# Pyramidal architecture for stereo vision and motion estimation in real-time FPGA-based devices

Matteo Tomasi

Departamento de Arquitectura y Tecnología de Computadores

Universidad de Granada

A thesis submitted for the degree of

*PhilosophiæDoctor (PhD) in Computer Science*

May 2010

Editor: Editorial de la Universidad de Granada
Autor: Matteo Tomasi
D.L.: GR 3189-2010
ISBN: 978-84-693-4303-6

# Declaration

Dr. Eduardo Ros Vidal, Catedrático de Universidad, y Dr. Javier Díaz Alonso, Profesor Ayudante Doctor de Universidad, ambos del Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada,

CERTIFICAN:

Que la memoria titulada ”Pyramidal architecture for stereo vision and motion estimation in real-time FPGA-based devices”, ha sido realizada por D. Matteo Tomasi bajo nuestra dirección en el Departamento de Arquitectura y Tecnología de Computadores de la Universidad de Granada para optar al grado de Doctor por la Universidad de Granada.

Granada, 14 de Mayo 2010

Fdo. Eduardo Ros Vidal                    Fdo. Javier Díaz Alonso

# Abstract

In this work, an exhaustive study for a low level vision engine is presented. Our hypothesis is based on recent development and advantages of FPGA devices (reduced power consumption, high processing capabilities) and the performance of new HDLs and synthesis tools. Thus we address with these powerful means the novel target of a low level vision engine on the same chip. The study aims to demonstrate that is possible the integration of multiple complex algorithms thanks to a proper adaptation and good design techniques. In particular we focus our architecture to a fine grain pipeline in opposition to the multi-core approach largely used in last architectures. Our approach benefits the power consumption and the size of the final implementation providing a very competitive system useful for industrial, robotic and research fields. For the first time is afforded a multi-scale and a multi-orientation optical flow and stereo on FPGA. The iterative nature of this approach degrades the processing speed but achieves an important accuracy and significantly enhances the working range. Final results in synthetic and real sequences demonstrate the competitive performance of the presented system.

To my family

# Acknowledgements

I would like to acknowledge all people that with more or less contribution helped me in these years. Without their presence my work it would be impossible. I would remember all of them but in few pages it is a very difficult task. For this reason I express a first enormous ”THANKS” to all persons that in my opinion can be considered as a little brick for the building of this work. However it is important for me to mention some of them. In a special way I would like to acknowledge my thesis directors Edu and Javi for their charisma. They give me the possibility of a very fruitful research and they attend to me with patience. Thanks also to all the ATC department in special way to the ”CIE” group. I would like to acknowledge all the DRIVSCO members for their example and great scientific contribution.

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

*Historia magistra vitae est.* Exploring the ancient theories, is possible to find in the old Greek philosophy some rudimental definitions or concepts of human vision interpretation. Aristotle in his book "On Sense and the Sensible" affirms that *"Of the two last mentioned, seeing, regarded as a supply for the primary wants of life, and in its direct effects, is the superior sense; ... The faculty of seeing, thanks to the fact that all bodies are coloured, brings tidings of multitudes of distinctive qualities of all sorts; whence it is through this sense especially that we perceive the common sensibles, viz. figure, magnitude, motion, number ..."*. After these first preliminary steps, a lot of relevant philosophers and scientists have contributed to build many theories for the human vision system. This great and long history demonstrates how much important is for our life vision understanding. In day life, for example in a sport environment, all kind of trainer agrees with colleagues thinking that a good player has to be rapid in the vision of the game and in the interpretation of the play before the adversary. If we move to a more dramatic environment such as a driving scenario, a person driving in bad environmental or physical conditions need a very fast reaction to a sudden presence of an object in his way. The reaction will be faster if the processing capabilities need less time to understand the danger. Both cases depend strongly from the velocity of vision system. In a hunting scenario, also vision is critical and the major advantage resource of many natural hunters (such as eagles). With three simple citations we remark importance of vision, velocity and size. In this work we want to collect these three challenges in a single one: a fast and small vision engine. The realization of this ambitious task will take advantage of recent developments in technology and research of these last years and it is possible only with an accurate and intelligent use of previous contributions. Despite the great advance of computational processing in general purpose machines we focus our attention on small embedded systems that can afford at the same time our three aims (low power consumption, high performance, and physical size). As we can see from everyday experience, our life is more and more rounded of small electronic devices powered with powerful and fast chips. In this work, we contribute to this research field adopting Field Programmable Gate Array (FPGA) devices and defining novel architectures of image processing.

## 1.1 Vision Framework

One of most complex and perfect example of vision systems is the human sight. As reported by Kandel in his book [6] *"Studies of artificial intelligence and of pattern recognition by computers have shown that the brain recognizes form, motion, depth, and color using strategies that no computer can achieve. Simply to look out into the world and recognize a face or enjoy a landscape requires an immense computational achievement more difficult than that required for solving logic problems or playing chess."* Beginning with the firsts neural and biological studies, scientists have been attracted from this wonderful system that is the human "machine". Observing and understanding the nature mechanisms, researchers try to reproduce its functioning. Human vision system is not an exception. Following this philosophy has been born photography and video cameras and now a lot of vision algorithms are inspired by nature. At the moment the human visual system is not completely understood. We know that the complex process of vision and scene interpretation can be divided in different level of processing. As displayed in Fig. 1.1 and taking inspiration from neuroscience, a vision system can be divided into low-level, middle-level and high-level [7]. With respect with other processing parts of the brain we know that visual cortex occupy a significant area. For example in the macaque neocortex the 50% is dedicated to process the visual information while only the 11% is somatosensory cortex and 3% is auditory cortex. Furthermore as stated in [6] we know that among the different parts of visual cortex the biggest ones are the V1 and V2 (low level vision) that occupy over 1100 mm$^2$, while one of the smallest is the MT (middle temporal) that occupies only 55 mm$^2$. In an analogical way a visual system in computer vision requires a larger computational effort for the low-level part with respect to the middle or high level. Despite a quite detailed information about low-level and middle-level vision in the human vision, we have not much information about high-level. The well-known binding problem is not clear yet and opens a complicate relationship with neural studies and psychological ones. In computer vision, researchers disagree about a common canonized division of roles among vision levels. [7] explains that if a representation is based on arrays of numerical data that correspond directly to image data (pixel-wise operations), it is low-level vision. Representations based on symbolic descriptions of extracted image events, or view-specific symbolic instantiations of stored models and knowledge are intermediate level [7, 8, 9]. Representations that are view or scene independent are high level. A high-level representation thus characterizes general models and knowledge, as well as view independent 3-D models and knowledge of the current environment.

Thus low-level vision represents the basis on which other levels build a scene understanding. Such complex vision systems are currently fascinating researchers and advances in technology benefits great improvements in the computer vision. Currently vision systems are of extreme importance in many application fields and our world is rounded of cameras and displays. Industrial processes use robotic platform eyes equipped to control chain productions. All public buildings get a vision system for security purposes and surveillance. Automotive industry is exploring the possibility of a new concept of sensor based on smart cameras that advice the driver of unexpected dangers [10]. Many other applications, as military [11] or medical, benefit of computer and machine vision [12].

