} \tag{3.13}$$

3.2.1.3 Lucas & Kanade

The Lucas and Kanade approach [16, 17] is considered to be a method of calculating locally the optical flow. For a given window of size N×N, this method optimizes the $(v-\bar{v})$ movement to that specific set of pixels. This allows the authors to consider that the speed model is locally constant instead of globally smooth. Ultimately, if window size is equal to frame size this model could be extended to produce a *global* optimization.

Besides this, the error propagation is considered to be less and to exert less influence on the final result as it is confined to a specific N×N sized window.

Further research on this method [18, 19] suggests that using large windows leads to a good optimization of the movement components if, and only if, a previous characterization of the movement is known, otherwise the window may enclose various velocity patterns. By employing small sized windows, velocity pattern discontinuities do not occur since the movement is being characterized as infinitesimal as possible.

A common window size used in this technique is a 3×3 pixel grid, which results in a total of nine OFCE restrictions. As referred before, a larger 5×5 sized-window may be applied if the scene dynamic is previously known to fit in it. Furthermore, it is possible to extend the optimization process in time dimension by applying 3×3×3 window, summing a total of 27 restrictions, although this type of analysis will not be further explored in this document.

The corresponding stochastic formulation of this algorithm is:

$$\nabla I(x,t)\cdot v(x,t) + I_t(x,y) = 0, \text{ where I stands for pixel intensity.} \tag{3.14}$$

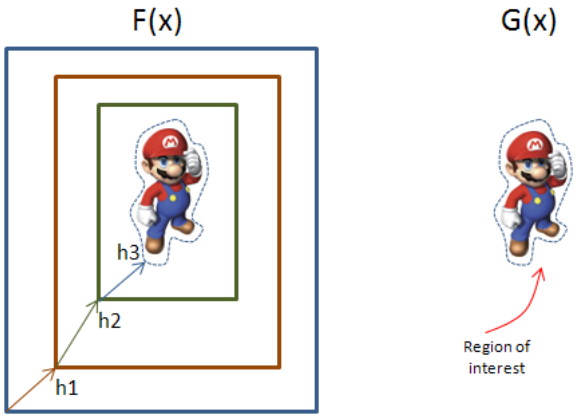


Figure 3.3: Newton-Raphson converging method.

The previous equation rewritten results in:

$$\begin{bmatrix} I_{x_1} & I_{y_1} \\ I_{x_2} & I_{y_2} \\ \vdots & \vdots \\ I_{x_N} & I_{y_N} \end{bmatrix} \times \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} I_{t_1} \\ I_{t_2} \\ \vdots \\ I_{t_N} \end{bmatrix} = 0, \text{ N is the window size.} \tag{3.15}$$

$$\begin{aligned} A.\vec{v} &= b \\ A^T A.\vec{v} &= A^T.(-b) \\ \vec{v} &= (A^T A)^{-1} A^T (-b). \end{aligned} \tag{3.16}$$

$$\begin{aligned} A &= \nabla I^T(x,t) \\ \vec{v} &= v^T(x,t), \\ b &= I_t(x,t). \end{aligned}$$

Lucas and Kanade argument that, by applying an error minimizing method, it is possible to compute an accurate solution. The authors use the Newton-Raphson method to devise an iterative algorithm. After some manipulation the following expression is derived (for further detail refer to [16]) to one dimension only:

$$\begin{aligned} h_0 &= 0, \\ h_{k+1} &= h_k + \sum_x \frac{\frac{w(x)\left[G(x) - F\left(x + h_k\right)\right]}{\sum_x w(x)}}{F'(x + h_k),} \end{aligned} \tag{3.17}$$

where h is the iterative hop, G(x) and F(x) are as pictured in figure 3.3.

Finally, most implementations of Lucas and Kanade algorithm employ the Gauss-Markoff theorem to compute a solution to the OFCE restrictions. Introducing a weighting factor in equation

3.16, it may be rewritten as:

$$\vec{v} = \left(A^T W A\right)^{-1} A^T W b. \tag{3.18}$$

The W matrix is a positive diagonal matrix, containing a set of empirical coefficients which weigh more the central pixels. They are presented next, under the form of a vector which represents the diagonal of matrix W:

$$\vec{w} = [0,0625 \;\; 0,25 \;\; 0,375 \;\; 0,25 \;\; 0,0625]^T \tag{3.19}$$

The solutions and minimizing functions are summarised in the following table, extracted from [20]:

Method	Cost function	Analitic solution
Classic Least Mean Squares	$\sum_{i=1}^{N} (I_{xi}u + I_{yi}v + I_{ti})^2$	$v = \left(A^T A\right)^{-1} A^T b$
Weighted Least Mean Squares	$\sum_{i=1}^{N} w_i (I_{xi}u + I_{yi}v + I_{ti})^2$	$v = \left(A^T W A\right)^{-1} A^T W b$

Table 3.1: Methods and respective solutions.

3.2.2 Correlation Based Algorithms

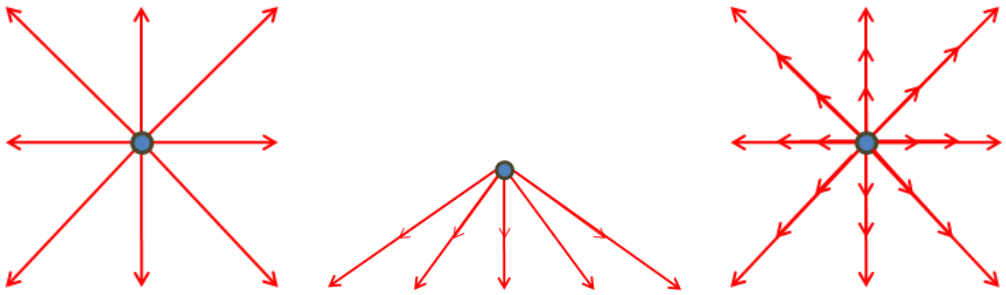
3.2.2.1 Camus

Camus approach [4] results in a correlation based method, which according to [7], is a robust and one the fastest general purpose implementations of optical flow algorithms. It is essentially an upgraded block-matching technique that determines a measure of coincidence for a window of $(2\eta+1) \times (2\eta+1)$ pixels of likely displacement. The η parameter is related to the scene's maximum expected motion, as it defines the radius of the search of the algorithm. The matching is accomplished by *moving* the patch of pixels which form the reference window in all possible displacements. By devising a simple matching as, for example, the sum of all differences, the most probable displacement can be determined. If M((x, y); (u, w)) is the matching strength for a pixel at (x, y) position and affected of a (u, w) displacement, that is calculated by a matching function ϕ, then Camus' mathematical model resumes to the following:

$$\forall (u,w) M(x,y;u,w) = \sum_{(i,j) \in P_v}^{n} \phi(E_1(i,j) - E_2(i+u, i+w)) \tag{3.20}$$

In addition, any of the pixels involved in the calculation above, for a given pixel, come from a total universe of a $(2\eta+1) \times (2\eta+1)$ image patch P_v of size (i, j). It is then possible to estimate the matching strength of two given windows by doing a simple difference between the brightness E_1, E_2 of the pixels, sharing the same relative position. E_1 refers to the brightness of the pixels of the reference window while E_2 regards the pixels of the displaced window. The candidate window

to which the maximum match strength concerns is said to represent the occurred movement, thus making it a "winner-take-all" algorithm. Figure 3.4 shows different searching patterns.



(a) The most common pattern.

(b) Alternative pattern.

(c) Pattern with linear time complexity though known for producing bad results.

Figure 3.4: Different searching patterns presented in Camus' algorithm.

Since this algorithm is based in the local information that is represented by the P_V search area, it becomes resistant to noise and less prone to be affected by the aperture problem (section 3.3.1). In fact, any unpredictable effect that influences either globally or locally the image, is usually over past by this method because the reference and the candidate windows share inherently the same characteristics in terms of pixels' disposition and intensity, thus originating smooth optical flow fields. On the other hand, gradient-based algorithms, do not share the same flexibility in such affected environments, thus behaving worse due to the fact that they are more sensible in their numeric analysis and rely on ideal constraints to be met. Camus algorithm requires only that a match with sufficient strength to be found.

The simplicity in concept and calculation invites the implementation of this algorithm in massive parallel processing cores. However, its complexity depends greatly on the search area, as well as the maximum velocity that is measurable by the algorithm. Truly, if the expected maximum speed doubles, the radius of search doubles consequently but the complexity of the algorithm grows quadratically. This is related to the fact that the search area is approximately a circle centred at the pixel in question, with a radius that equals the maximum expected speed. In order to maintain real-time performance, the author reveals two possible choices to tackle this problem.

Firstly, it is necessary to take in consideration that velocity is the derivative of distance in order to time and, additionally, that the speeds, distances and time involved are represented only in discrete values. As consequence, this algorithm can be modelled to search over variable distances, while the inter-frame delay δt remains constant. This is equivalent to state that:

$$\Delta v = \frac{\Delta d}{\delta t}, d \leq \eta. \tag{3.21}$$

In this case, the complexity quadruples, although the range of detection becomes the double. Viewing the velocity derivate in an alternate way:

$$\Delta v = \frac{\delta d}{\Delta t}, d \leq \eta. \tag{3.22}$$

This case favours algorithm simplicity, as it avoids the quadratic increase due to spatial search. However, this point of view also allows searching for the same radius with the implication that the system must be altered to accommodate a longer search in time. As before, the most probable displacement continues to be represented by the strongest match of all possible shifts.

A successful setup suggested by the author is $\delta d = 1$ pixel, searching over integer delays $\Delta \in \{1,2,3...S\}$, where S is the maximum time delay allowed and limits the slowest detectable speed as being $\frac{1}{S}$ pixels per frame. Consequently, slower moving objects take more time to be detected, a maximum of S frames.

Furthermore, this technique also requires that an object, moving at the slowest speed, must remain in motion for a minimum of S frames, otherwise it risks suffering from temporal aliasing and temporal aperture problems. Although these are not negligible issues, it can be demonstrated [4] that linear search in time ($\delta d = 1$ pixel) with a high frame rate is more efficient than a quadratic search in space using a slower frame rate, i.e., enough to compensate for that constraint. The algorithm's performance is known to improve accordingly to the maximum S allowed, under these circumstances.

3.2.3 Other methods

Energy based algorithms try to apply a set of filters specifically tuned to different speeds, and are therefore also referred to as frequency filters methods. It is assumed that movement has a spectral layout that can be tagged and understood by its density.

On the other hand, phase algorithms explore the same concept as before. However, they concern about local phase attained from special image filtering. The phase evolution represents a pattern that can be observed and translated into optical flow vectors by the system computing it.

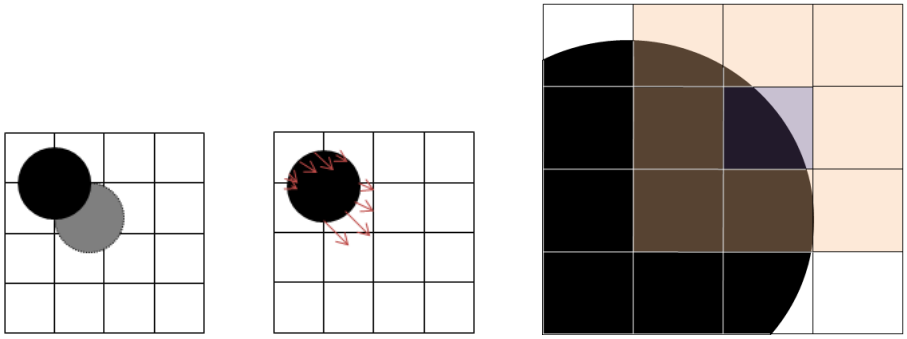
These methods are not, however, the most widely referred in the majority of the papers concerning this subject.

3.3 Known Issues

3.3.1 Aperture Problem

The aperture problem lays on the texture flatness of an image or object in a scene. As discussed before, optical flow calculation is based on the brightness information. If brightness variation does not occur, as in the body of the circle pictured in 3.5, the algorithm will not detect and calculate correctly the movement. The brightness gradients in this internal region of the object are null. However, when the algorithm reaches the object's border it immediately detects peak variations due to the natural distinction with the background. In the referred figure, a black circle is pictured

moving across a grid. Figure 3.5(b) shows that the optical flow vectors are calculated only in the border of the object, as explained before. Gradient algorithms would calculate flow around the object border, in discrepant directions to the real motion of the object.



(a) Image 1. (b) The corresponding Optical Flow field. (c) Closeup on 3.5(a).

Figure 3.5: Sequence of images in which the aperture problem is demonstrated.

Camus method is more resistant to this kind of problem. Figure 3.5(c) shows a close-up on the circle's border for explanation purposes, though its representation is not exact since a pixel can only take a single value. The 3×3 coloured grid shows a Camus reference window. This method only cares about how similar the selected grid is to another one. Thus, the aperture effect is substantially solved as the same composition of pixels is present in past frames, independently of its disparity with the background, resulting in positive matches.

However, performing a variable time interval search, Camus algorithm is prone to be affected by the *temporal* aperture problem. This happens, in rare occasions, like the one pictured in figure 3.6 in which the subsampling effect leads to erroneous representation of objects. It is shown a triangular object movement downwards the sampling grid. The resulting apparent motion leads to an inconclusive flow by the algorithm, since the pattern alters in an incongruent manner in relation to true translation. Still, Camus [4] claims that implementations using a matching function based on the sum-of-absolute-differences (SAD) are capable of performing correctly in these situations.

3.3.2 Aliasing Problem

Traditionally an inconvenience in image or signal processing systems, this effect is caused by the inherent loss of information due to sampling errors generated by inadequate sample intervals. The associated flickering of the images is explained by figure 3.7, where a pixel moves along the pixel grid.

The sampling *spreads* the pixel brightness by two other adjacent pixels, when the motion makes it overlap two blocks of the grid. This discrete representation of the pixel motion, leads to

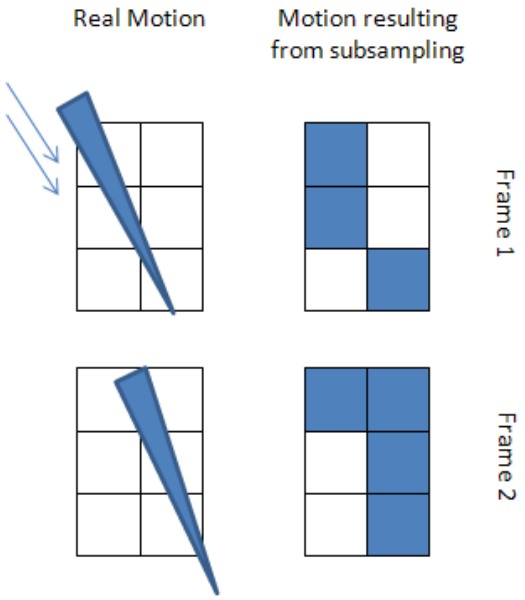


Figure 3.6: Aperture problem originated by subsampling.

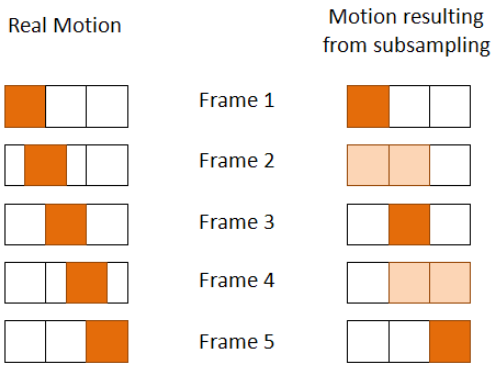


Figure 3.7: The image to the right shows an aliased version of the left one.

attaining less representative brightness patterns. If this effect is significant, gradient based algorithms perform worse, due to the somewhat abrupt brightness variations, than correlation based algorithms as these search for *blind* patterns, i.e., window similarities.

3.3.3 Occlusion

The occlusion problem is related with the way elements in the scene move, thus in the viewer's perspective, standing sometimes in the way of other objects. This usually prejudices optical flow algorithms since they require brightness information from neighbouring points either in space or in time. In gradient based algorithms, the occlusion of objects affects the accuracy of the results if the background or foreground is strikingly different from their texture, only due to the fact that the OFCE and smoothness constraints are not met.

Correlation based algorithms are inherently protected against this sort of spatial occlusion. However, as they search in along the time dimension, the phenomena of *temporal* occlusion might occur. It is basically a normal occlusion that takes place in past frames but, owing to motion, the object in question is disoccluded in the present. Consequently, when the search for a candidate window is initiated, the most recent information that is available is not in the past frames, no matter how much the algorithm deepens its search in time or range. As a result, no truthful match is found, ruining flow calculation. The converse is not possible to happen, because the full object *information* remains in the past frames.