## 1.2 Vision processing platforms: State of the Art

Currently to afford the implementation of a vision system, different technologies are available. For research purposes and algorithm development, the faster and practical solution for a method validation is the high level programming in conventional general purpose processors. The processing power of these machines has grown up very quickly rising the Moore's law and the presence of many programming languages offers to the researcher a lot of possibilities for a rapid implementation. For instance, a very common platform for validation in engineering is the MATLAB environment. Imaging tools, information representation libraries for this language and the big efficiency with

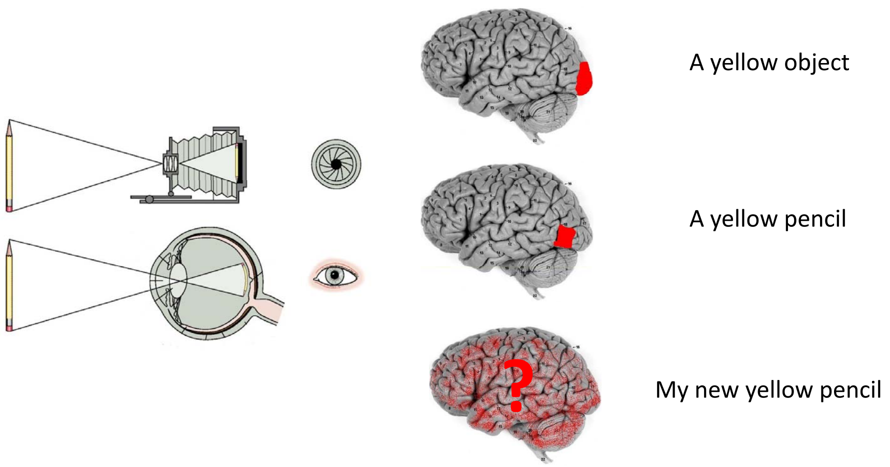


**Figure 1.1: Vision Levels** - Division of different vision levels (figure adapted from [6]). At low level (V1 and V2 regions of the human vision system) we process the whole image extracting cues: major effort. At middle level (V4, inferior temporal cortex) we associate a determinate object. At high level (binding problem) we recognize a specific object accessing to information external to the scene.

matrix operations allow a very short time of implementation and validation. Due to the time of realization this step can be considered the first one to address a faster implementation of a model.

To speed up the code and going on to potential industrial products different steps are required (Fig.1.2). Depending on our purposes we can stop on every one of these steps. If we have no constraint in size and power we can stop the implementation to a PC based code but we can speed up the low performance implementation with a more optimized code like in [13] or adopt specific hardware accelerators such as Graphic Processing Units (GPU). In the first case the optimization adopted can follow different strategies: from a simple compiler optimization to a low level code (assembler). The second one is generally a C-like implementation with specific environments such

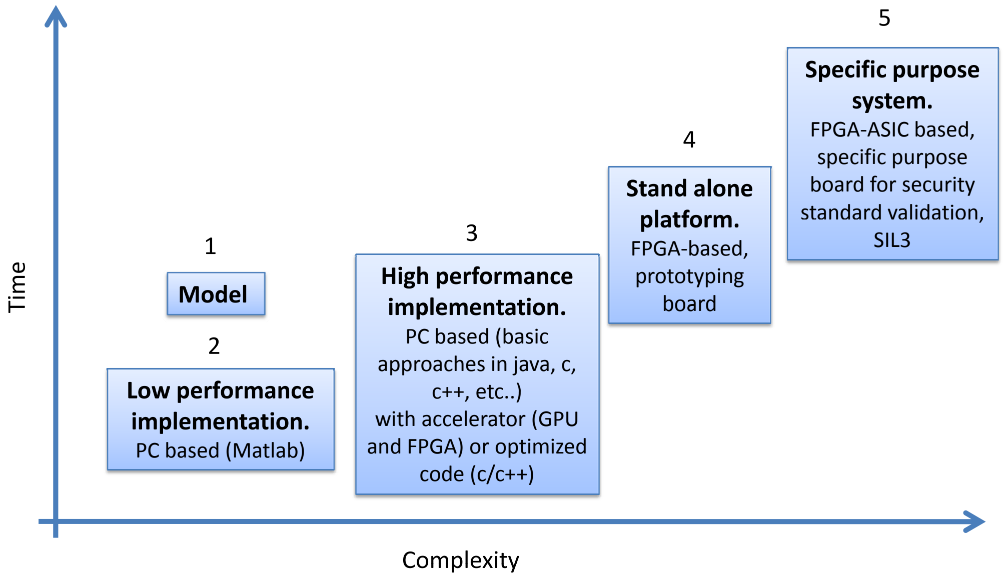

Time
Complexity
1
Model
2
Low performance implementation.
PC based (Matlab)
3
High performance implementation.
PC based (basic approaches in java, c, c++, etc..)
with accelerator (GPU and FPGA) or optimized code (c/c++)
4
Stand alone platform.
FPGA-based, prototyping board
5
Specific purpose system.
FPGA-ASIC based, specific purpose board for security standard validation, SIL3

**Figure 1.2: Development Stages** - Different development steps for a vision system. Process begins with a simple high level language until the synthesis on a VLSI industrial product. Steps are situated in a complexity vs. time plot.

as CUDA [14] that vary depending from the hardware platform. Currently due to the game industry the technology of GPU has grown up enormously and this platforms use the best solutions in parallelism and memory accesses. A more optimized work in terms of size and power consumption is a prototyping board implementation, for example on an FPGA based platform. This approach achieves a very high level of parallelism in operations with a reduced clock frequency: about two orders of magnitude less than a conventional processor or a GPU device. It benefits the power consumption and the use in portable applications. The last and more complex step for an optimized implementation is the design of a specific purpose system. Starting from the prototyping board previously validated is possible to produce a VLSI chip for industrial purposes that can be adopted for specific portable applications and can abates costs with a large production. Depending on our target application we could stop at different design stages or continue towards more complex steps. For example if we are interested in mobile robotics for research applications, a specific hardware platform provided with DSPs or FPGAs is the target choice and we can eliminate the optimization of the software code for commodity processor or the implementation on GPU that do not contribute to final work. Nevertheless, basic implementation using a simple language such as MATLAB, Octave or equivalent are appropriate. The last step (much more oriented to commercial purposes) needs an extra time for design and validation and involves other issues as system certification according to the target application and its respective Security Integrity Level (SIL). This last stage is a necessity of industry and generally is not addressed by most of the research fields. Currently, the literature presents different approaches for the diverse implementation steps presented. The first one (model in software) and the second one (optimized code and/or coprocessors boards) are the most active fields.

High performance has been achieved in accuracy for the many low level applications. Some contributions as [15] and [1] help us to understand the state of the art in the stereo and motion implementation approaches. Researchers contribute to the literature with the generation of diverse sets of images benchmark [2]. This valorous work allows an important validation for algorithms but, though we have reached a high level of accuracy for these synthetic benchmarks, new implementations are voted to an exasperate attempt to improve the actual level of accuracy with these benchmarks and do not address many of the practical aspects of the real world (real-time, illumination changes, unconstrained environments, etc). This problem is due to the evident difficulty of quantitative validation of real sequences. Current image correspondence techniques are mainly divided into two categories: local approaches and global approaches. Local (window-based) algorithms, where computation at a given point depends only on pixel values within a local spatial window, usually make smoothness assumptions for aggregating support implicitly. In order to increase the accuracy of estimations, particularly along depth borders, state-of-the-art algorithms deploy a variable support to compute the local matching cost rather than using, as in the traditional approaches, a fixed squared window. Conversely, most global methods attempt to minimize an energy function computed on the whole image area by employing minimization strategies such as variational techniques, Markov Random Field model, Graph Cuts (GC) approaches, Belief Propagation (BP), etc. [15, 16, 17]. Since this task turns out to be an NP-hard problem, the estimation is approximated by efficient strategies [15, 16, 17]. Moving towards an optimized implementation or a hardware specific architecture (right part of plot in Fig. 1.2) fewer contributions are presented in the literature. This means that the complexity of the implementation grows up significantly. We can find diverse GPU implementation of previous studied approaches but only some simple algorithms have been implemented in FPGA or DSPs processors. The reason for this actual tendency is the easier programming environment and the shorter "time to market" for commodity processors compared to DSP or FPGA based solutions. The evident problem of the utilization of standard PC based solutions is the power consumption and the considerable size unviable for many portable applications. For the other part an FPGA and DSPs implementation can afford a portable application and evolve into an industrial product. The comparison between DSPs and FPGAs shows that DSPs are better suited for low power applications whilst FPGAs are better option if the performance requirements are very high [18] or we plan to address the development of ASIC solutions. Works such as [19] and [20] represent very important contributions to the FPGA based vision on chip but at the same time lack of generic applicability. The main problems for existing FPGA approaches are the processing speed and the adaptation to large changes in the scene as in movements or in camera variations. Multi-scale approaches that can solve some of these problems present a high computational cost and are not hardware friendly. For this reason they are rarely adopted for hardware implementations. If we move to the last step of Fig. 1.2 we can find only few works as [21] that represent a very appreciable contribution for a specific purpose vision system on-chip.