3.4 Real-Time Implementations

One of the major goals of this thesis is to implement a real-time system capable of determining optical flow. As a result, the extent of the research was widened to determine what existing implementations of the various algorithms were available. Hence, it was possible to better understand how such systems were designed in practice. This resulted in attaining more information for the design of this project in terms of algorithm and system performances.

The different types of implementations studied are described next, although the information available about them is neither detailed nor clarifying in most of the publications that present them.

3.4.1 Horn & Schunck Implementation

This estimator, referred in article [5], uses a direct implementation of the Horn & Schunck algorithm and is therefore a gradient based estimator.

The system was implemented in an Altera FPGA (Field Programmable Gate Array) device EP20K300 (300 000 equivalent gates) with an overall throughput of 60 frames per second with a 256×256 image resolution.

The system setup is comprised of a memory structure, image input subsystem, the main estimator, and image output, as figure 3.9 shows.

To achieve optical flow calculation it was used the iterative model, already presented in equation 3.13. A simple sample cube $2 \times 2 \times 2$ was selected in order to calculate the spatial and temporal gradients. To hold information about pixels in previous frames it was required a FIFO memory structure of size:

$$MemorySize = (n-1).(1 + L + L \cdot H), \tag{3.23}$$

where n refers to the sample cube dimensions, L is the length of the image and H the corresponding height.

In order to optimize the access of pixel information for parallel computations, a FIFO structure was designed with external memories in a similar structure to the one in figure 3.8.

In order to implement the optical flow estimator, the system was divided in minor modules each corresponding to specific steps of the calculation. Analysing the following equations in

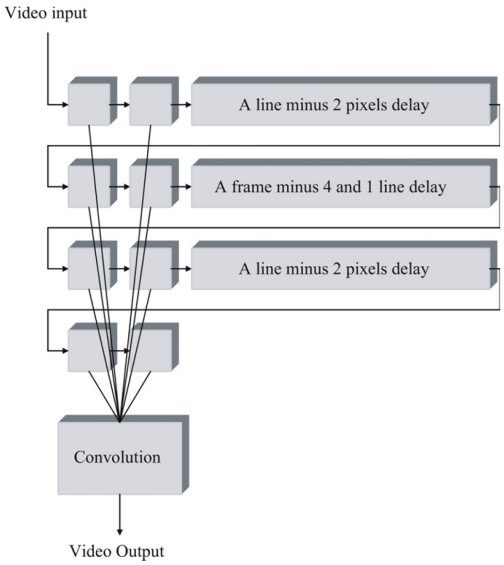


Figure 3.8: Memory architecture used to store pixels in different frame delays, from [5].

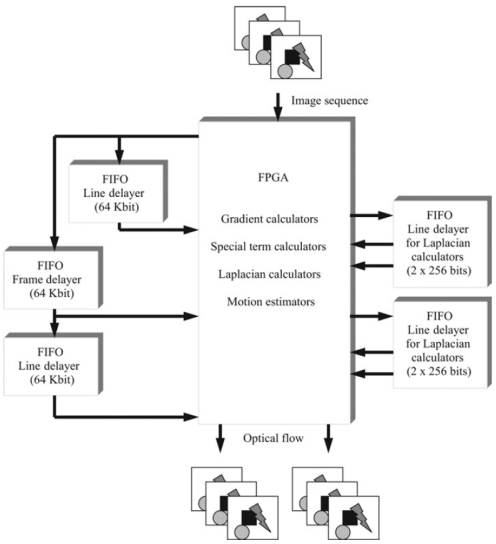


Figure 3.9: Top level architecture of the project's hardware setup, from [5].

conjunction with the iterative model present in equation 3.13 and figure 3.10, it is possible to understand how this partition was achieved:

$$\begin{aligned} P_x &= E_x(E_x\bar{u} + E_y\bar{v} + E_t), \\ P_y &= E_y(E_x\bar{u} + E_y\bar{v} + E_t), \\ R_x &= \frac{P_x}{\alpha^2 + E_x^2 + E_y^2} = \frac{P_x}{D}, \\ R_y &= \frac{P_y}{\alpha^2 + E_x^2 + E_y^2} = \frac{P_y}{D}, \\ u &= \bar{u} - R_x = \bar{u} - \frac{P_x}{D}, \\ v &= \bar{v} - R_y = \bar{v} - \frac{P_y}{D}. \end{aligned} \tag{3.24}$$

By simplifying each stage of calculation, the estimator is capable of achieving a higher frequency of operation and working with a pipelined dataflow, which is most suitable due to the high number of pixels that are required to be computed. Firstly, exterior modules determine the brightness gradients E_x, E_y, E_t which are, afterwards, input into a cyclic process (with fixed number of iterations). The chain of calculus that follow is divided in its u and v components (respective P and R terms) in order to allow parallel operation. At their end, the local averages $(\bar{u}, \bar{v})$ are back forwarded to initiate the next iteration, thus creating the intended numerical approximation method. As stated, for each pixel, the process is repeated a fixed number of times, value that is suggested empirically. The pipelined nature of this architecture allows the authors to claim better performance results, although it can only be thought for each iteration individually. This system produces accurate results in spite of using only integer variables for.

The FPGA device occupation is as follows:

Characteristics	Available	Used
Logic Elements (LEs)	11 520	7 018
RAM bits	147 456	12 288
User I/O pins	152	69

Table 3.2: Summary of the FPGA device occupation.

3.4.2 Camus Implementation

The Camus based hardware implementation described in [6] is a real-time system estimating optical flow on 96×96 pixel images, at a rate 22,5 frames per second. The employed FPGA hardware

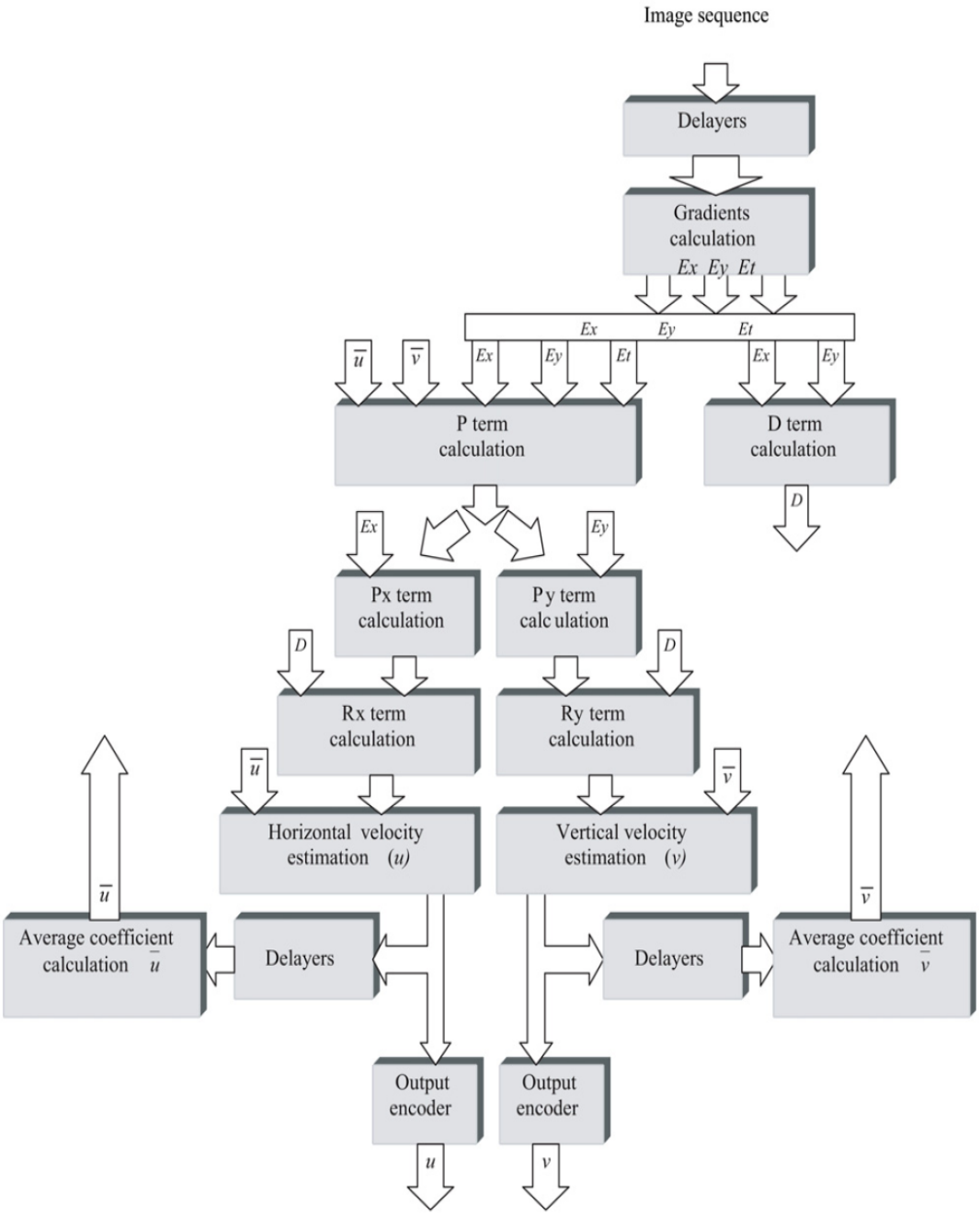


Figure 3.10: Suggested data flow diagram of the Horn & Schunck estimator, from [5].

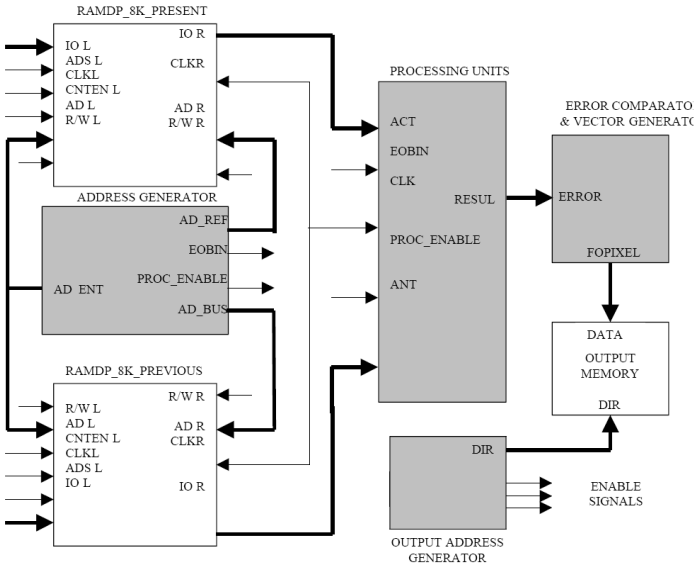


Figure 3.11: General description of the hardware design - a core that generates the flow vectors and three peripheral memories, two for algorithm calculation and one last for flow estimation storage, from [6].

platform is an Altera EPF10K50 working at approximately 15 MHz frequency.

Although this article is not explicit about how the systems works, in 3.11 it is possible to conclude that two RAMDP 8Kbyte memories are used as input buffers to allow calculation time and some line storage. These devices are double port RAM memories in order to allow faster line handling. A larger memory at the end of the data flow holds the calculated flow. The processing core seen in this figure is implemented, as the authors state, by the Address Generator, the Processing Units, the Error Comparator & Vector Generator and finally the Output Address Generator.

The two 8KB memories are used to store the previous and current lines. They are read in parallel to ensure no time loss, since lines share the same address and differ only in timestamp. This is essential, as Camus algorithm is a correlation based process that searches in the time dimension. If more precision was to be required, thus augmenting the number of frames that needed to be stored, more of these devices should be used, sharing the same parallel accessing method.

The *Address Generator* is responsible for generating the next address to be read, while the Processing Units are busy measuring the displacement likeness of an image patch.

The *Processing Units* is assumed to contain all the necessary blocks to implement a Camus algorithm. In order create a fast way to calculate correspondence between blocks of pixels, the authors devised a read protocol. When accessing memory for a given line, there is a mechanism that keeps track of what lines are being viewed and the ones that are not needed for that computation. By doing so, it is possible to use the freed positions to write new lines from the camera, taking advantage of the double port feature. This creates a virtually continuous correspondence block and thus the process can be pipelined.

Finally, the *Error Comparator* unit generates the OF vectors by comparing the match strengths of all possible displacements, data that is attained from the *Processing Units*.

3.4.3 Similar Project

The article presented by [21], discusses the implementation of a tensor based (also based on gradients) optical flow estimator. Although the mathematical algorithm that supports this work is out of the extent of the research of this thesis, it is relevant to stress the current performance of recent implementations of Optical Flow estimators.

This system can process 640×480 images at a rate of 64 frames per second. It is implemented on a Xilinx XUP V2P hardware platform which contains a Virtex-II Pro XC2VP30 FPGA and a 256 MB off-chip memory. The system works at a 100 MHz clock frequency and uses 10 288 slices out of a total of 13 969.

The importance of this work resides on providing a view of the potential and the amount of resources that are necessary to implement a system with similar features to the ones desired to the present thesis.

3.5 Benchmarking Optical Flow Estimators

The subject of evaluating Optical Flow estimators has been a side issue in this area. In fact, there is, until the present time, no systematic evaluation method to characterize each implementation. However, some publications have focused on this topic.

Mccane *et al* [22] have addressed this issue stating that large comparative studies like the one presented by Barron *et al* [23] are not the ideal form to proceed to such standardization. It is argued that replicating another author's work is always a biased task when implementing it from a text description. As a result, such implementations may not follow truthfully the original papers. Therefore, it is suggested to create a set of test sequences and error measurements that should be kept at a central repository. This would allow each implementation to be tested by recognized testing standards.

To access this issue, the present work will try to use standardized test sequences to characterize its results.

The following sections describe various test sequences, their aims and establishes a performance comparison.

3.5.1 Test Sequences

3.5.1.1 Synthetic Images

This group of test sequences is characterized for being precisely defined in their scene properties and thus producing highly methodical results. The following tests are extracted from [23, 24]:

- Sinusoidal Inputs: consists of the superposition of two sinusoidal plane-waves. This originates a chess pattern, see figure 3.12(a).

- Translating Square: consists of a black square moving over a bright background. This allows to evaluate algorithm response to some known issues as the aperture problem due to the sharp brightness transitions, see figure 3.12(b).

- Yosemite Sequence: involves moving the viewer between mountains and horizon making it a complex test case. It tests aliasing problems and existence of various velocity patterns, see figure 3.12(c).

- Diverging Trees: the scene which involves a tree with a bright background, moves along the line of sight, see figure 3.12(d).

- Sphere: a sphere rotates on its own axis over a striped background. It is a simple test case where object motion detection is evaluated, see figure 3.12(e).

- Office: an office scene is pictured in an expansion motion, creating a medium difficulty dilatational pattern, see figure 3.12(f).

- Street: in a city environment a car makes a turn, synchronised with camera movement which follows it. This is a complex test case where there is object and background scene motion, due to camera rotation, see figure 3.12(g).

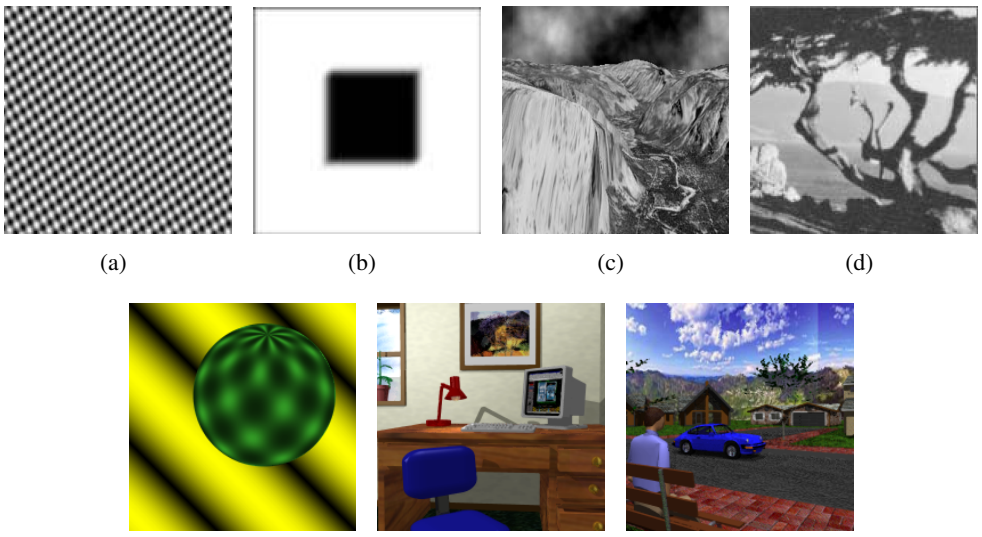
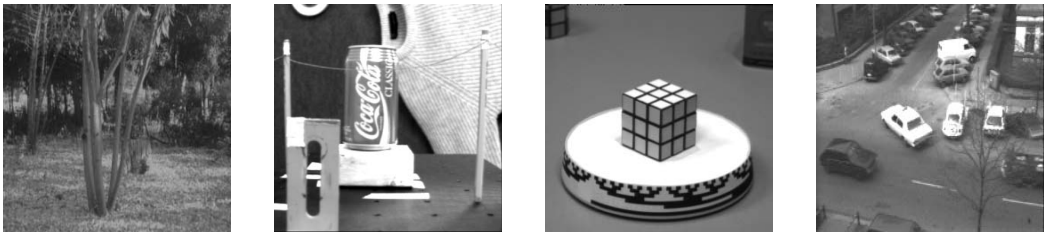


Figure 3.12: Frames from different synthetic sequences.