## 1.3 Our contribution

In this work we start from existing vision models. We study and validate these models adapting algorithms to a hardware implementation and analyzing error due to the utilization of fixed point arithmetic (quantization degradation). Previous works of the literature as [19] and [20] lead directly to the step 4 of Fig. 1.2 without passing through the optimized code or the GPU implementation. We introduce for the first time vision architectures of a very high complexity and design methodologies. Range problem of existing approaches are solved with a multi-scale implementation. Hardware unfriendly operations such as warping are included in architectures with up to 4 clock domains. We introduce and optimize fine sharing strategies and study the performance vs. hardware utilization trade-off. The general idea is to obtain a balanced throughput reduction and a parallel processing unit sharing as it is displayed in Fig. 1.3. Diverse chip implementations have been explored to validate the modularity and scalability of the architecture.

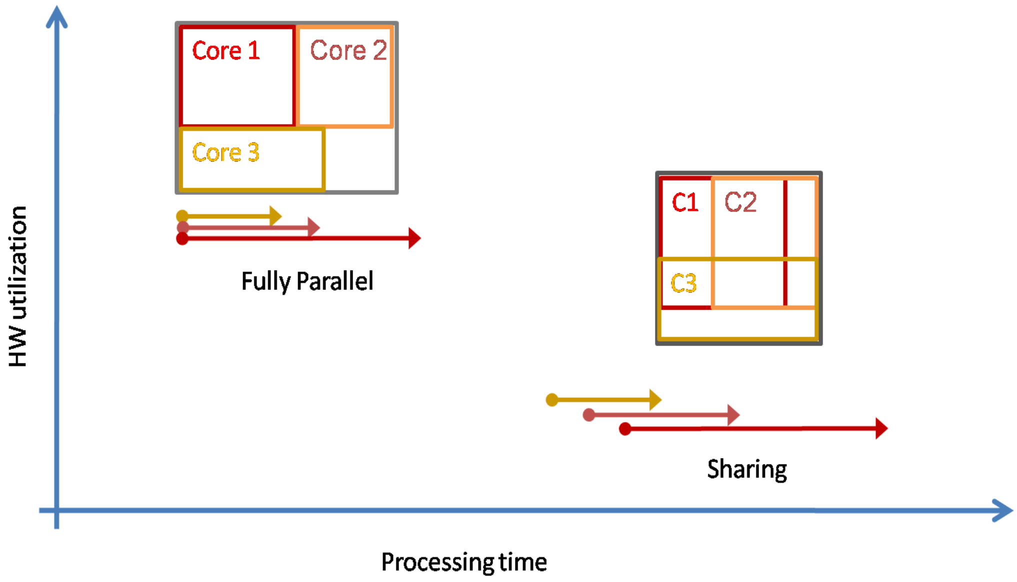


**Figure 1.3: Sharing strategy** - A fully parallel implementation (left) vs. a shared approach (right). With sharing we lose in throughput but save hardware recourse.

We study the adaptation of the design to the chip size and system requirements. Our minimum requirement is a real-time processing (25 frames per second) for an image resolution of 512x512 pixels. This work contributes to the advances in computer vision, especially in machine vision. We try to compare our approach with the state of the art providing quantitative results. Well-known benchmarks [2] with ground truth are used; furthermore we validate qualitatively the stability of the algorithm with real sequences exploring possible industrial applications as automotive.

## 1.4 Project Framework

The low level vision system has been developed within the European project "Learning to emulate Perception-Action Cycles in a driving school scenario" (DRIVSCO) [22] in collaboration with six different universities. The goal of DRIVSCO is to devise, test and implement a strategy of how to combine adaptive learning mechanisms with conventional control, starting with a fully operational human-machine interfaced control system and arriving at a strongly improved, largely autonomous system after learning, that will act in a proactive way using different predictive mechanisms. The research group at the University of Granada was involved in the implementation of the processing engine on chip. Inside our group, five different persons contributed to this work. The final implementation includes the development of a co-processing system with a FPGA-based platform. As indicated in Fig. 1.4 the complete work is composed by:

- External interface controller for GPIO, PCI-Express, Ethernet (for communication with PC, vehicle, robot, etc..)
- Memory Controller Unit
- **Processing cores (optical flow, stereo and local features)**
- Condensation modules
- Embedded processors (particulary suited for middle-high level vision algorithms development)

Our work, presented in this thesis, focuses mainly in the vision processing cores and the multi-scale architecture. The work in group obviously means a crossed collaboration with other members especially in testing tasks and validation of algorithms and models.

## 1.5 Outline

We start this work with a brief description of the harmonic representation used by our vision algorithm and a hardware friendly adaptation of it. Technological constraints and real-time requirements compel us to specific algorithm simplifications. On chapter 2 we study these changes for each single vision modality that we estimate: stereo, motion and local features (magnitude, orientation and phase). Following chapters describe every

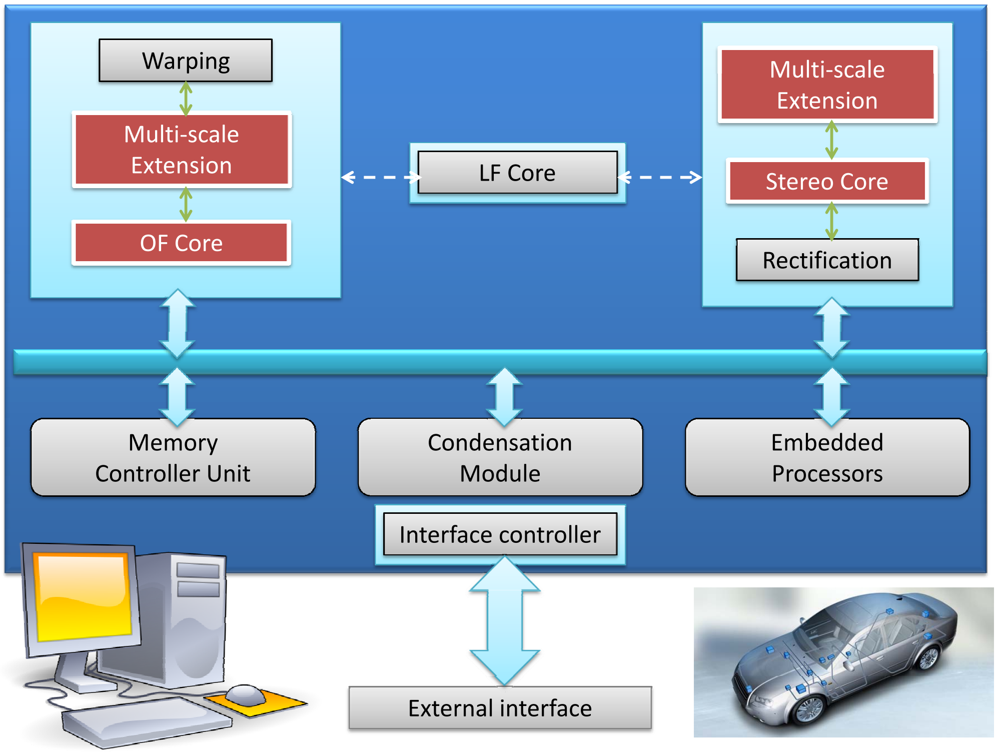


**Figure 1.4: Project framework.** - Block diagram of the DRIVSCO system developed by the Granada University group. Red parts have been addressed in this work. The system may be used as co-processor for a PC or as stand-alone platform (for example in a car).

single modality implementation and its multi-scale architecture. Chapter 3 presents a stereo architecture and chapter 4 an optical flow system. These two architecture architectures themselves represent a very important innovation for the state of the art. Furthermore another important step towards a vision system on chip is made in chapter 5 with the description of a large low level vision engine that includes all previous modalities in the same chip. A novel architecture and its design strategies are presented in this section. Further vision primitives can be easily incorporated in such system.

# 2

# Preliminary study towards a Hardware implementation

In this chapter we present a brief description of the original phase based algorithms used for our system development, ranking them with respect to other state of the art approaches. We remark here the stability of the phase information and compare it with other approaches. The original model complexity compels us to a specific hardware adaptation. This part aims to examine each of these hardware motivated modifications and how they affect the error. For this study we use a software simulation and leave the hardware evaluation for the following chapters. We repeat the study for every one of the single vision modalities that we have designed: disparity, optical flow and local features (magnitude, orientation and phase).

## 2.1 Optical flow and stereo computational models review

In order to address the initial goal of a high quality vision system, the preliminary stage is a proper study of the existent approaches and their possible adaptation for a hardware implementation. The method we are looking for has to possess a good trade-off between performance and computational cost. For example linear operation and well defined access to memory are hardware friendly operations. At the same time the algorithm has to provide an integrated framework for different vision modalities extracted based on the same features. In general optical flow methods can deal with a disparity computation with just some integration of epipolar constraint and a few modifications. Our goal is to find general methods with high potential sharing capabilities. Currently, exploring optical flow literature we can catalogue existing approaches as explained in [23] and add some novel techniques:

- Differential methods
  - Local
  - Global (typically variational approaches)
  - Surface models
  - Contour models
  - Multi-constraint models
  - Hierarchical approaches
- Frequency-based methods
  - Orientation selective filtering
  - Phase-based filtering
  - Hierarchical approaches
- Correlation based methods
  - Correlation-based matching
  - Hierarchical approaches
- Multiple motion methods
  - Line processes
  - Mixed velocity distribution
  - Parametric models
  - Temporal refinements methods

Among existing methods the best solutions in terms of accuracy for single modality vision are global approaches such as the variational ones. Variational methods as [24] are hybrid (local-global) approaches and achieve very high accuracy. But due to its iterative nature (minimization of energy functional) they are difficult to implement in real-time using reconfigurable hardware (typically slow for iterations due to the low clock frequency). Nevertheless there are some implementations using commodity processor and sophisticated minimization methods [24] or GPU based implementations [25] that are able to achieve the real-time processing. Unfortunately, for embedded applications, these techniques are not affordable. Frequency based algorithms follow variational approaches in accuracy. They present a robust response to unconstrained and unstable scenarios. A software comparative between different approaches can help us to make a choice. Discarded the variational methods we can focus on the other ones. By exploiting, on a local basis the spectral information content of the image signal (amplitude and phase), it is possible to derive perceptual entities, useful to gain interpretative elements of the observed scene, such as edges/contours, motion, and binocular disparity. Although most of the classical algorithms available in the literature rely upon the amplitude information, in the last two decades alternative techniques based on phase measures have been asserted themselves. The importance of global (Fourier) phase has been first demonstrated with respect to image coding and representation, by comparing modulus-only and phase-only image reconstructions [26], [27], and has been confirmed also in case of the local phase spectrum [28]. On that ground, the popularity of the phase information, as a robust feature descriptor, has risen in relation with the numerous important properties that have been reported and analyzed [29, 30, 31, 32], such as: (1) the capability of measuring changes much smaller than the spatial quantization (giving sub-pixel accuracy without a sub-pixel representation of the image, due to its continuous nature); (2) the stability with respect to small geometric deformations of the input; and (3) - perhaps the most desirable property - the invariance with both mean luminance and contrast (e.g., with respect to smooth shading and lighting variations), which makes phase, in principle, robust against typical variations in image formation. For these reasons, during the recent past, the phase from local bandpass filtering has gained increasing interest in the Computer Vision community and has led to the development of a wide number of phase-based feature detection algorithms in different application domains [19, 29, 33, 34, 35]. The harmonic representation will be the base for a systematic phase-based interpretation of vision processing, by defining perceptual features on measures of phase properties. From this perspective, edge and contour information can come from phase-congruency, motion information can be derived from the phase-constancy assumption, while matching operations, such as those used for disparity estimation, can be reduced to phase-difference measures. In order to motivate quantitatively the choice of a phase-based method we take for example the optical flow. Optical flow is the pattern of motion that results from the projection of object and induced scene motion on the retina of a, possibly moving, observer. It is represented by a vector field that contains a 2D velocity vector for every spatial location. The optical flow can be thought as the instantaneous positional velocity field (Gordon 1965) which associates with each element on the retina the instantaneous velocity of that element. Horn and Schunck [36] defined optical flow as follows: *"The optical flow is a velocity field in the image which transforms one image into the next image in a sequence. As such it is not uniquely determined. The motion field, on the other hand, is a purely geometric concept, without any ambiguity it is the projection into the image of three-dimensional motion vectors."* As described before, a lot of mathematical algorithms [29, 37, 38] try to find the velocity vectors that transform a frame of a sequence into the following one. One of the motivations for implementing a phase-based approach is its robustness against illumination changes that appears on real systems in unconstrained scenarios.

In order to demonstrate robustness of the phase-based method we have processed sequences with artificial illumination changes using several well known algorithms: Lukas and Kanade (local), Horn and Schunk (global) and a high accuracy model (that can be seen as an improved version of the Horn and Schunk one) that is based on the work of Brox et al. [39] and that we shall call as the variational method. We use a synthetic diverging sequence generated with a previous assigned ground truth as shown in Fig. 2.1. After the generation of the input sequence, we simulate the different local luminance. For each variation we measure the Angular Error (AE), the standard deviation (STD), and the density for valid values as described in [2]. In 2.2, we report the behavior of the system for the different variations (image artifacts). The variational method is based on [39] where, however, the data terms are based on the gradient and $L^2$ norm of the gradient thus making the method more robust against illumination changes. Since the LK (Lukas and Kanade) and the HS (Horn and Schunk) methods are based on intensity levels we expect that they will not be robust with respect to illumination changes whereas the variational method should yield more stable results. Another version of this method based on the phase input achieves a further improvement. This confirms our hypothesis on phase robustness applied to a different approach with a significantly higher computational complexity.

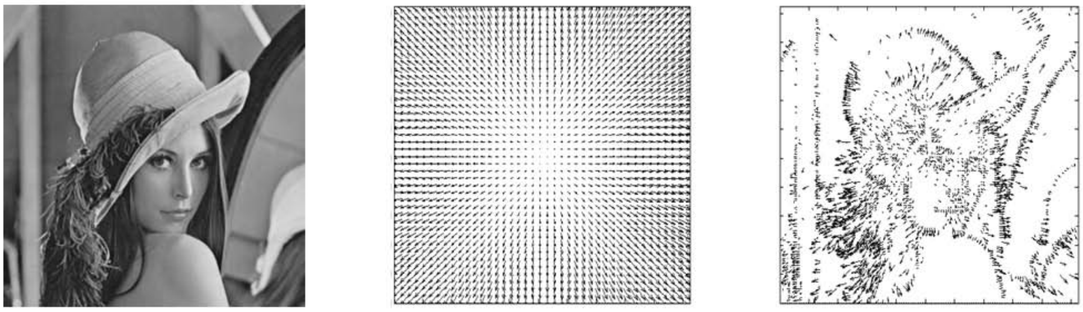


**Figure 2.1: Optical Flow** - Central frame of the input sequence (on the left), its correspondent ground truth (on the middle), and the optical flow computed in standard (original) conditions.