3.5.1.2 Real Images

Conversely to synthetic scenes, these test cases are characterized for their authenticity and less controlled environments. Next are described some test sequences, from [23]:

- SRI sequence: the camera translates in a parallel vector to the back scene and due to lack of resolution and contrast the estimator sees it as a blurry sequence, also because of the many existing trees, see figure 3.13(a).
- NASA sequence: camera moves in the orthogonal pane in the Coke can direction thus creating a dilatational effect, see figure 3.13(b).
- Rubik Cube: camera remains static and the Rubik cube rotates counter-clockwise, very demanding for estimators that look only at local gradients, due to the self-rotating motion, see figure 3.13(c).
- Hamburg Taxi: static camera scene where 3 cars and 1 pedestrian are moving, in different directions, creating a stir of vectors patterns the system must compute, see figure 3.13(d).



(a) (b) (c) (d)

Figure 3.13: Frames from different real sequences.

3.5.2 Accuracy and Performance

Optical flow estimators are, inherently, systems with real-time ambitions. Artificial visual perception depends, in part, on them being capable of successfully calculate flow out of real-world environments. Therefore, their design cannot be too specific as not to be restricted to controlled test cases but, instead, should be robust, fast and sufficiently accurate.

However, it is not simple to establish frontiers between those variables. For example, real-time requirements implicate that under no circumstances may robustness and speed be sacrificed in detriment of accuracy [4]. It is this trade-off established between accuracy and performance that has been sometimes overlooked in the existing publications, as they usually focus on the optimization of either speed or error minimization.

This work as alluded until now the most implemented and tested OF algorithms, namely gradient and correlation algorithms. The survey which is presented in [7], as it can also be perceived

from reading the previous sections, acknowledges that gradient based algorithms are generally more accurate than correlation based ones, at the cost of speed performance. The tests and analysis mentioned in this article were performed using a 80 MHz HyperSparc 10 board and the same implementations as Barron *et al* [23]. The test cases are the same as those have been presented in section 3.5.1.

The diagram, see figure 3.14, illustrates a 2D performance comparison in terms of processing time and direction error affecting the result. It is possible to confirm that gradient estimators take longer to calculate flow. As a matter of fact, Horn & Schunck is the best in terms of accuracy but the worst in speed. Camus estimator presents an error in direction that is always above ten degrees which, in some applications, could be unsatisfactory. As stated in [7], Camus approach is highly concerned with speed which explains why its accuracy-efficiency curve is stiffer.

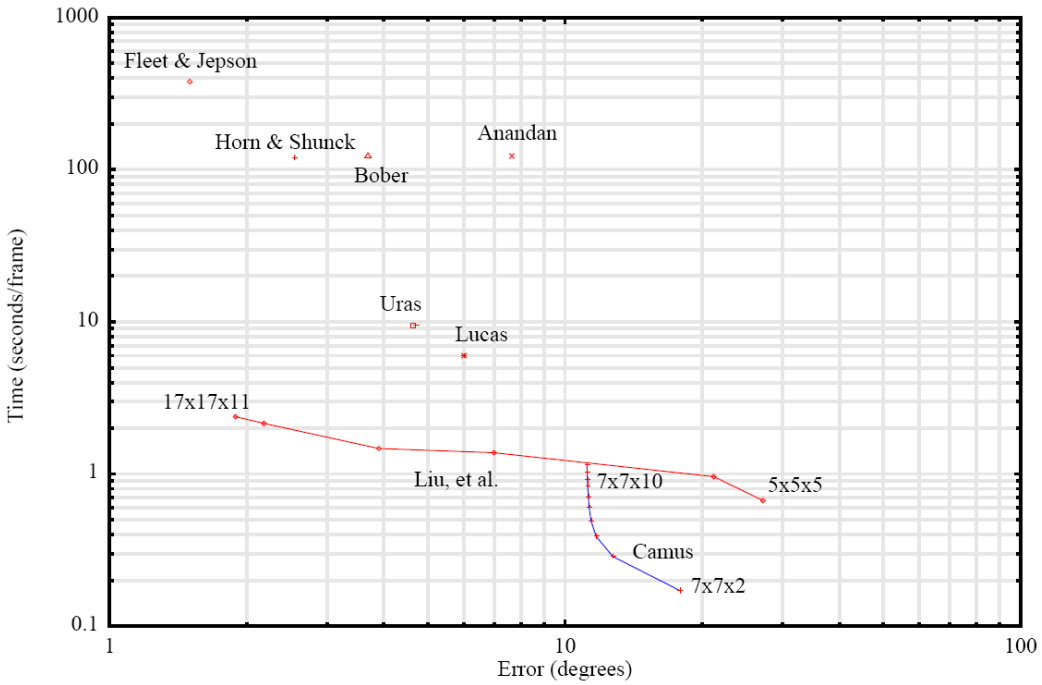


Figure 3.14: Accuracy-Efficiency diagram for Diverging Tree Sequence, from [7].

A comparison between Lucas & Kanade and Horn & Schunck shows the first to have a worse accuracy. Liu *et al* devise a interesting performance variation test that evaluates, among others, these two algorithms. The test consists of sampling frames at varying intervals and introducing them in the estimators. In such an uncontrolled test environment Lucas & Kanade is indeed better than Horn & Schunck, as it is capable of producing more consistent results.

In order to characterize systematically gradient and correlation algorithms, the following tables were elaborated:

Finally, although Camus Algorithm might give the impression that, due to its simplicity and quantitized results, it is not sophisticated enough, Camus [25] states that it is sufficiently coherent and accurate to compute time-to-contact robustly.

Gradient Algorithms	
Strengths	**Weaknesses**
Accurate Flow	Noise sensitive
Handles various motion velocities	Affected by subsampling
Captures expansion patterns	Latency due to frame delays
Occlusion is handled correctly	Requires floating point variables

Table 3.3: Gradient algorithms strengths and weaknesses.

Correlation Algorithms	
Strengths	**Weaknesses**
More robust to noise	Quantized Flow
Less affected by subsampling	Limits detectable motion velocities
No latency due to frame delays	Detects translation only
Uses integer variables	Occluded objects are distorted

Table 3.4: Correlation algorithms strengths and weaknesses.

Chapter 4

Estimator Implementation

4.1 Introduction

This chapter is essentially dedicated to describing the implementation process. In addition to this, the key design decisions which were made in the course of this work, are explained and justified. It was considered relevant to identify project constraints and requirements in order to achieve a robust implementation. The practical implementation, in which the main architecture is detailed, provides an insight about the system operating mode.

The descriptions taken follow an intricate line of thinking with previous modules and architecture overviews, as to replicate a *top-down* approach. For future reference, the term complexity is used to refer to associated implementational cost.

4.2 Project Constraints

The present section serves the purpose of making a survey of inherent limitations and requirements which affected the course of the practical implementation. Consequently, some of the facts that are presented concern only the target technology, thus any project decisions which are direct result of these constraints may not apply in different occasions. However, this thorough analysis may be used as a baseline for other similar works, which must weigh estimator's complexity and performance.

The calculation of optical flow, as stated in section 2.7, has its major field of application in robotic systems. However, if one has in mind the technologic limitations which are introduced by the implementation platform and project extent, it is important to identify the requirements that fulfil the application needs. Decisions regarding the project design must take into account these previous factors.

4.2.1 Available Technology

The intent of this project is, as stated before, to implement an optical flow estimator. Having in mind that image processing tasks are demanding in terms of processing requirements, it was

initially decided to implement the system architecture in a FPGA platform. Due to its high speed and programmable characteristics it fits the project design needs in terms of system throughput and flexibility.

The available hardware platform is a SUZAKU-S SZ130-U00 [26] which contains a Xilinx XC3S1200E FPGA chip. The board's most relevant features to this context are as described:

- System Chip: Xilinx Spartan-3 XC3S1200E
- Crystal Oscillator: 3.6864 MHz
- Configuration: JTAG or Linux netflash utility
- Pre-built: uCLinux running on MicroBlaze

The block diagram 4.2 illustrates the hardware architecture of the SUZAKU board. As it is presented, this platform is comprised of a FPGA chip, communication interfaces, data buses, memories and I/O pins. The FPGA chip is a System on Chip (SoC), since it integrates a 32-bit MicroBlaze processor embedded in resident logic (soft core), BRAM memories, Data buses and an I/O controller. This board is also supported by a developing environment.

The on-board FPGA chip is a powerful system and its attributes are summarised by the following table:

System Gates	Equivalent Logic Cells	Total Slices	D.[1] RAM	BRAM	DCMs
1200 K	19 512	8 672	136 Kbytes	504 Kbytes	8

Table 4.1: Summary of Spartan 3 1200E specifications.

There are three major types of memory on the SUZAKU board which are relevant to this work, namely the SPI Flash, the SDRAM and the Block RAM (BRAM). The first is used by the Linux Kernel and a high performance bootloader. From a total of 8 Mbyte (its organisation is presented in table 4.2, only 1 Mbyte is free in the *fpga* area to store the FPGA configuration bitfile. The SDRAM totals a 2×16 Mbyte storage area which is provided by two external memory chips, as shown in figure 4.1. The default project, which already implements a 32 bit MicroBlaze CPU, stores each half of a 32 bit word in each chip, for faster access. In spite of being possible to reconfigure the default XPS (Xilinx Project Studio) project to use only a one of the SDRAM chips and free the remaining one (see SUZAKU's hardware 1.0.2v manual at [8]), this task has proven to be very complex. Such is due to technical complications when reconfiguring the project to store and load 32 bit words from a single SDRAM memory. Finally, the memory space provided by BRAMS is limited to 8 Kbyte due to the embedding of MicroBlaze and the default XPS project.

In addition to these types of memories, it is also possible to use Distributed RAMS, which are implemented in the available user logic, thus, not physically. As a consequence, they were excluded from consideration as they would represent an inefficient slice occupation.

In terms of data transfer buses, there are four types of them:

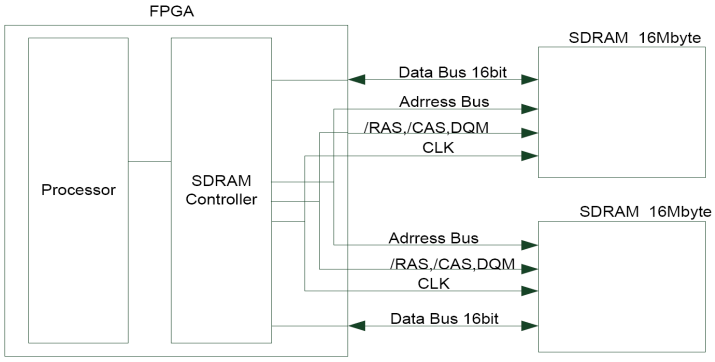


Figure 4.1: SDRAM Organisation in SUZAKU's board, from hardware 1.0.2v manual at [8].

Address	Region	Size	Description
0x00000000	fpga	1MB	-
0x000FFFFF			
0x00100000	bootloader	128KB	Hermit Boot loader
0x0011FFFF			
0x00120000	image kernel	3MB	Linux Kernel
0x0041FFFF			
0x00420000	user	≈ 3.81MB	User Area
0x007EFFFF			
0x007F0000	config	64KB	Configuration Area
0x007FFFFF			

Table 4.2: SUZAKU's board SPI Flash Memory Organisation, from the software 1.3.1v manual at [8].

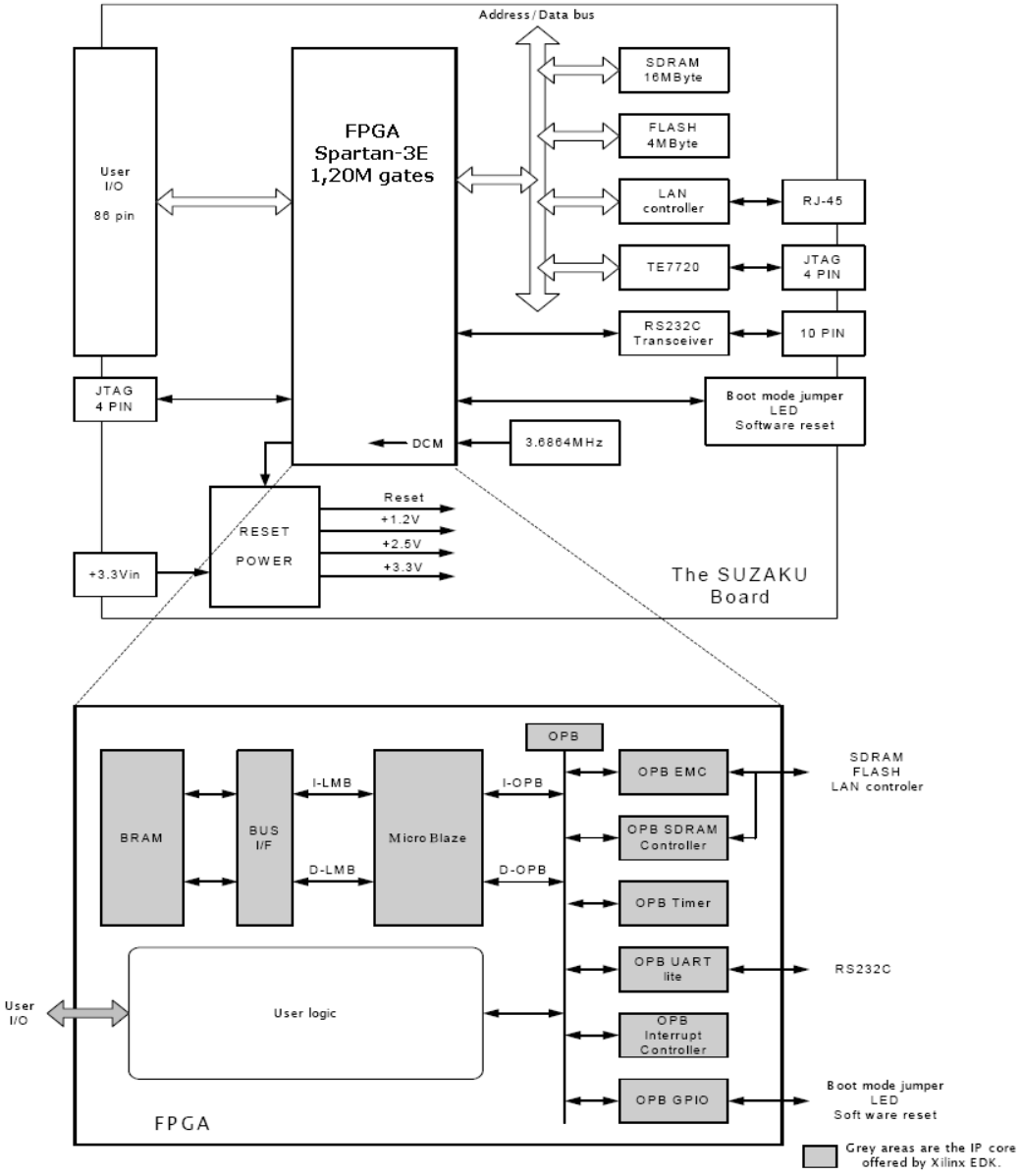


Figure 4.2: SUZAKU board architecture overview, from SUZAKU's hardware 1.0.2v manual at [8].

- PLB: Processor Local Bus (fast);
- OPB: 32-bit address, 32-bit data, On-chip Peripheral Bus (slow), connected to processor through PLB;
- LMB: Local Memory Bus (fast) connects MicroBlaze instruction and data ports to BRAM;
- FSL: Fast Simplex Link Bus creates a Point-to-Point, Single Master/Slave, unidirectional communication.

From these, OPB and FSL buses are the best suited for data transferring between MicroBlaze and an user IP core. The OPB bus is a fully synchronous bus which allows devices connected to it to be mapped as memory addresses. The complexity of the OPB is, however, greater than with the FSL bus. To minimize this issue, the OPB bus implements also a standardized OPB IPIF (On-chip Peripheral Bus Intellectual Property Interface) which uses a back-end IPIC (IP Interconnect) to simplify the connection between IPIF services and the final user logic. This is possible, however, at a user space cost. On the other hand, the FSL bus is supported by much simpler hardware. In fact, it uses FIFO (First In First Out) memories to transfer data from one core to the MicroBlaze, with the help of simple flagging mechanisms. The FIFO structures are available to be implemented either in LUTs or BRAMs. The inconvenient of this sort of protocol lies on the fact that it requires the CPU *to be* in the datapath of the bus. As a result, multithreaded operative systems or applications cannot make use of the blocking FSL operations since these could cause a hard lock of the system, for instance, when a read command was issued on a empty communication channel. In comparison, the OPB bus is inherently unaware of such issues and any read or write operation consists simply on accessing a virtual memory position, which frees immediately after the CPU to run other processes.

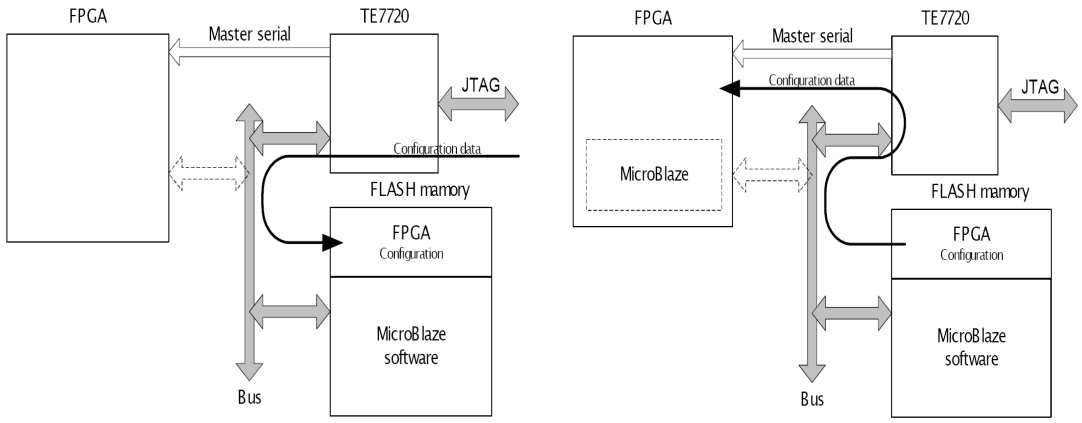
The *fpga* Flash area can be programmed using various methods. The most simple and recommended process consists on programming it using Linux's netflash utility which writes the configuration bitfile directly into the Flash memory, thus making possible a in-operation programming method.

By default, the start up process loads the new configuration into the FPGA. This is achieved, as figure 4.3 presents, by the action of the on-board device TE772, which configures the FPGA based on the bitfile stored in *fpga* Flash area (see 4.3(a)). In case of system corruption, it is possible to configure the FPGA by sending the bitfile directly via a JTAG interface as figure 4.3(b) illustrates.

4.2.2 Requirements

Robotic systems, depending on their purpose require more or less accuracy in the optical flow estimation. Usually, they take the form of slow moving robots which use OF for object detection. Thus, the accuracy of the estimator is not considered to be of the utmost importance.

On the other hand, real-time analysis is essential and consequently it is required that the rate of flow outputs meets the rate of incoming pixels. Additionally, it is relevant to process the full vision field of the on-board camera as to take advantage of all the available information, to improve



(a) Writing to Flash memory via the TE7720 device.

(b) Configuration of the FPGA via the TE7720 device from Flash memory, at power-up.

Figure 4.3: FPGA Configuration, from the SUZAKU's hardware 1.0.4v manual at [8].

perception in its action field. As to compensate less accuracy, calculating a more dense flow might also improve the resulting analysis.

Embedding the FPGA MicroBlaze soft core in the resident logic, with the associated resource costs, results in the following occupation:

Number of BUFGMUXss	3 out of 24	12%
Number of DCMs	2 out of 8	25%
Number of MULT18X18SIOs	3 out of 28	10%
Number of RAMB16s	14 out of 28	50%
Number of Slices	2660 out of 8672	30%
Number of SLICEMs	378 out of 4336	8%

Table 4.3: Summary of FPGA occupation with embedded MicroBlaze, with pre-built Linux.

As a result, the following specific objectives were identified:

- The rate of processing must be around 25 frames per second;
- The full camera field of view must be analysed, i.e. frames with resolution of 640x480 pixels have to be computed;
- The algorithm must output a flow map, 100% dense [2];

[2]Optical Flow is determined for each pixel

- Accuracy of the optical flow must be enough to allow post-processing.
- The estimator must fit in the remaining space of the FPGA user-logic space and use no more than its resources;
- The estimator must be accessible through a C framework, comaptible with Linux operative system.

4.3 Design Flow

The work developed in the course of this thesis consists of an initial research phase, a test and performance evaluation of the existing tools [22], followed by an implementational period. The research phase was needed to survey the current state-of-the-art in optical flow calculation and the tests which followed were vital to register the different performances and complexities of the algorithms. As a result of these tests, it became clear the importance of understanding the issues that are associated with the flow calculation. Considering the project requirements and constraints, Camus algorithm was determined to be the fittest choice. The implementation was carried through following a *bottom-up* approach, which permitted both an individual an iterative debugging of the system. The C framework and test application were successfully designed afterwards.

Due to hard technological constraints much effort was employed in devising alternative ways to implement a data transfer mechanism between the softcore MicroBlaze and the OF core. The initial design previewed a direct input from the CMOS cameras into memory structures, which was not viable since there was insufficient memory space. As a result, it was decided that the MicroBlaze would load the images and send via user application and through an inner FPGA data bus, each necessary pixel (from current frame plus frame history) to compute the flow. As an abstraction, three frames, adjacent in time, were sent in parallel to achieve that purpose. However, problems encountered in the final phase of programming the FPGA chip, inhibited the practical implementation of such design, see chapter 5. As a consequence, it was decided to abandon the practical implementation step and return proceeding to implement the system with a similar design to the initial one (with external memories). However, due to the memory limitation this architure could not be physically implemented it became necessary to use a truthful simulation model to validate it.

This project was developed using Xilinx development tools, namely ISE (Integrated Synthesis Environment) and XPS (Xilinx Platform Studio), both 10.1.03(nt) version. The first was used to synthesise and debug project modules and the latter was necessary to configure the FPGA and generate the corresponding programming bitfile to the SUZAKU board 4.2.1. The project was designed using Verilog HDL (Hardware Description Language).

A Ubuntu Virtual Machine was used as a C developing environment with cross-compiling tools for the embedded MicroBlaze. Additionally, ModelSim was used as the validation tool.

4.4 Algorithm Selection

The selection of the optical flow algorithm was considered an important step in the course of this work as it determined the further implementation decisions and the system architecture. This choice is obviously influenced, as referred in 4.2, by the target technology as well as the various constraints and requirements that have been discussed throughout this document. Therefore, this critical decision was supported by reference studies and articles which address the issue of Optical Flow benchmarking, examined in section 3.5. The initial research also focused on successfully implemented algorithms, with proven results.

In order to be more specific, the complexity of the algorithm had to be weighted as the FPGA user space is limited due to the fact that the embedded MicroBlaze is required. As a result, almost 30% of the slices are used, as table 4.3 indicates. Moreover, parallel calculation structures had to be considered in order to meet the 25 frames per second and 640×480 image size constraints.

In addition to this, Linux install uses the SDRAMs of the SUZAKU board as its main memory. This fact creates storage space problems if one has in mind that these algorithms require information about two frames, at least. Thus, the memory structure needs to be implemented within the FPGA logic with the minimum size, since it lessens vital available space necessary to algorithm accommodation.

The full information about the frame is not required for the flow calculation, instead, only the pixel grid or sample cube is needed, in the case of Camus or Lucas & Kanade (3×3) and Horn & Schunck (2×2), respectively.

As discussed in section 3.5.2, correlation based algorithms have less latency in calculation and usually have higher throughputs than their gradient based counterparts. However, the precision attained by the latter is not equalled by the first, although it comes at the cost of more noise sensitive, error minimizing functions. Besides it, they require the use of floating point variables which adds complexity latency. In fact, Horn & Schunck has a distinctively high latency. A previous comparison established in 3.5.2, selected Lucas & Kanade as a more interesting choice for real test cases, in what concerns gradient algorithms. Revisiting figure 3.14, Camus and Lucas & Kanade have contrasting performances in terms of efficiency and accuracy. The analysis of the project requirements indicates that accuracy is less important than speed, in the context of this work. Thus Camus algorithm resembles the best choice of all researched algorithms.

A suitable setup to this algorithm uses a 3×3 pixel grid. The employed time *depth* by Camus in [4] suggests $S = 3$ frames. The AE curve in figure 3.14, reveals, however, that the ideal time search is of about 10 frames. Due to memory limitations of the FPGA plataform, that is not sustainable, fact discussed in the next section.

4.5 System Setup

4.5.1 System Overview

The algorithm selection phase has selected Camus as the estimator. Camus method uses block matching techniques with minimum search windows of 3×3 pixels and, as discussed before 3.5, employs a 3 frame search in time. The displacement search pattern used is the one presented in figure 3.4(a). In terms of the match measure function, SAD was considered to be sufficiently accurate [4] and able to perform well with most of the known problems. It is also the one that requires the less hardware to be implemented. The implicated memory cost is approximately:

$$MemoryCost \;=\; 640 \;\times\; 480 \;\times\; Sbytes. \tag{4.1}$$

The system was originally designed to permit the CMOS camera to send pixels directly to a memory structure where they were later loaded into a shift-register memory-like module to reduce computation latency. However, the impossibility to use the board's SDRAMs proved this to be unsustainable to implement, since the intended frame resolution easily requires a space in the Mbyte magnitude. As a result, it was determined that an abstraction model would be used in which three cameras, separated by one frame in the time dimension, would send in parallel their frame information. This removed the necessity for a large memory area but added the problem of selecting a suitable data transfer method between the MicroBlaze and the OF core. The FSL protocol was identified as ideal for this matter for its simplicity of operation and low hardware cost.

The only remaining space cost was then related to the shift-register structure, which is as follows:

$$(S+1) \;\times\; 5\; lines \;\times\; 640 \;=\; 12800\; bytes, \tag{4.2}$$

using S = 3 frames.

As the practical implementation later showed, it was not possible to use S = 3 frames, since memory usage required too much logic, in proportion to the rest of the system description. Consequently, S = 2 was selected for the final implementation. However, neither of these structures sizes fits the available BRAM space.

Another issue which highly influenced system design was the circuit operation clock rate. As the initial objectives were, in part, to integrate the OF estimator in an image-processing system, it was necessary to account for CMOS cameras output rate which is about 12,5 MHz. Consequently, system was set to work at a 100 MHz clock rate in order to compensate with some clearance for the inevitable latency of the estimator.

The combination of the previously presented facts and requirements (see 4.2.2) led to the following general system architecture, presented in figure 4.4.

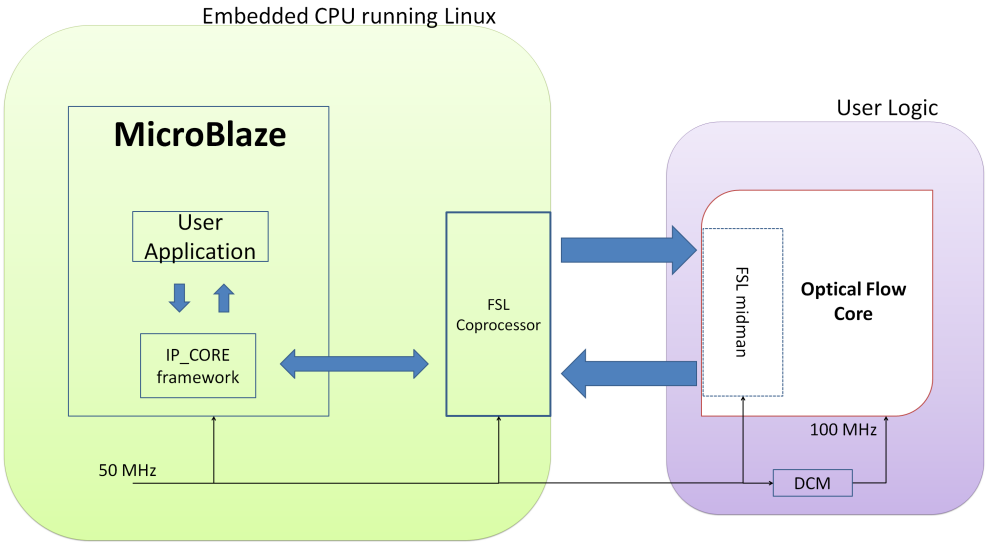


Figure 4.4: System overview.

4.5.2 Module Specification

This section is dedicated at discussing the inner working of the OF core. As figure 4.5 illustrates, the designed OF core is constituted by the SAD calculator, the FSL subsystem, the memory structure and some side modules.

4.5.2.1 SAD Calculator

The SAD calculating module explores parallelism in order to compute the best match in all dimensions. As figure 4.6 shows, the present reference window is placed side to side with the displacements referring to different frames and to search pattern, t_1 and t_2. As it can be interpreted, the SAD calculation is done for each of the displacements D1, D2 ..., D8 at the same time. The proper window forwarding process is achieved at module input, thus after the first clock cycle the subtraction between each pixel of the reference and displaced window is complete.

In order to prevent negative results from the subtractions, a magnitude test is performed previously. In the following clock cycles the SAD is completed by adding the differences and storing the result. The further steps are dedicated to determining the best match of all the calculated SAD, for all displacements and time shifts.

To minimize the loss of speed associated to the number of comparisons that are necessary to determine the match strength of each SAD, it is employed a tree like comparison mechanism, as shown in the example present in figure 4.7.

At each level, the SAD strength magnitudes are compared. To proceed to next levels, two registers are updated with the highest match strength and input position. This process is repeated for all stages, except in the *output* level where the input position register is used to trace back the information initially inputted on that referred position and produce the best match pair (DX, DY).

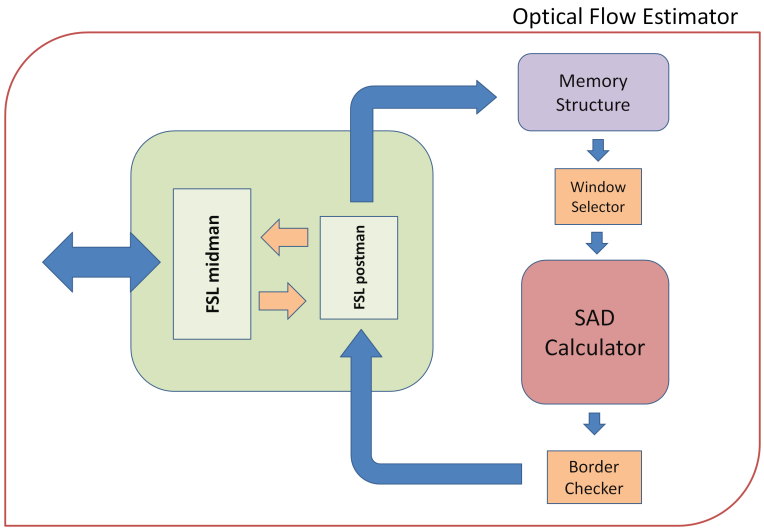


Figure 4.5: Optical Flow core elements.

4.5.2.2 FSL Subsystem

This subsystem is constituted by two modules which work in conjunction to receive and send data words from and to MicroBlaze, respectively.

The *FSL midman* module is responsible for interpreting and decoding commands received *via* FSL, into system operations. Due to FSL bus operation, this module works at a 50 MHz frequency . It is essentially a finite state machine (FSM), as represented in general terms in figure 4.8. The commands which are presented are as follows:

- HELLO: ping type command to access the core status;
- START: initiates OF calculation and immediately awaits for test sequence bytes;
- STOP: stops OF calculation;
- RESET: sets to default state the OF core and peripheral modules;

These commands are received using FSL protocol signalization and are followed by replies, upon action completion. Those responses consist on *HELLO THERE* and *DONE* bytes, sent using also FSL.