As we can see in Fig. 2.2, the phase-based algorithm is robust to variations compared with other methods. Obviously the variational method achieves a better accuracy in lack of variations but as shown in the plots its final error is worst than ours. Although a phase input to the variational method (dashed line in Fig. 2.2) gives more stability to this approach, it is worthwhile to remind that such large computational effort is not suitable for FPGA implementations. Note that in the case of brightness variation (additive noise) both approaches of variational method behave in a stable way: this is due to the nature of gradient, in this case the derivative is not affected by an additive constant. The algorithm has been applied with the same parameters and changing the input images. For contrast variation, we multiply each image by a 1-$\alpha$ where $\alpha$ is a variable in the interval [0:0.4]. For brightness variation, we add a global constant to the pixels of the image. We use a variable $\beta$ in the interval [0:25] considering an image gray color map from [0:255]. Taking into account that we have three temporal images, we applied a positive variation to the future frame, a negative variation to the previous one and no variations to the middle frame. Interestingly enough, for both the contrast and the brightness variations, the AE increases with the variation in the case of the LK method and the HS but it seems to maintain its base value for the phase-based approach. It confirms the hypothesis that phase information is very robust and, on the other hand, shows the well known problem of gradient models to illumination variations. Note that for all tests, we have a density of 98% valid values.

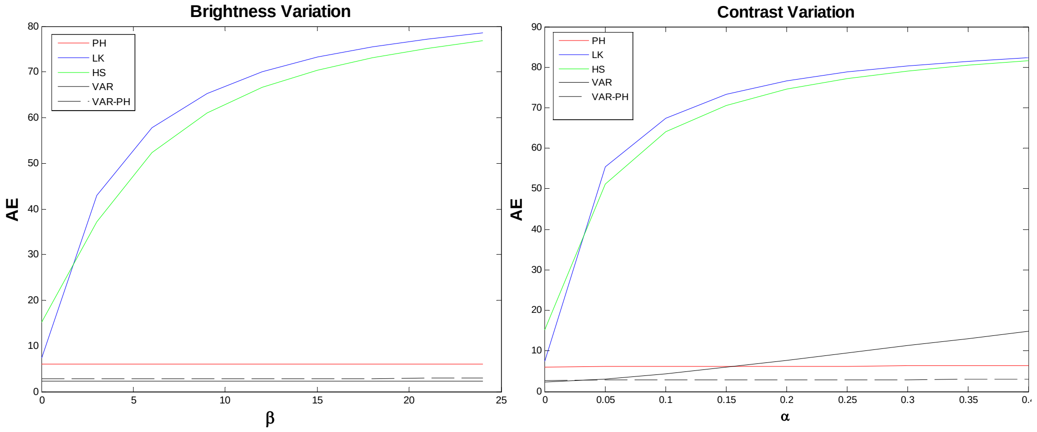


**Figure 2.2: Phase robustness** - System behavior for different camera variations (image artifacts). On the left side, a variation in contrast; on the right, a variation in brightness. On the y axis the AE value is reported; on the x axis, $\alpha$ is reported for the contrast while $\beta$ is shown for the brightness.

### 2.1.1 Reference optical flow computation model

We have focused on the phased-based computing model proposed in Sabatini et al. [40] and [41] which is a multi-scale extension of the Gautama and Van Hulle original approach [33]. The advantage of phase-based approaches has been pointed out by different authors because their robustness against luminance variations and cameras imbalance problems. Furthermore, these phase-based approaches lead to a better behavior against affine transformations (for instance due to different cameras perspectives) [29, 30]. Spatially localized phase measurements can be obtained by filtering operations with quadrature-pair filters. It is possible to use the original Gabor filters or some Gabor-like approach as band pass steerable filters based on Gaussian derivatives as described in [42]. The filter response, obtained by convolving the images with the oriented quadrature filters is used as input to the phase-based algorithm. For every spatial orientation $\theta$ and location x, the temporal phase gradient in time t, noted as $\phi_\theta(x,t)$, is computed through a linear least-squares t to the model as indicated in (2.1).

$$\phi_\theta(\mathbf{x},t) \approx c_\theta(\mathbf{x}) + \phi_{t,\theta}(\mathbf{x})t \tag{2.1}$$

A simple unwrapping technique is used to cope with the periodicity of the phase. Next, for each orientation $\theta$ a component velocity is computed directly from $\phi_{t,\theta}(x)$:

$$v_{c,\theta}(\mathbf{x}) = \frac{-\phi_{t,\theta}(\mathbf{x})}{2\pi(f_{x,\theta}^2 + f_{y,\theta}^2)}(f_{x,\theta}, f_{y,\theta}) \tag{2.2}$$

Where f$_{x,}$ and f$_{y,}$ are the spatial frequency values at $\theta$ orientation. The f components are not explicitly computed assuming phase linearity that allows translating the sum of squared frequency values to a constant characteristic which is the peak frequency of the filter used. Note that the spatial phase gradient is substituted by the radial frequency vector. The reliability of each component velocity at each orientation $\theta$ is measured by the Mean Squared Error (MSE) of (2.3), where n is the number of frames and $\Delta\phi_\theta(\mathbf{x},t) = (c_\theta(\mathbf{x}) + \phi_{t,\theta}(\mathbf{x})t) - \phi_\theta(\mathbf{x},t)$.

$$MSE = \sum_t \frac{(\Delta\phi_\theta(x,t))^2}{n} \tag{2.3}$$

Therefore, a linear regression value gives a temporal derivative of the phase and the quality of this linear fit provides a good reliability estimator. Please note that, for uniform motions (no acceleration), the flow computation benefits of the use of a large number of temporal frames. Unfortunately, real-time and (accessible) memory constraints reduce this number to few frames, using typically 3-5 frames. Finally, provided that a minimal number of reliable component velocities are obtained (threshold on the MSE of each orientation (2.3)), an estimate of the full velocity is computed for each pixel by integrating the valid component velocities at that pixel only, as indicated in (2.4).

$$v^*(\mathbf{x}) = \underset{v(\mathbf{x})}{\operatorname{argmin}} \sum_{\theta \in O(\mathbf{x})} \left( \|v_{c,\theta}(\mathbf{x})\| - v(\mathbf{x})^T \frac{v_{c,\theta}(\mathbf{x})}{\|v_{c,\theta}(\mathbf{x})\|} \right)^2 \tag{2.4}$$

Where O(x) is the set of orientations at which the valid component velocities have been obtained for pixel x. As a summary, the following are the different processing stages for the mono-scale optical flow:

S1 Convolution with 8 Quadrature pair filters tuned at different orientations.

S2 Phase calculation for each orientation with an arctangent core.

S3 Temporal filter: wrapping of phase values considering their periodicity for temporal derivative estimation. Three temporal frames are used.

S4 Velocity component computation as described in [40, 41] and indicated in 2.2.

S5 Threshold operation based on confidence values and combination of valid values for each orientation for the final velocity vector estimation. Depending on the final confidence value, the optical flow vector or a tag indicating *non valid* data is generated for each pixel.