Upon OF core *START* command, the present module returns the computed flow which is directly received from the *FSL postman* module.

The *FSL postman* module function is to retrieve information from the surrounding modules, ensuring the correct data flow. As a matter of fact, it forwards to *FSL midman*, as stated, the best matches determined by the *SAD module*. This operation is done by storing three (although it can be a number up to four bytes of information) flow bytes into a 32 bit word before dispatching it for final send in the *FSL midman*, at a cost of three 100 MHz clock cycles. Each FSL communication from the OF core to MicroBlaze involves sending a package containg information about three

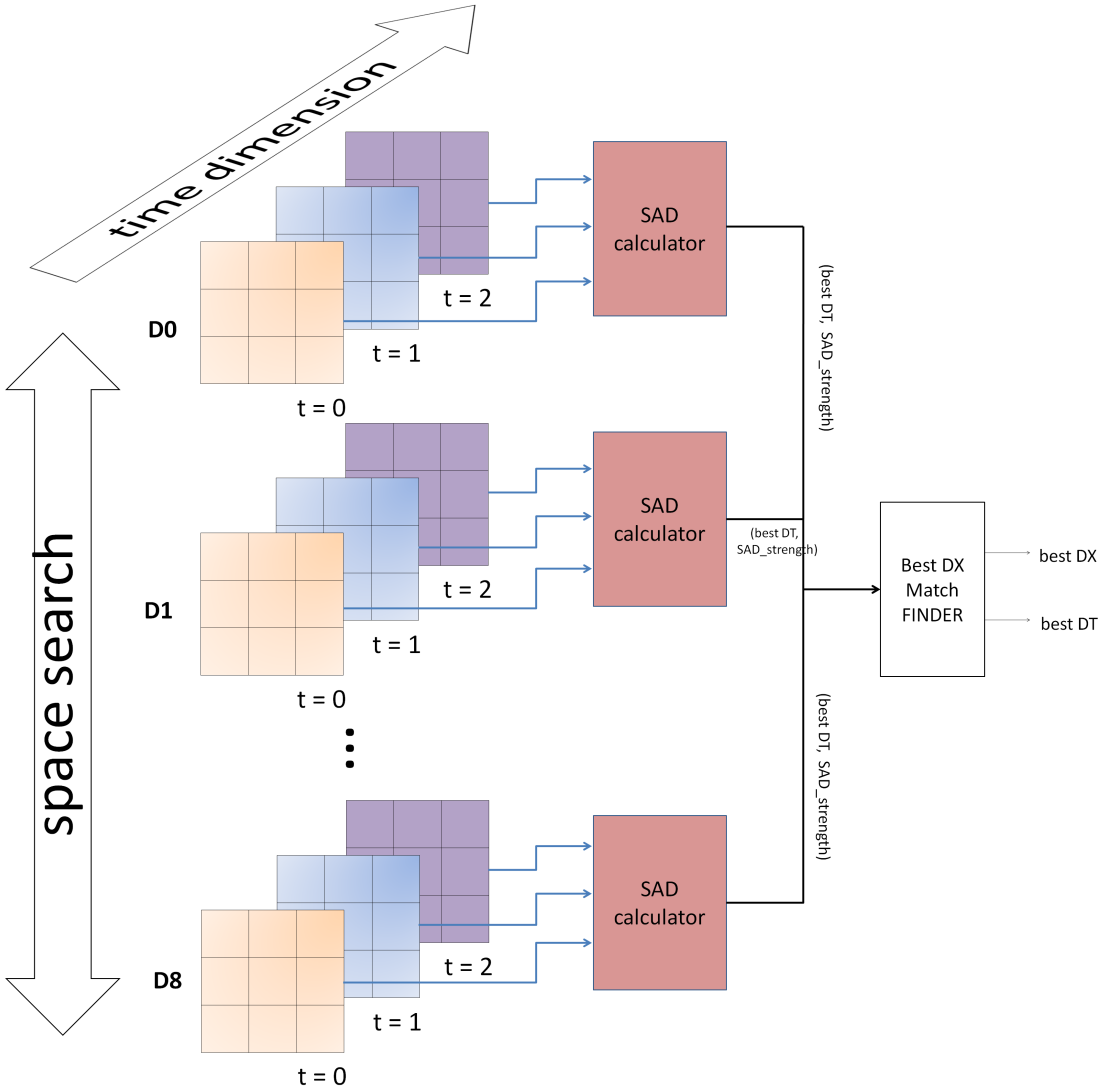


Figure 4.6: General view of the SAD parallel architecture.

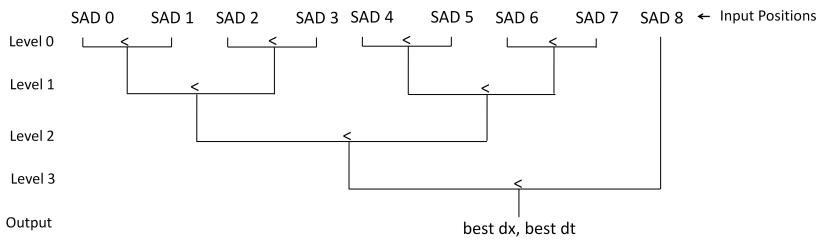


Figure 4.7: Compare mechanism used, to avoid compromising speed performance.

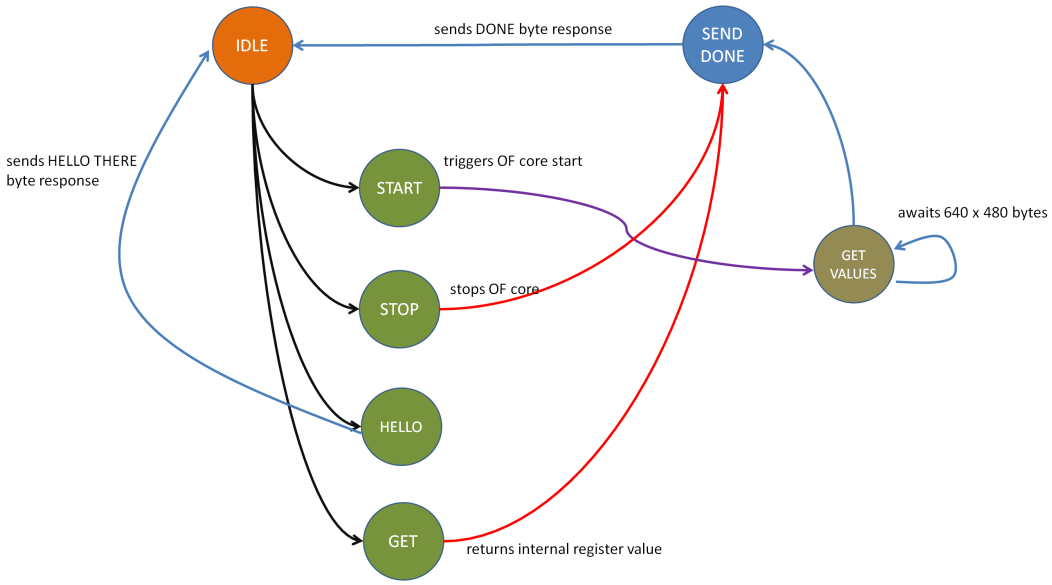


Figure 4.8: The FSM implemented by the *FSL midman*.

bytes. This waiting time has no consequence over system performance, as it a straightforward output line, i.e, the remaining system does not have to wait for the package sending to be finished.

4.5.2.3 Memory structure

Due to limitations of memory storage space, as discussed in section 4.2, it was necessary to implement a memory structure using an alternative architecture. Hence, it was decided to use a shift-register structure to store five lines of each frame t_0, t_1, t_2, singularly represented in figure 4.9. Since this requires a great amount of the FPGA resources, an alternative method was applied. Using the existing Xilinx templates, it was possible to implement a LUT (Look Up Table) based shift-register memory, thus reducing area occupation as some of the partially used slices had still free LUTs.

Each of these modules uses the camera signals (*vsync* and *href)* to sample the pixel information at the correct sample rate, using a synchronous 12,5 MHz with the camera system.

Each of the shift-registers, *samples* a 25 pixel window, which is then outputted to the *Window Selector* module except in the case of the memory associated to the reference window. In this case, it is only required to output a 9 pixel window. Moreover, it was decided to use for all frames, reference and candidates, the same memory size, although some space could be saved if this was optimized for the reference window. However, the complexity that such optimization was expected to introduce in system controllers, limited the flexibility and expansibility according to which architecture was designed.

The use of this type of memory also provides easier synchronisation between all system modules and, most importantly, enable the possibility to attain in parallel all the necessary pixels for calculating the optical flow for a given a pixel. Such technique allowed maximizing the calculation process, at a high cost of space though.

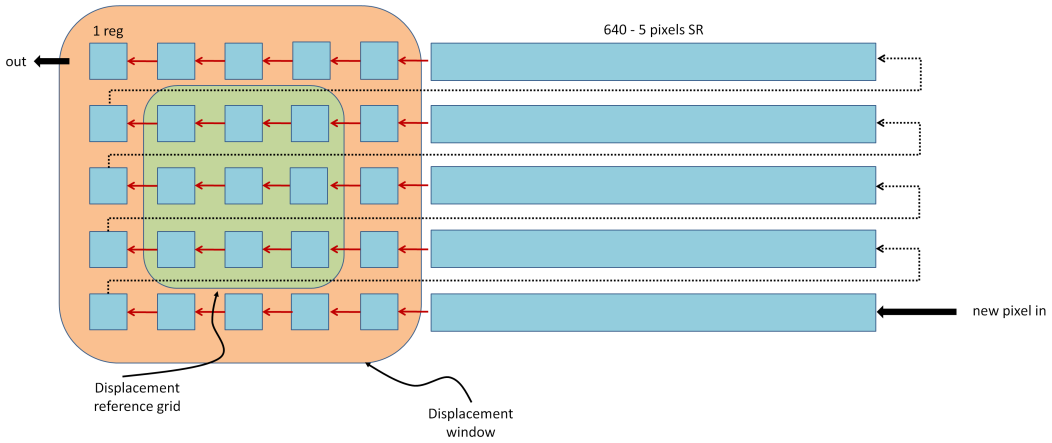


Figure 4.9: Shift-register type memory structure.

4.5.2.4 Additional Modules

Module *Border Checker* implements a watch mechanism that constantly evaluates the present pixel position in the frame. This is essential to prevent *SAD calculator* from computing erroneous results when the pixel being evaluated is less than 2 pixels away from the image border. This occurs since Camus grid needs a 1 pixel clearance to which is added another one due to the associated displacements window. Figure 4.10 illustrates the region which module *Border Checker* determines as valid result. As the system is set to work in a continuous *blind* mode, it is necessary to signal when the results are prone to be sent.

The remaining module, *Window Selector* represents an abstraction of the process of automatically generating the candidate windows, by virtually displacing them. In fact, as picture 4.11 illustrates, it samples different grids of pixels and assigns them to the input of the *SAD calculator* in order to allow parallel calculation. The sampling is made according to the search pattern which is indicated in the same figure.

4.5.3 Simulation Model

The simulation model which is used to debug and validate the final system reproduces the full OF core designed in the extent of this work. Besides evaluating the functionality of the estimator, it

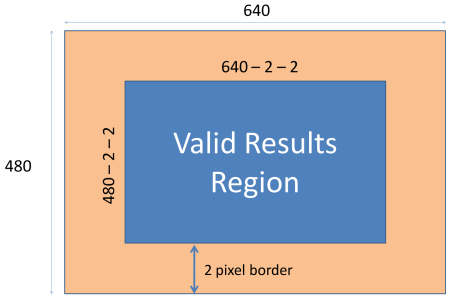


Figure 4.10: The region of valid results is pictured in the center.

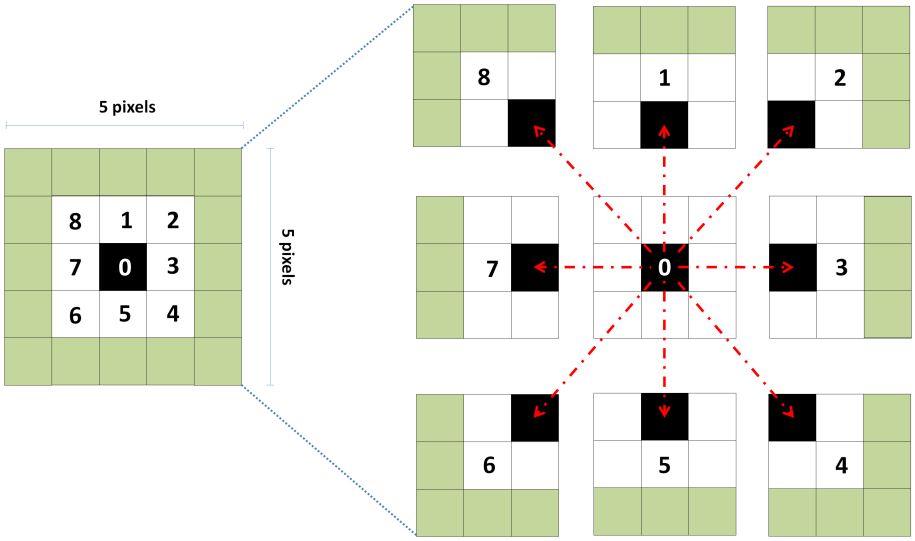


Figure 4.11: Method of generating the candidate windows used by *Window Selector* module.

attempts to prove the system capability to use the FSL to exchange information and commands with the MicroBlaze CPU.

As figure 4.12 presents, the testbench FSL emulator sends the test case by partitioning a video file in several sequential frames. The files are opened at the same time and sent parallely. The testbench then emulates the camera system with the corresponding pixel output rate. Each of the files refers to a virtual camera displaced one frame in time whose signal is input to each of the existing shift-registers.

Tasks simulating the FSL signalisation protocol send commands in order to initiate Optical Flow calculation and receive the incoming data saving it to an output file. Further C implemented processes use this information to create an intensity Optical Flow map.

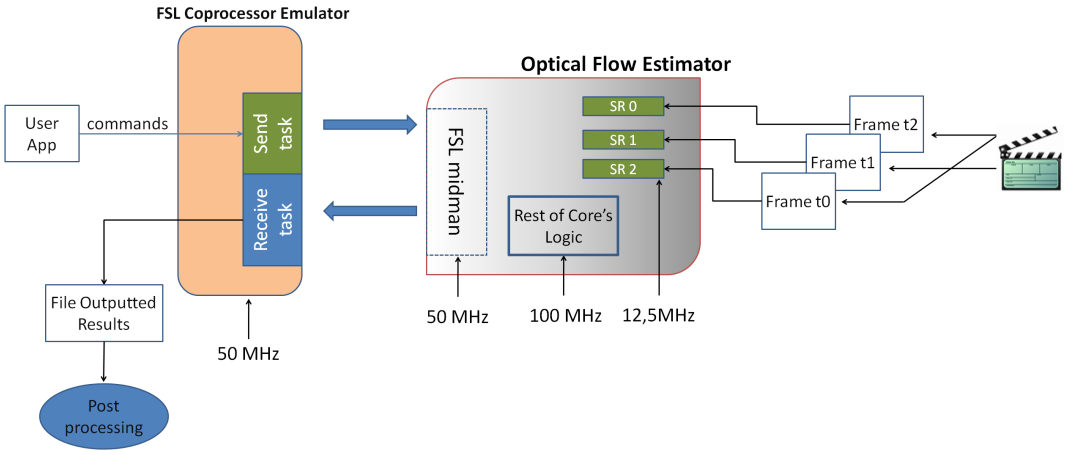


Figure 4.12: General view of the simulation model and testbench used to validate the final system.

4.5.4 Optical Flow Core Framework

The C framework which was developed to allow operation with the OF core implemented in the FPGA available logic, provides the user all the basic utilities. It is built in abstraction layers to allow clearer interpretation of the code and simplify its utilization.

One of the major disadvantages of the FSL bus lies on the fact that the CPU, which is using it to transfer data to and from a peripheral core, is on the communication datapath. This means that for the information to be extracted from the FSL FIFO's it is required the Microblaze to execute a read instruction (the same happens with writing). As a result, the flow of data depends on the processor (for more details revisit page 41). There are basically two types of functions: blocking and non-blocking. The first, on special occasions, such as starting a write command on a full communication channel, result in a hard lock of the CPU. The non-blocking instructions do not suffer from this problem and are, therefore, suitable to be implemented in multithreaded operative systems as Linux without compromising its stability. In this case, if a command is issued and the system is not ready, it simply returns an invalid status code to the user. This concept resembles the functioning of the OPB operation mode in which any command issues a read or write action over a memory address.

The FSL library found on Ubuntu takes the previous facts into account and only places available non-blocking functions. When a command is executed successfully, these functions return the desired data, if that is the case, along with a confirmation status code.

The library developed in the extent of this work implements the sending of commands which the *FSL midman*'s FSM recognizes and waits for the specific responses to each of those commands.

The lowest layer of abstraction sends the intended command byte code through the FSL and prints the respective information on the console for user feedback. In addition to this basic method, this framework provides a simple *blink–led*[3] function to feedback visually to the user that the hardware platform is working well. To clean any remaining information which may have been left unread from previous communications, there is a utility method that reads, a number of times equal to the FIFO's size, the same structure, printing any valid data found (with good status code).

The next level of methods is responsible for sending a command and waiting for a specific response from the peripheral core. These functions implement a mechanism that waits for incoming valid data that matches the indicated response byte code. To prevent them from blocking, only a fixed number of cycles are waited. When receiving the optical flow data bytes, it is used a counting mechanism to determine when a response must be waited.

The user layer comprehends a set of functions that implement the sending of all the available commands. Each of those methods returns the response byte code, thus allowing the user to easily control the status of the connection.

[3]LEDs blink repeatedly at a user-defined interval.

4.5.5 User Test Application

The user test application was developed having in mind that it would not be possible to implement a practical video system due to lack of memory. Therefore, it is configured to send via FSL the frames of the test sequences.

To guarantee a simple and user-friendly interface, it was created a straightforward menu in which the user configures all the necessary parameters to start Optical Flow calculation. All the vital information required to run the program is printed in the console for user feedback including error handling. Figure 4.13(a) presents a printscreen of the application running on Ubuntu Linux.

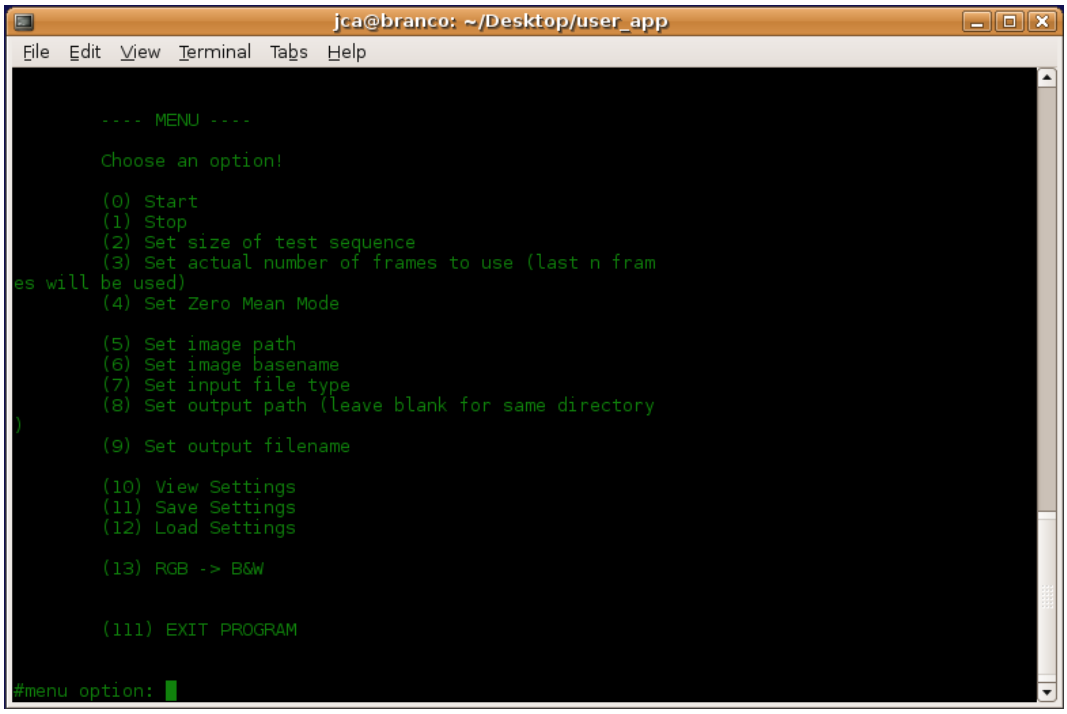
In order to avoid configuring the application each time it is started, a *save config* utility was created which enables the user to reload a project configuration. These configurations can be saved anywhere according to the inputted path in the menu, in a *.confs* file. Figure 4.13(b) shows the directory of the application and the subdirectories where some project configurations were previously saved and test sequences are kept. The paths to each of these directories are also configurable via the console menu.

The application runs on the assumption that the test sequence frames share the same name apart from a simple number identification, thus making easier the process of grabbing the different images. To simulate the process which relates to cameras sending their pixels into the OF core, it opens a number of frames equivalent to the depth of the algorithm's time search – in this case it matches 3. The bytes are sent in parallel to ensure synchronisation inside the core. Each time a trio of images reach their end, a new set is loaded, advancing the group a frame in time.

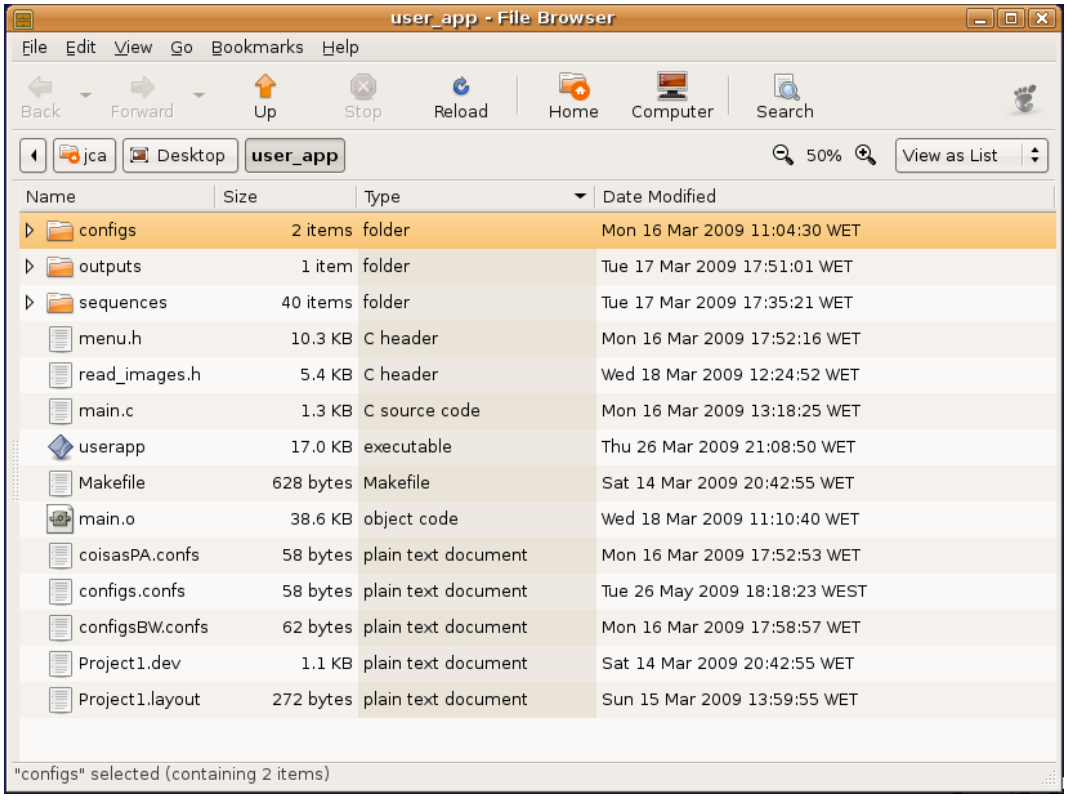
The practical utilization of this application showed to be necessary a smaller application to convert coloured images into greyscaled ones, which was also included in the application. This utility uses the configurations already set, fact that is promptly indicated to the user, through console hints.

It is also important to state that this application also provides a configured makefile to allow cross-compiling to the MicroBlaze system. Additionally, the code was designed to be easily altered to accommodate different image sizes or time searches, by altering the header's constants.

However, it becomes relevant to state that, due to the problems which were referenced throughout this chapter, the presented application was not used in conjunction with the final system.



(a) Application's console menu.



(b) Application's directory contents.

Figure 4.13: User Test Application.

Chapter 5

Experimental Results

5.1 Implementation Results

The implementation phase of this work has resulted in a successful architecture of an optical flow estimator. The decisions which were taken during this project helped to converge on a practical and functional architecture, making plain usage of the FPGA resources. The fact that it was achieved with a tight occupation result on the FPGA chip reveals how difficult it was to select a convenient algorithm and its architecture, see table 5.1 below.

The individual tests which were carried out revealed to be operational the modules which form the full system. The estimator was also tested for functional demonstration with successful results. However, these tests were not aimed at evaluating the estimator's performance, i.e. in terms of accuracy quantization, since its structure was picked out of an offline algorithm, used by McCane *et al* [22], in their publication on optical flow benchmarking.

The developed architecture is easy to extrapolate to other systems or hardware platforms with higher capacity. This is a significant feature since the implementation was strongly conditioned by the available technology and can therefore be object of future work. The resulting estimator was implemented, with the following critical setup:

- Search window of 3×3 pixels;
- Frame history $S = 2$;
- Embedded memory structure of *shif-register type* of size $5 \times 640 \times (S+1)$;
- Test frames are directly inputted to the SR-memory;
- Result visualisation is possible only after offline processing.

Some of these features introduce slight variations to the practical objectives, which were initially established. However, they are considered to not affect the purpose of this thesis as they are viewed as enough to demonstrate its functionality. As a matter of fact, the search window size and frame history refer only to the algorithm performance and accuracy and could be easily augmented since the HDL description is easy to expand. The implemented memory structure could

be designed as external memory to accommodate the required increase of storage space. Hence, a proper subsystem could be designed to create a frame history in the referred external memory, which could then be sent to SR-memory. It is also important to remark that the system's optical flow estimator performs well, independently of the form through which the test cases are inputted. Finally, the last feature affects only viewer's empirical analysis, thus circuit operation can be easily set to output an optical flow intensity map directly to a VGA monitor. Consequently, system functionality remains intact along with the essential desired objectives.

The implementation final phase consisted on programming the FPGA chip of the SUZAKU board in order to run a developed application in a Linux environment, running on the MicroBlaze. However, due to an unidentified problem this final step could not be accomplished. This issue was confined to the step of configuring the XPS project with the FSL middleman and the OF core inner modules, with different clock frequencies. Unpredictably the system does not work under these conditions. Tests were made where the *FSL midman* was connected to a simple counter module, each working with distinct clock rates. After much time spent experimenting alternative configurations, this problem was not over passed. Conversely, the system responded well using only the *FSL midman*, returning the expected responses.

As a result of the subsequent implementation delay, it was necessary to follow a different path in terms of project flow. Therefore, it was decided to create a truthful simulation model capable of demonstrating the system's functionality, i.e, of the OF core, FSL communication and of the C framework. Due to the imminent lack of time, only the OF core and FSL bus were validated under that simulation model.

The OF core implementation produced the following results in terms of FPGA resources occupation:

Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slices	7202	8672	83%
Number of Slice Flip Flops	6351	17344	36%
Number of 4 input LUTs	11897	17344	68%
Number of bonded IOBs	72	190	37%
Number of GCLKs	3	24	12%

Table 5.1: Summary of the OF core occupation, without the MicroBlaze.

Most of the slice occupation is due to the implementation of the SR-memory structure, which could not be implemented in any other resources as they are not enough. The LUT implementation allowed to further optimize the utilization of slices. Each of these modules uses about 1300 slices and 2100 LUTs.

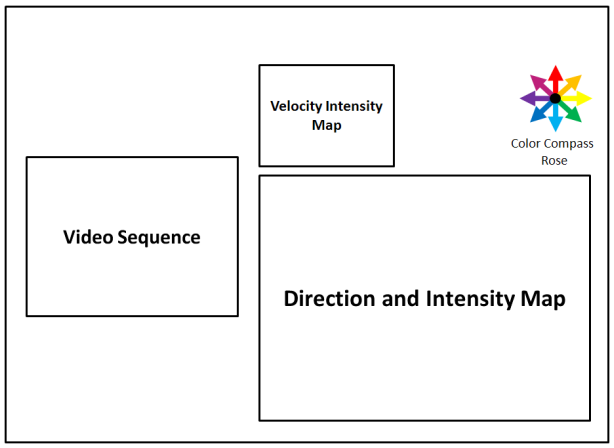


Figure 5.1: Template used to present the attained Optical Flow maps.

Analysing the previous table with 4.3 it is possible to conclude that a working version of this system, i.e., the OF core in conjunction with the MicroBlaze, does not fit within the FPGA's resources. This occurs even though the minimum project settings were used. As a consequence, the major conclusion to be drawn from these facts is that the available hardware platform is not suitable for the implementation of this system even though its design has been validated.

5.2 Optical Flow Maps

The following subsections present a variety of patterns which are the result of the developed Optical Flow estimator architecture, designed in the extent of this dissertation. These were attained applying the previously referred simulation model, described in section 4.5.3. The test cases used attempt to avail the estimator's capability to correctly characterize movement in a scene. The inputted sequences test the full frame resolution and multiple known OF patterns.

The existing test databases do not offer standardized frame sequences with resolution of 640×480 pixels. In the attempt to over pass this issue it was decided to use custom test cases which were first filmed with simple web cams. The quality of the video proved to be highly affected with noise. In order to create relative noise-free input sequences, it was decided to record live action from a computer based flight simulator. These later sequences proved to be very realistic and to fulfil the testing requirements.

The results which are presented next, namely the intensity and directions maps, are the result of offline post-processing tasks. These tasks read the output OF file and accordingly to a straightforward metric calculate the associated pixel movement in terms of intensity and direction. These two parameters are represented individually and simultaneously in the intensities and directions maps, respectively. In the first, the brighter spots indicate faster movement, and, in the latter, the colours are associated to each of the nine possible directions. Figure 5.1 presents the referred colour compass rose with eight directions plus the static one (black). The colours which are pictured indicate the tones which are associated with highest velocity, i.e., the lower the speed the

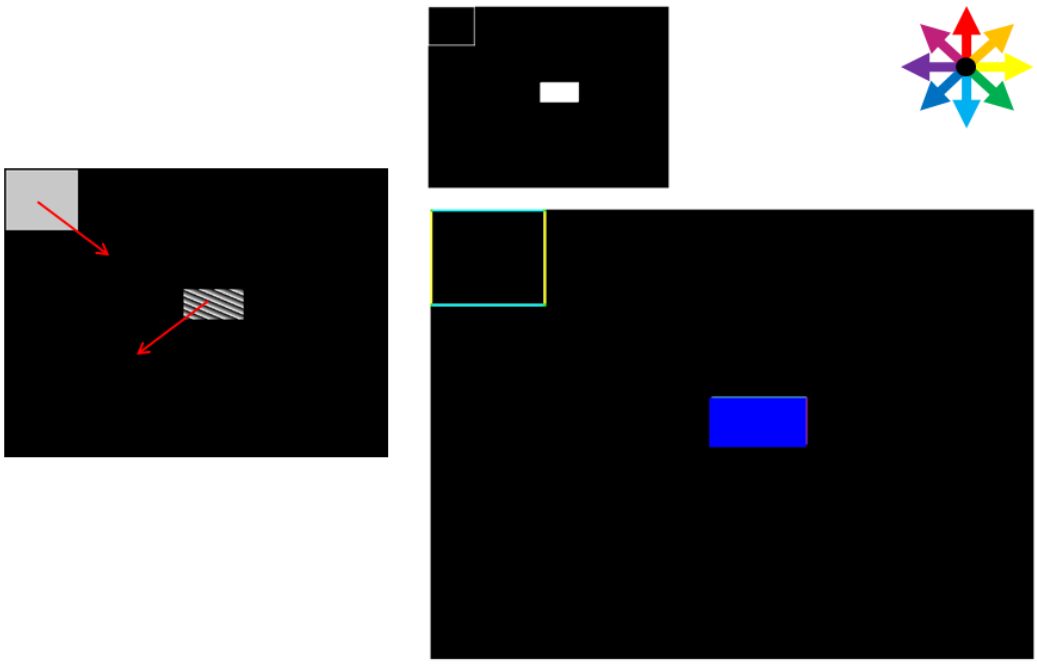


Figure 5.2: Simple validation test with computer generated test case.

dimmer is the colour.