### 2.1.2 Reference binocular disparity computation model

This model can be extended to stereo computation: disparity can be calculated as an optical flow with only two frames (left and right). In our case we adopt a simplification that uses the phase difference between left and right image, as described in [43]. In this case, the phase difference is computed from 2.5.

$$\phi(x) = \frac{(\phi^L(x) - \phi^R(x))}{k(x)} = \frac{1}{k_0} atan2(C^R S^L - C^L S^R, C^L C^R + S^L S^R) \tag{2.5}$$

where we note with $\phi^L$ and $\phi^R$ the left and right local image phases, $C^R$ and $C^L$ correspond to the values of left and right image pixels after convolving with the even part of the quadrature filter, while $S^R$ and $S^L$ are the results after convolving to the odd quadrature filter outputs, arctan2 stands for the principal part of the argument (i.e. the argument belongs to $[-\pi, \pi]$) and finally, k(x) is the average instantaneous frequency of the band pass signal, which can be approximated by $k_0$, the quadrature filter peak frequency. Basic steps of mono-scale computation can be summarized as follows:

1. Even (C) and odd (S) filtering with quadrature filters pairs of left and right images.

2. Disparity computation using equation 2.5 at each orientation and threshold operation assuming $k(x) \approx K_0$

3. Chose of final disparity estimation between different orientations: median value.

We remark that threshold operations are based on the energy of the signal and the last choice of the final values is evaluated through a median value as described in [40]. The median value can be replaced by a more accurate method that for example takes information from exterior; this information can be provided from another algorithm or from a middle or high level control as in the novel concept of Signal to Symbol loop [44].

### 2.1.3 Local features: magnitude, orientation and phase

Starting from filter response is possible to calculate local features from real part $C_\theta$ and from imaginary part $S_\theta$. We extract the magnitude and the phase directly from the information of the filter at each orientation. As described in [45] and [46], different methods can be used for accurate edge detection. Quality of the first filtering stage influences the final results and as described in [46] better performances are obtained with second order Gaussian derivatives filters. If we consider 8 oriented filters (computing using Gabor or Gaussian Derivatives), is likely that the local orientation of some features do not fit this discrete number of orientations. Under this circumstance, we require to interpolate the feature values computed from this set of outputs in order to estimate the filter output at the proper signal orientation. Different methods can be used. We note $E_i$ and $P_i$ to the magnitude and phase of the filter oriented with $angle = i * \pi/N$ and noted by $h_i$. This filter is expressed by:

$$h_i = c_i + js_i \tag{2.6}$$

And the primitives features are computed with this filter orientation and computed as:

$$Filter\,energy \to E_i = [c_i]^2 + [s_i]^2 \tag{2.7}$$

$$Filter\,phase \to P_i = arg(c_i, s_i) \tag{2.8}$$

If only the main orientation information is required (1-D local signals), we can apply several strategies to interpolate the primitives from this multi-valued set:

i Winner- take-all. We will take for each pixel the phase, energy and orientation of the filter with maximum energy.

$$E_{local} = E_{max} \qquad P_{local} = P_{max} \qquad \theta_{local} = \theta_{max} \tag{2.9}$$

ii Weighted-average: (we consider linear case, though the energy can be power to different orders).

$$E_{local} = \sum_i E_i^N \qquad P_{local} = \frac{\sum_i P_i E_i}{\sum_i E_i} \qquad \theta_{local} = \sum_i \frac{\theta_i E_i}{\sum_i E_i} \tag{2.10}$$

where all angles are properly shifted for avoiding angle wrapping effects.

iii Tensor-based method [45]. Based on a local tensor that projects the different orientations, information can be computed as follows (where j stands for the complex unit):

$$\begin{aligned} E_{local} &= \sum_i E_i^N && (2.11)\\ \theta_{local} &= \frac{1}{2}\arg\left(\sum_i \frac{4}{3}\sqrt{c_i^2 + s_i^2}\exp(j2\theta_i)\right) && (2.12)\\ P_{local} &= \arctan\left(\frac{s}{c}\right) && (2.13)\\ c &= \sum_i c_i \cos^2\theta_i - \theta_{local} && (2.14)\\ s &= \sum_i s_i \cdot sign\cos(\theta_i - \theta_{local}) \cdot \cos^2(\theta_i - \theta_{local}) && (2.15) \end{aligned}$$

iv Energy Fourier series expansion for Gaussian derivatives based approach. As described on [42], using the n$^{th}$ Gaussian derivative G$_n$ and its Hilbert transform H$_n$ as band pass filter oriented to the angle $\theta$, we have that the energy at this orientation is expressed by:

$$E_n(\theta) = [G_n^\theta]^2 + [H_n^\theta]^2 \tag{2.16}$$

Writing these functions using the separable basic filter outputs, this equation can be expressed as a Fourier series in angle and described as:

$$E_n(\theta) = C_1 + C_2\cos(2\theta) + C_3\sin(2\theta) + high\ order\ terms \tag{2.17}$$

Note that values of coefficients $C_i$ can be found on [42] for n=2 case. From equation 2.12, local orientation is computed based on the lowest frequency term as:

$$\theta_{local} = \frac{arg(C_2, C_3)}{2} \tag{2.18}$$

The previous equations work only for the Gaussian derivatives filters case. For the sake of generality, the energy is estimated using equation 2.16 and the phase using

2.8 taking into account that, for n even, $c_i$ is $G_n$ and $H_n$ is $s_i$ and the opposite for odd values of n.

We basically based our approach on the iii method because it is independent from filtering stage (Gabor or steerable filters) and it achieves a high accuracy.

## 2.2 Model modifications towards a hardware friendly implementation

The original algorithm presents some high cost operations for a hardware system. For example we know that in FPGA we have to reduce memory accesses as much as possible and also to reduce the memory utilization. Thus it is better a limitation in data width and in quantity of stored data. Critical operations for memory are for example the compensations (warping) of values in the pyramidal processing. In this section we analyze all the hardware friendly modifications and their accuracy loss. The analysis is repeated for all different features extraction and includes typical simplification as fixed point representation.

### 2.2.1 Optical flow model modifications for hardware implementation

As previously explained, we study different modifications with respect to [40] and how they affect the optical flow computation. They can be summarized as follows:

1. Warping on images

2. Gabor filters vs. Steerable filters

3. 5 frames vs. 3 frames in the temporal window

4. Floating point vs fixed point arithmetic

Before starting the hardware implementation, we explore all these changes or simplifications in a software simulator of the algorithm and report the results of accuracy degradation for all these cases. This study is of crucial importance in order to evaluate the impact of these modifications in the system accuracy. The impact in terms of computational resources and performance are estimated based on our previous experience and partial implementation of critical stages. The proposed modifications are applied

**Table 2.1:** Comparison of warping techniques in optical flow (phase vs image) for "Yosemite" sequence.

| Measure | Warp on Phase | | Warp on Image | |
|---|---|---|---|---|
| | 3 scales | 4 scales | 3 scales | 4 scales |
| Density | 81.05 | 81.81 | 74.44 | 71.83 |
| Ang. error | 2.38 | 2.14 | 3.16 | 3.17 |
| Std deviation | 3.20 | 3.12 | 3.35 | 3.54 |

progressively. First of all, we study the warping simplification. Original method proposed in [40] operates with a warping on the 8 oriented local phases of the image. This implies the repetition of a large number of warping operations, one per each orientation but has the advantage of reducing errors produced by wrong matching values across the pyramid (wrong warping values provide artifacts on the warped image that produce wrong phase values and reduce accuracy over the scales). Although some accuracy is lost, a hardware friendly implementation requires a warping operation on images before the filtering operation. This is a mandatory simplification for a FPGA based system but for example can be not a problem for GPU based system where memory capacity and bandwidth benefits of last advances on memory technologies. We cannot avoid this change but we report drawbacks of the new implementation. On Table 2.1 we appreciate the accuracy loss of the warping on images and in Fig. 2.3 a qualitative result for the Yosemite sequence.

Next simplifications are tested on the mono-scale version. The first is the reduction of the number of frames in the temporal sequence the original approach use 5 frames but we will work only with 3 frames. This choice is motivated by the restriction of external memory resources and accesses. Note that in certain situations where the velocity constancy assumption over 5 frames is violated, the current approach leads to better solutions. Unfortunately this is not the case for most of the situations and therefore we should consider this modification as a degradation of the model: detailed values are reported in Table 2.2. The accuracy loss is measured with respect to the original algorithm with warping on phases.

The modification of the filters (steerable filters instead of Gabor ones) significantly affects the accuracy for a mono-scale approach but it is not so important for multi-scale schemes (Table 2.4). For the steerable filters we adopt the second order derivative

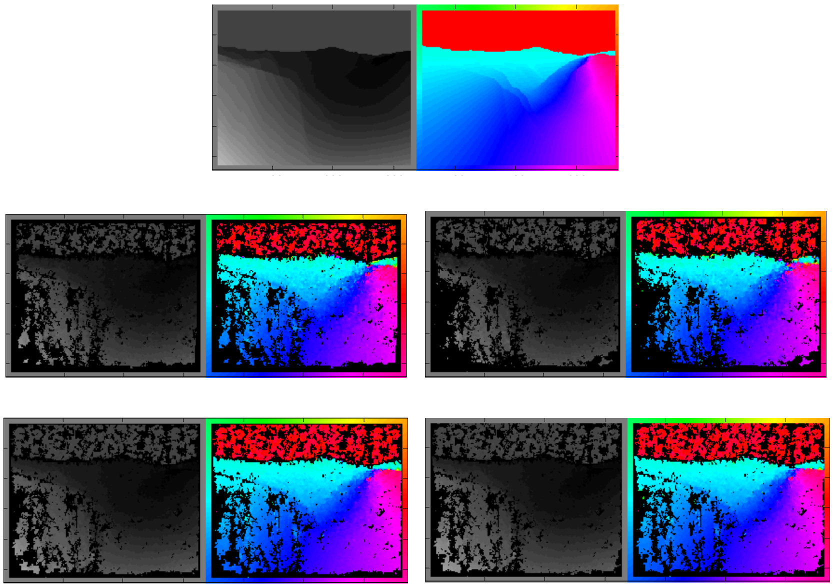


**Figure 2.3: Optical flow: warp on phase vs image** - On the first row the true optical flow for the Yosemite sequence. The gray-scale image represents the module (bright colours represent fast motion) and the coloured image the velocity direction. The warping on images scheme is displayed in the second row. Both design choices use a multi-scale scheme with three scales on the left and 4 scales on the right. The third row shows the warping on phase scheme.

approximation with 9 taps as described in [42]. The steerable filters choice implies a smaller computational cost but it leads to a significant degradation of the error. Thus this simplification has a high cost in terms of accuracy loss and shall only be adopted in case of addressing low cost approaches. On a multi-scale extension we can consider also this approximation. One of the drawbacks of the original Gabor filters is that they are not separable, which means that a hardware implementation of them requires a huge amount of hardware resources. For this reason, we adopt a separable version of these filters (Gabor filters) as described in [47] and extended to 8 orientations by Pauwels [48]. The kind of arithmetic used plays a key role in terms of the accuracy and resources utilization of embedded systems. Nowadays FPGA are able to synthesize floating point

**Table 2.2:** Accuracy loss due to the reduction of frames in the temporal window. The AE is expressed in degrees. Note that the density is also changed.

| # frames | AE | STD | Dens |
|---|---|---|---|
| 3 | 4.06 | 7.33 | 96.7 |
| 4 | 2.33 | 3.43 | 86.8 |
| 5 | 1.83 | 2.25 | 80.1 |

units but as high cost in terms of resources. Therefore, we shall make an effort on using fixed-point arithmetic and properly studying the quantization error motivated by this choice. For the fixed point study, we made an exhaustive bank of tests changing bit width for all the five stages of the architecture described in Section 2.1.1 accordingly to the method described in [20]. As error measure we use Mean Absolute Error (MAE) for the stereo algorithm and the Angular Error (AE) for the optical flow defined as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i| \tag{2.19}$$

$$AE = \cos^{-1}(\hat{c} \cdot \hat{e}) \tag{2.20}$$

Quantization is applied gradually and when a choice is made on a processing stage this quantization is fixed and the test goes on with following stages. The quantization error produced by previous stages is lossy and cannot be restored. Final bit width for all stages is reported in Table 2.5. Note that in the convolution we are loosing precision for the sake of hardware saving, in a high performance circuit it is possible tune this parameter. In all the others stages we generally maintain the accuracy; for example in case of addition or difference we add a bit more to the result, in case of multiplications we give to the results the sum of factors bit width while in case of division is needed a study of possible results. We apply the same simplifications of Table 2.3 to the multi scale approach and repeat all tests; results are reported in Table 2.4.

### 2.2.2 Disparity model modifications for hardware implementation

In the disparity computation we adopt, where possible, the same modifications. Starting from the warping on images and analyzing results of Table 2.6 and Fig. 2.5, we appreciate again how the accuracy lost due to this modification. In the case of disparity we cannot obviously operate a reduction of frames but we can proceed as in the optical

**Table 2.3:** Percentages of accuracy loss and final model error for different model simplifications using the single scale approach and ”Yosemite” synthetic sequence [1]. The % Lost is computed taking as reference the error obtained with the warping on images algorithm. AE stands for Angular Error as defined in [1] and is presented in the last column. The regularization includes a cascade of two 3x3 median filters at each scale output (see Appendix A).

| Change | % Loss | AE(dens) |
|---|---|---|
| Original (warp on images) | 0 | 9.53°(58.72%) |
| Original + median filter | -11% | 8.52°(60.12%) |
| 3 frames + Gabor filters | +50% | 14.37°(58.84%) |
| 3 frames + Steerable filters | +140% | 22.88°(54.21%) |
| 3 frames + Gabor filters + Fixed point | +50% | 14.15°(59.59%) |
| 3 frames + Gabor filters + Fix. point + regularization | +44% | 13.81°(59.19%) |
| 3 frames + Steerable filters + Fix. point + regularization | +139% | 22.94°(52.78%) |

**Table 2.4:** Percentages of accuracy loss and final model error for different model simplifications using the multi scale approach with warping on images and the Yosemite synthetic sequence [1].

| Change | % Loss | AE(dens) |
|---|---|---|
| Original (warp on images) | 0 | 4.57°(88.36%) |
| Original + median filter | -28% | 3.28°(94.02%) |
| 3 frames + Gabor filters | +84% | 8.44°(96.38%) |
| 3 frames + Steerable filters | +140% | 10.99°(96.84%) |
| 3 frames + Steerable filters + Fixed point | +166% | 12.18°(86.99%) |
| 3 frames + Steerable filters + Fix. point + regularization | +82% | 8.34°(99.11%) |
| 3 frames + Gabor filters + Fix. point + regularization | +23% | 5.64°(95.32%) |

**Table 2.5:** Bit-depths of the input of each processing stage as described in Section 2.1.1. The first values represent the signed integer part and the second values, the fractional one. Output of S5 has [8 4] that is the representation for the final optical flow values.

| Stage | Input bits |
|---|---|
| S1 | [8 0] |
| S2 | [8 2] |
| S3 | [3 7] |
| S4 | [5 5] |
| S5 | [7 5] |
| Output | [8 4] |

**Table 2.6:** Comparison of warping techniques in stereo (phase vs image) for "Tsukuba" sequence.

| Measure | Warp on Phase | | Warp on Image | |
|---|---|---|---|---|
| | 3 scales | 4 scales | 3 scales | 4 scales |
| Density | 100 | 100 | 91.64 | 100 |
| Avg error | 1.63 | 0.32 | 2.05 | 0.39 |
| Std deviation | 3.75 | 0.61 | 1.99 | 0.63 |

flow for the filters and for the fixed point representation. A comparative between second order derivative Gaussian filters and Gabor filters with 11 taps shows that we have an affordable accuracy loss in the well-known "Tsukuba" sequence (Fig. 2.6). At the same time we can remark that steerable filters approach needs more scales to achieve similar results.