The test case and the scene movement are indicated on the left frame identified as the *Video Sequence*. The intensity maps are pictured above the central result area labelled as *Direction and Intensity Map*.

The following test cases and respective results can also be found at [9].

5.2.1 Validation Test

This is a simple test case in which a computer generated sequence is used to determine if the estimator is capable of outputting coherent results.

The test is comprised of three main features: two rectangular objects with different movements, textures and a static background. These features were conceived to evaluate the movement detected for each object as well as to understand how it would perceive opposing directions and, finally, note which implications does object texture has in the employed OF algorithm.

The results, presented in figure 5.2 indicate that movement is determined correctly for each object, specifically in the borders of the left rectangle and overall for the right one. This occurs due to fact that the first has a flat texture, thus the algorithm cannot perceive that movement has occurred and therefore *prefers* to attribute the static direction. The green colour is perceived around the object, in a 1 pixel frame, and largely in its vertex that stands in the direction movement takes place. The cyan and yellow appear due to a preference scheme which is inherent to the developed architecture. In fact, the system is configured to pick out cardinal directions when match strength is not convincingly pointing to an intercardinal direction. The intensity map is as expected.

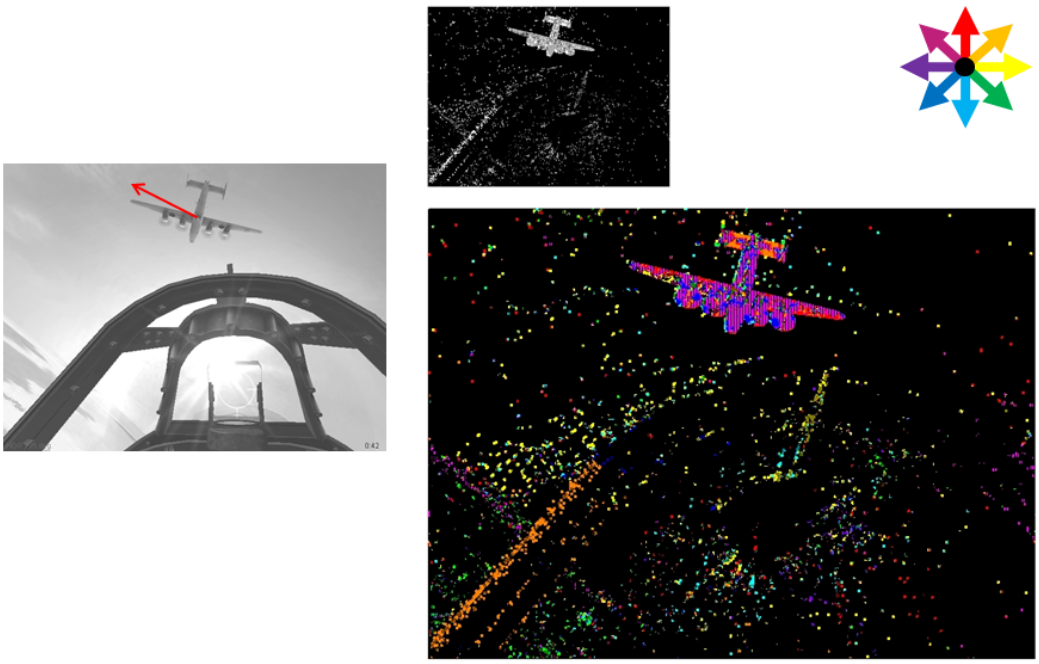


Figure 5.3: Characterization of single object motion under good detection conditions.

5.2.2 Single Object Detection

This test was designed to evaluate how well would a single object in motion, in relative good detection conditions, be detected and characterized by the OF estimator. Figure 5.3 indicates that under such conditions the calculated flow is extremely accurate, thus producing a clear map in which the object stands out.

5.2.3 Rotation of the Viewer's Perspective

This is a typical OF pattern to which multiple directions are associated. The rotation of the viewer's perspective on its own axis leads to a counter wise movement of the scene background. As it is pictured in figure 5.4 the right side of the directions map is pictured in tones of red and violet (indicating an upwards movement) and the left side is essentially a green and cyan area (downwards). The map is not very clear in some areas mainly due to absence of more well defined reference points in the sky and due to the complexity of the movement itself.

5.2.4 Low Contrast Environment

In order to access the algorithm's performance under low contrasting environments it was decided to use this night scene which reveals very impoverished contrast information in most of the scene. The result presented by figure 5.5 indicates that it is possible to perceive the object shape and movement even under such conditions.

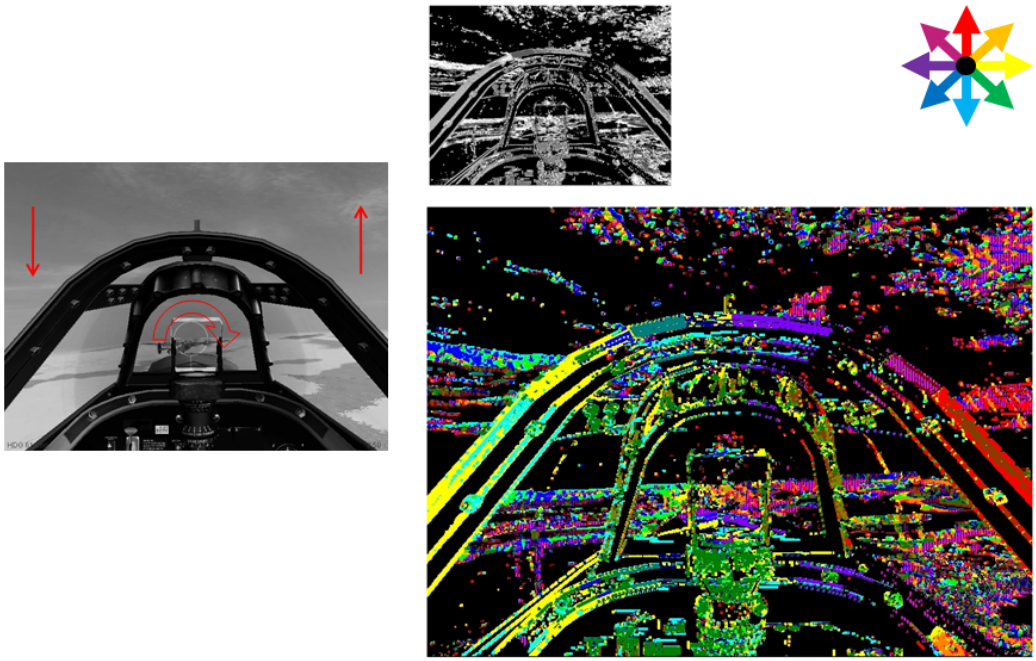


Figure 5.4: Rotation of viewer's perspective.

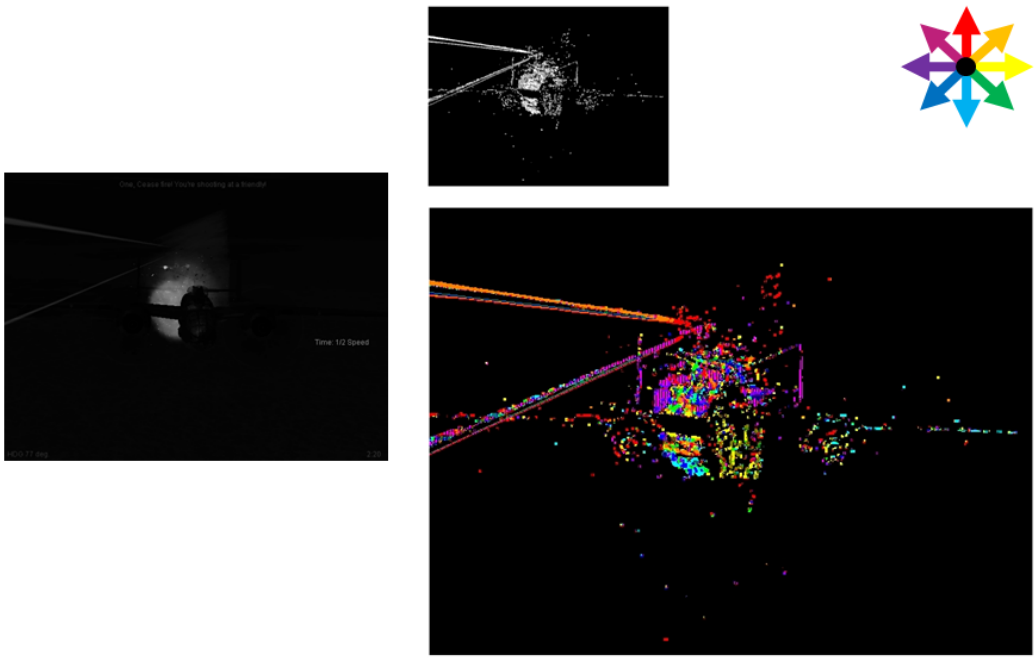


Figure 5.5: Object with low contrast with background.

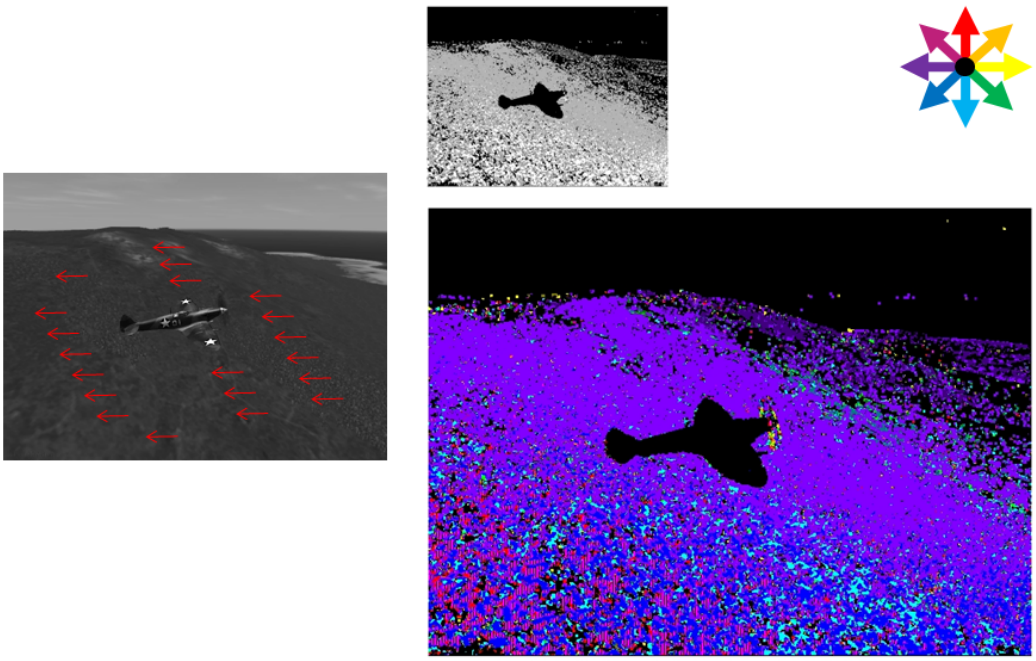


Figure 5.6: Detection of background movement.

5.2.5 Simple Movement of the Background

This test evaluates the scene flow in an opposite fashion to test of subsection 5.2.2. In fact, this test puts a fixed camera on the main object and watches the background move in a continuous direction. In this case, the estimator is very well succeeded, removing the central object from the output (as static) and accurately indicating the movement direction of the background landscape. In addition to this, the sky remains static as expected.

5.2.6 Divergent Movements

The scene applied in this test introduces some complex analysis features which the estimator must handle. In the first place there two objects with diverging movements. The one found on the lower side of the image (bomber) is turning in the direction of the viewer while the smaller (fighter), at the centre, is turning over its own axis. Additionally, the camera is centred on the fighter while the bomber is moving slightly faster than the background, which is also translating (as the camera is centred on the fighter). Since the bomber is moving faster than the background its direction will be opposite to the background's one.

Figure 5.7 reveals that the estimator is able to perform satisfactorily in this case. The directions are correctly represented. Both background flow and objects are well calculated, and, somewhat surprisingly, the estimator determines the flow resultant from the fighter roll.

It is also of notice that, due to the intricacy of the relative velocities, it is not possible to stand out any object in the intensity map – as expected though.

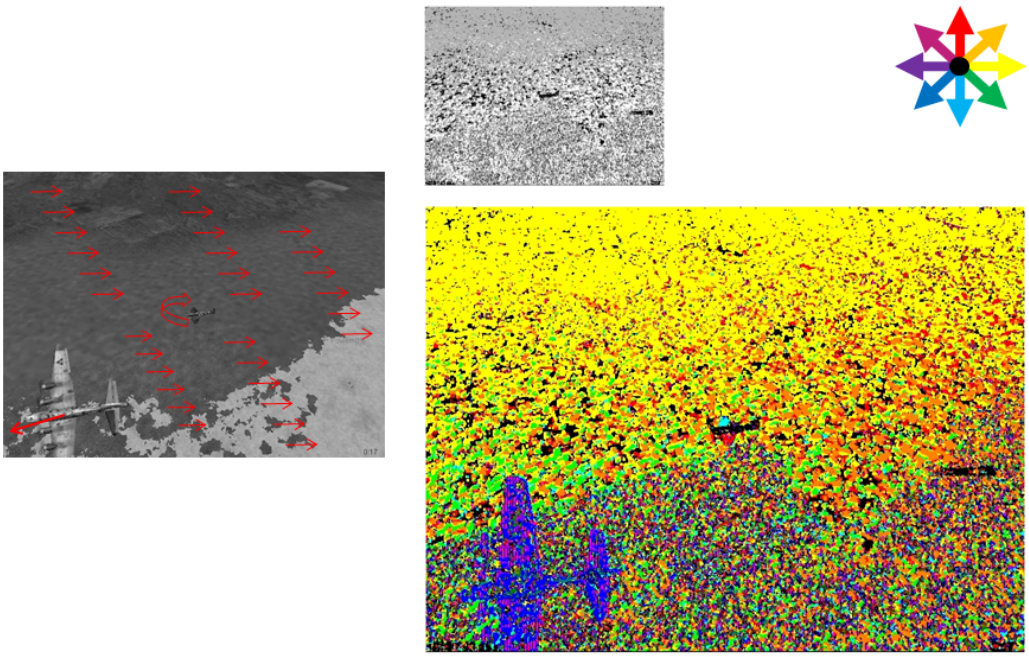


Figure 5.7: Complex test scene with divergent movements and confusing velocity patterns.

5.2.7 Scene Expansion

The present test is aimed at testing a known OF pattern resulting from a forward motion, when reference objects are present nearby. The result pictured in figure 5.8 shows two relevant features. The first relates to the fact that information derived from the scene expansion allows the estimator to be able to perceive the structure of the object that is closing in. The other is associated to the fact that it is possible to distinguish two main plains: the ground whose movement characterization is very fuzzy but very intense and the sky which is pictured static. This last feature is relevant for navigation, for instance, to provide the system with a *sense of proximity*.

5.2.8 Performance under Scene Complexity

The scene that constitutes this test is characterized for including multiple objects moving with various velocities and with different directions. There are also multiple contrast variations for each object together with slow background movement as the camera follows the turn of the object in the centre.

Figure 5.9 presents the calculated OF. It indicates that it was possible to accurately calculate for each object the associated movement. The background flow is also visibly correct, although somewhat faded due to the slow motion of that plane and lack of texture variations. As a result, the estimator determines better flow in the terrain borders. The intensity map clearly identifies the various objects and the terrain movement.

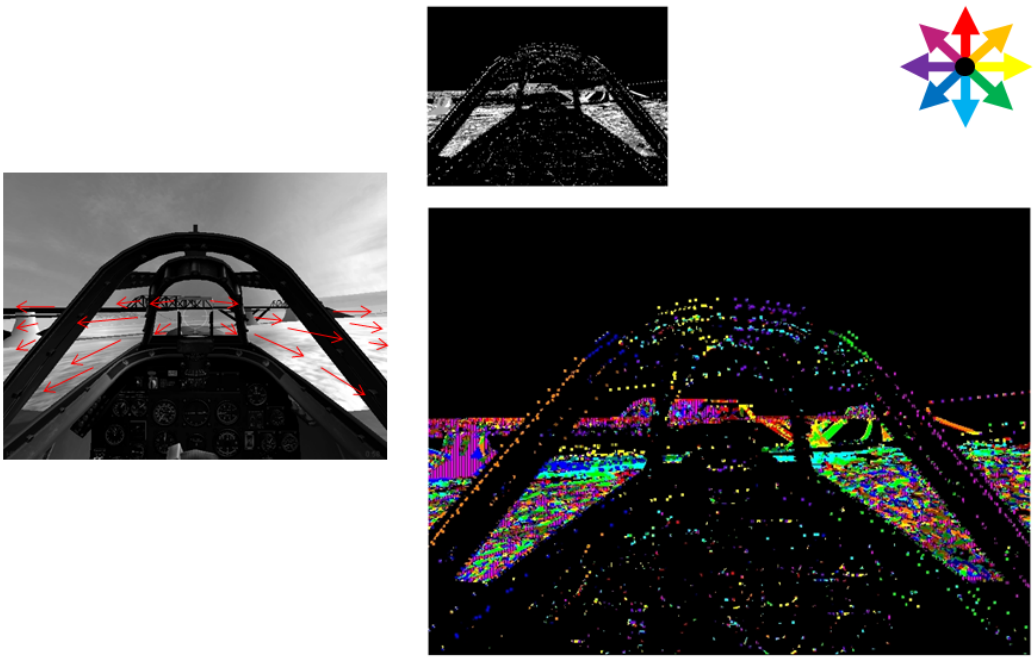


Figure 5.8: Test case which evidences an expansion pattern.

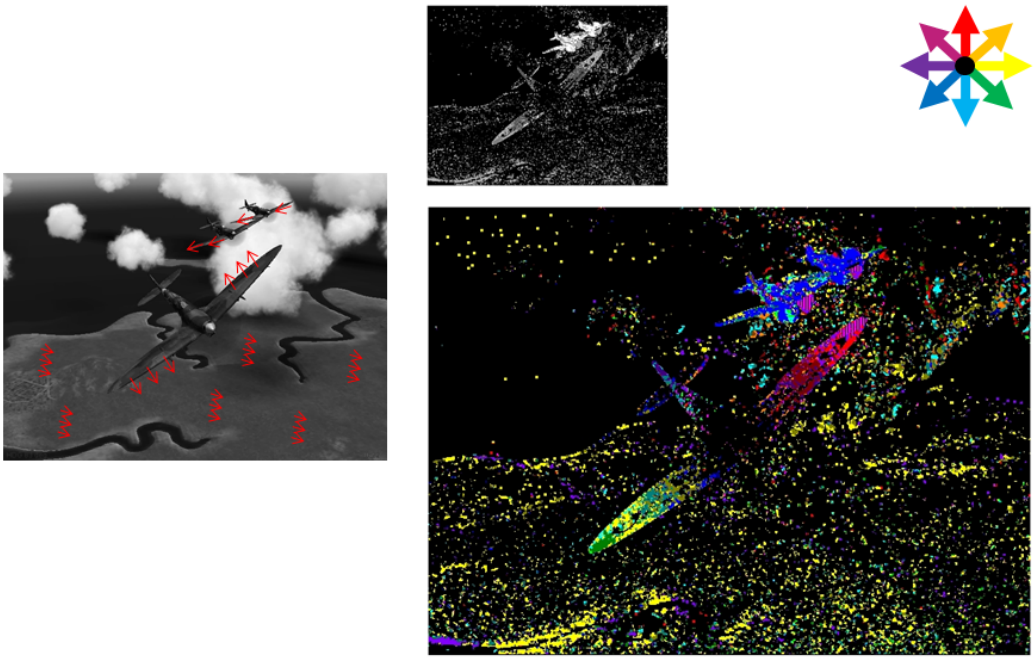


Figure 5.9: Complex test which includes multiple objects, directions and speeds altogether with varying detection conditions and background movement.

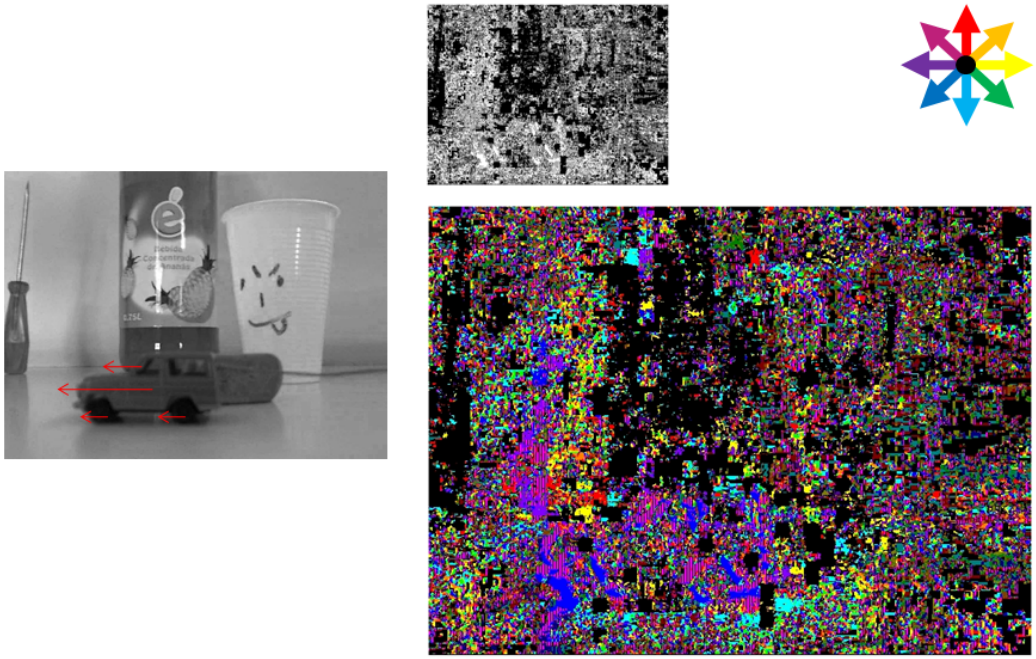


Figure 5.10: The OF map resulting from a video input suffering from severe noise.

5.2.9 Noisy Environments

The test sequence applied for this test was attained from a simple webcam. In the video there are some of the effects which are known to degrade performance of correlation based algorithms such as: noise, mosaicing and also some subsampling effects.

The resulting OF map is presented by figure 5.10. The quality of the calculated flow proves to be extremely degraded due to the presence of the previously referred factors. In order to attempt to solve this problem, it was decided to apply a simple hardware threshold level in order to *strenghten the confidence* of the direction. As a result, only match strengths above a defined level are prone to be selected as a different direction other than the static one (pictured in black colour). Applying this technique it was possible to achieve considerable improvements as it is perceived from the results pictured in figure 5.11. The moving object stands out in the flow maps as most of the scene is understood as noise and thus drawn in black.

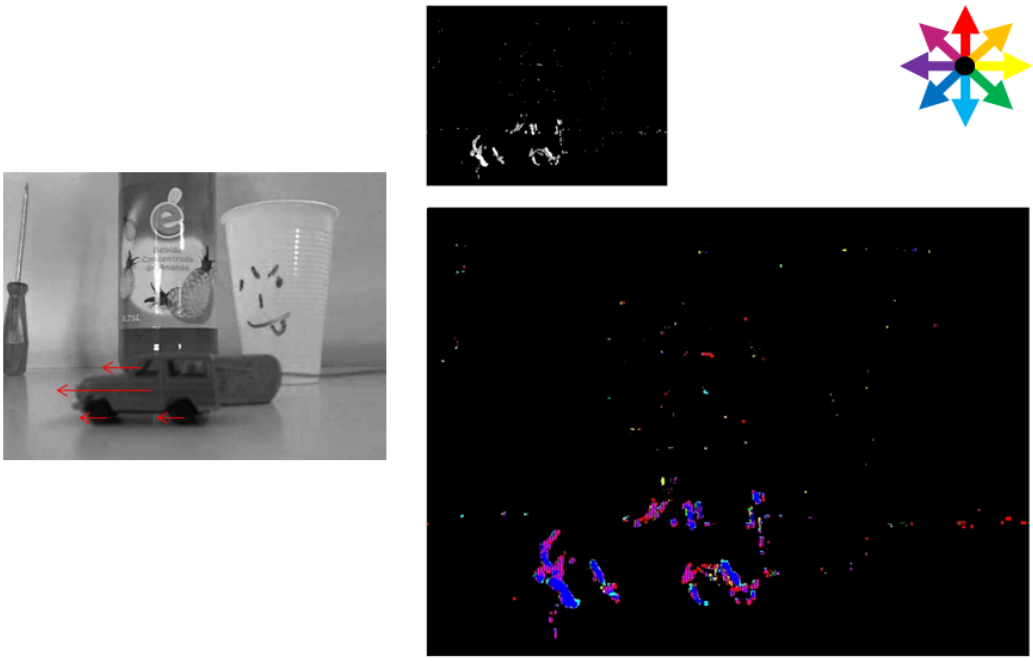


Figure 5.11: Result after applying simple noise filter on same test case as 5.10.

Chapter 6

Conclusions and Future Work

In the course of this work, the issue of Optical Flow calculation was addressed in the context of recreating Visual Perception in artificial systems. The concept of Optical Flow was discussed from the (neuro)psychology and engineering points of view. Different algorithms and optimizations were studied in order to design a functional real-time architecture of an optical flow estimator.

The practical result of this thesis was set by the design and implementation of an Optical Flow estimator, based on Camus correlation algorithm and McCane *et al* software tool.

This system is capable of processing 640×480 frames in real-time (40 frames/s), using a search window of 3×3 pixels and $S = 2$ frames, with an expectable maximum error of about 10^{o} in direction. This system is designed to operate in conjunction with an embedded MicroBlaze CPU, with a pre-built Linux Operative System, communicating *via* FSL bus. The OF core is capable of working at a rate up to 100 MHz and returning data to a user application at a rate up to 50 MHz. The OF core is handled using a user-friendly C framework. This system was implemented on a SUZAKU-S SZ130-U00 hardware platform, which integrates a Xilinx XC3S1200E FPGA chip. The embedded CPU works at a frequency of approximately 50 MHz.

The selection and design of the final architecture followed a rigorous and methodical research which allowed coming up with a scalable system making the best use of the FPGA resources.

The initial objectives of this thesis are considered to have been fulfilled by the outcome of the implementation phase. Although it was not possible to implement the final system in the available technology, it can be demonstrated that the project guarantees its functionality according to the parameters that were initially set.

The main contributions of this thesis in what concerns the subject of Optical Flow calculation lie on the following items:

1. shows that it is possible to establish an efficient partition between user software and dedicated hardware to compute complex data such as the Optical Flow, sufficiently accurate to allow further post-processing.

2. presents a distinct architecture of a Camus algorithm capable of processing 640×480 frames in real-time.

3. places available a simple and open-source C framework to handle the OF core, which can be used for complex user applications based on Optical Flow data.

4. creates a document which presents a starting point to a beginning research, by introducing a theoretical background of the subject, summarizing the form of operation of the most prominent optical flow algorithms, addressing the side issues which affect the calculation and, finally, delivering an expandable architecture of a functional Camus estimator.

6.1 Future Work

Most of the future developments that might be suggested are based on the technological limitations which were imposed due to the available hardware platform. On the other hand, the information retrieved during the research phase has revealed many possible uses of Optical Flow data.

Further implementations may, as result, use the present work to expand the designed Camus estimator to increase its accuracy, by using external memories to augment the depth of time search. Besides this, the estimator can be expanded to pursue better results in real test cases, by employing wider than 3×3 pixels reference areas and by extending spatial search – though this last is to avoid due to the quadratic increase in complexity. As the study presented in section 3.5.2, there is space for improvement in terms of algorithm performance, although it essentially depends on the available logic space.

This system can be also improved to implement a real-time video system calculating optical flow. In order to achieve this, it would be necessary to use an appropriate CMOS camera interface. Afterwards, it would be required to internally link the FPGA pins to the OF core's SR-memories, as this system architecture previews.

Optical Flow data may be used to compute more complex perceptions out of video frames such as obstacle recognition, time-to-contact, and navigation in general. Any of these applications represents a possible post-processing application which could be designed out of the existing framework.

An interesting tweak that can be suggested to this type of systems is related to the characteristic noise found in the frames coming from video devices. The key idea is to divide the vision area in major blocks and for each one keep a simple measure of the average flow. These measures can then be back forwarded in such way that the next calculated flow will not register much of images natural flow noise. By employing this technique, it is possible to *equip* the system with a virtual awareness of the scene's background flow and be therefore more prone to detect flow variations in such noisy environments. This method could prove to be adapted to security systems, for instance, as they often are required to watch over static scenes for long time.

References

[1] Otago University Graphics and Vision Research Laboratory, June 2009. http://www.cs.ru.nl/~ths/rt2/col/h9/9gebiedENG.html.

[2] Herman Gomes, January 2009. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/GOMES1/marr.html.

[3] Zhaoyi Wei, Dah-Jye Lee, Brent Nelson, and Michael Martineau. A Fast and Accurate Tensor-based Optical Flow Algorithm Implemented in FPGA. In *WACV '07: Proceedings of the Eighth IEEE Workshop on Applications of Computer Vision*, page 18, Washington, DC, USA, 2007. IEEE Computer Society.

[4] T. Camus. Real-time quantized optical flow. In *Proc. Computer Architectures for Machine Perception CAMP '95*, pages 126–131, 18–20 Sept. 1995.

[5] José L. Martín, Aitzol Zuloaga, Carlos Cuadrado, Jesús Láizaro, and Unai Bidarte. Hardware implementation of optical flow constraint equation using FPGAs. *Comput. Vis. Image Underst.*, 98(3):462–490, 2005.

[6] P. C. Arribas and F.M.H Maciá. FPGA Implementation of Camus Correlation Optical Flow Algorithm for Real-Time Images. *14th Int. Conf. Vision Interface*, pages 32–38, 2001.

[7] Algorithms Hongche Liu, Hongche Liu, Tsai hong Hong, and Martin Herman. Accuracy vs. Efficiency Trade-offs in Optical Flow. In *Computer Vision and Image Understanding*, pages 271–286. Academic Press, 1996.

[8] Inc © 2007 Atmark Techno, June 2009. http://download.atmark-techno.com/suzaku-starter-kit/.

[9] João P. Santos. Implementation OF Algorithms in FPGA Platforms with Embedded CPU, July 2009. http://sites.google.com/site/ofinfpgawithembeddedcpu/.

[10] R. L. (Richard Langton) Gregory. *Eye and brain : the psychology of seeing*. World university library. Weidenfeld and Nicolson, 2nd edition, 1972.

[11] James J. Gibson. *The Ecological Approach to Visual Perception*. Lawrence Erlbaum Associates, 1979.

[12] David Marr. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. Henry Holt & Company, June 1982.

[13] Jane Willson Paul Rookes. *Perception: Theory, Development and Organisation*. Routledge, 2000.

[14] Mark A. Georgeson Vicki Bruce, Patrick R. Green. *Visual perception: physiology, psychology, & ecology*. Psychology Press, 4th edition, 2003.

[15] Berthold K. P. Horn and Brian G. Schunck. Determining Optical Flow, 1981.

[16] Bruce D. Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. pages 674–679, 1981.

[17] Bruce D. Lucas. *Generalized Image Matching by the Method of Differences*. PhD thesis, Robotics Institute, Carnegie Mellon University, July 1984.

[18] Andrew Bainbridge-Smith and Richard G. Lane. Determining optical flow using a differential method. *Image Vision Comput.*, 15(1):11–22, 1997.

[19] Jonathan W. Brandt. Improved Accuracy in Gradient-Based Optical Flow Estimation. *Int. J. Comput. Vision*, 25(1):5–22, 1997.

[20] M. V. Correia. *Técnicas Computacionais na Percepção Visual do Movimento*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, Departamento de Engenharia Electrotécnica e de Computadores, Septembre 2001.

[21] Zhaoyi Wei, Michael Martineau, Dah-Jye Lee, and M. Martineau. A Fast and Accurate Tensor-based Optical Flow Algorithm Implemented in FPGA. In *Applications of Computer Vision, 2007. WACV '07. IEEE Workshop on*, pages 18–18, Feb. 2007.

[22] B. McCane, K. Novins, D. Crannitch, and B. Galvin. On benchmarking optical flow. *Comput. Vis. Image Underst.*, 84(1):126–143, 2001.

[23] J. L. Barron, D. J. Fleet, and S. S. Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12:43–77, 1994.

[24] Brendan McCane. Graphics and Vision Research Laboratory, June 2009.

[25] Ted Camus. Calculating Time-to-Contact Using Real-Time Quantized Optical Flow. In *National Institute of Standards and Technology NISTIR 5609*, 1995.

[26] Inc © 2007 Atmark Techno, June 2009. http://www.atmark-techno.com/en/products/suzaku/dev.