As commented before reconfigurable devices are very well suited for parallel processing operations but if we instantiate large number of parallel computing elements they need to be small and therefore are limited in terms of data representation and arithmetic. Though floating point processing units can also be synthesized, they are requiring a larger amount of resources and this motivates the utilization of fixed-point arithmetic. For all our operations we use a fixed point notation so system precision is strongly dependent from bit precision of fractional part. Different precisions in fractional part of variables may produce large quantization errors. To avoid error propagation we study bit width at first processing stage and we go on studying bit width in the next stages maintaining a suitable bit precision for previous ones. In Fig. 2.7

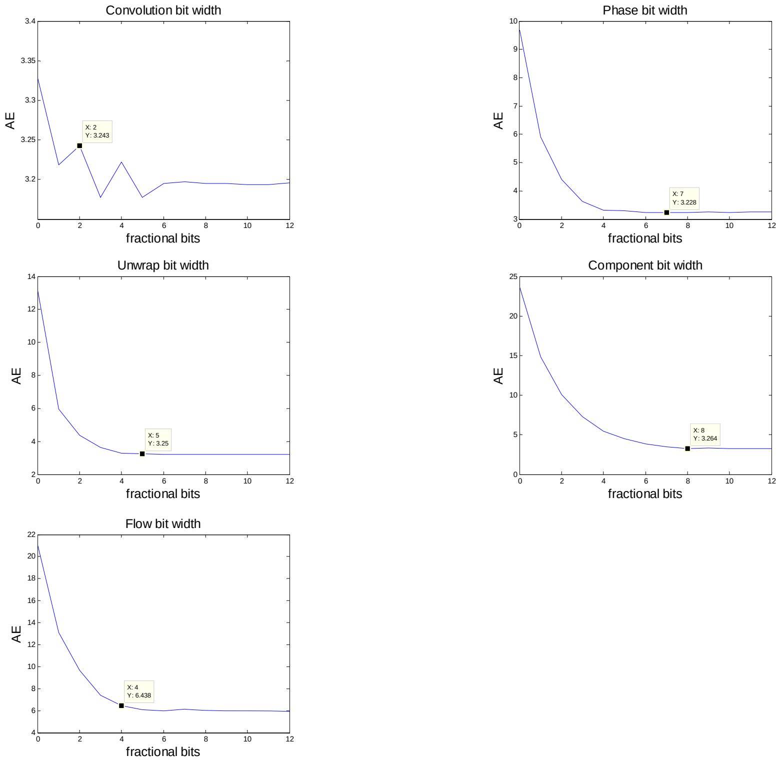


**Figure 2.4: Optical flow: bit width study** - Bit-width study for the different stages of the mono-scale optical flow core for the Yosemite sequence. a) AE for different convolution fractional bit-widths (S1 output). b) phase fractional bit-width (S2 output). c) unwrapping fractional bit-width (S3 output). d) component fractional bit-width (S4 output). e) fractional part of optical flow vectors (S5 output). Our different choices are indicated on the Figures by means of square marks.

are plotted several critical stages beginning from kernel bit width and ending with disparity bit width. In the choice of final bit-width we overestimate the number of bits required for several critical stages. This is because the bit-choice is sequence dependent. Therefore, in order to avoid over fitting for a particular scenario, it is better to increase the variables bit-width (at cost of higher resources utilization). For a specific scenario, more constrained bit-width values could be chosen but we do not go this way for the sake of circuit generality. For a high performance circuit it is possible to increment precision for stages or even implement customized floating point units.

Study is done on mono-scale disparity core variables because this stage mainly constraints the system accuracy. After this study we choose a suitable bit precision to

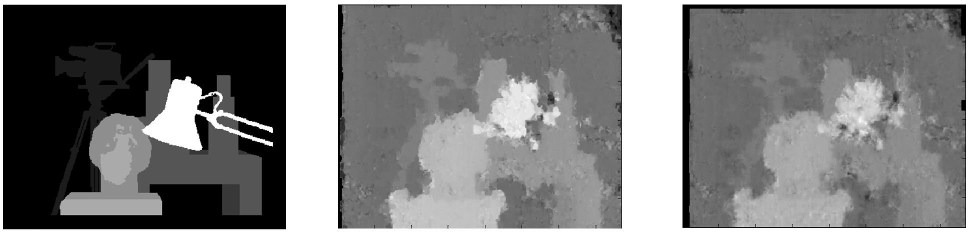


**Figure 2.5: Stereo: warp on phase vs image** - On the left the true disparity for the "Tsukuba" sequence. The qualitative result for the warping on phase (centre) and the warping on image (right).

reduce hardware consumption and obtain a feasible final MAE of 0.60 pixels for the multi-scale system. The MAE is computed comparing the final hardware simulation results with the ground truth. For each stage, we take the bit-width where the system accuracy starts to be approximately constant. In such a way, even working with a very limited number of bits we can achieve accuracy quite close to the software version.

### 2.2.3 Local features model modifications for hardware implementation

For the computation of magnitude, orientation and phase the hardware adaptations are the fixed point representation and an algorithm simplification to reduce computational effort. Equations of Section 2.1.3 are converted in the simpler versions, as follows::

$$M = \sqrt{\frac{\sum_{\theta=1}^{N} E_\theta}{N}} \tag{2.21}$$

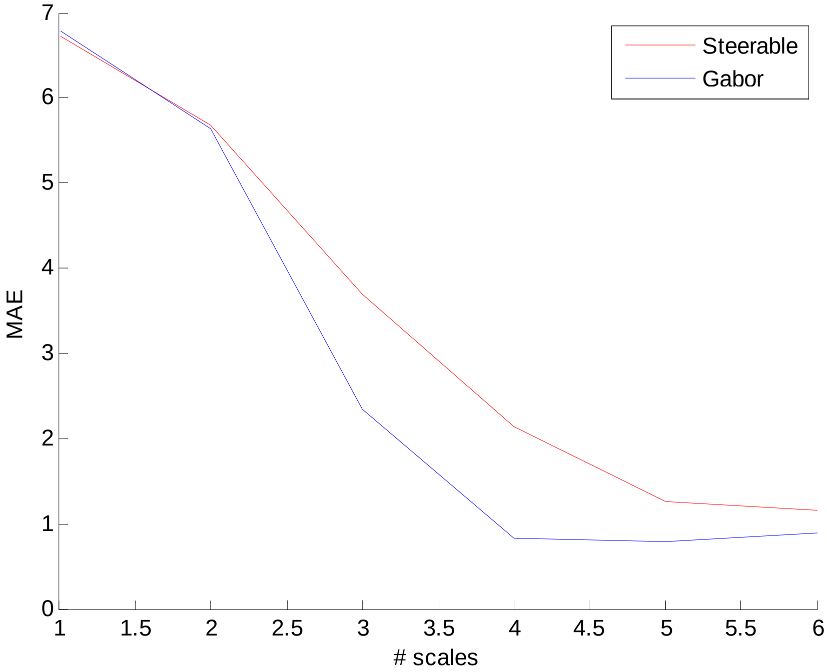


**Figure 2.6: Stereo: iteration Steerable vs Gabor** - Average error evolution for the Tsukuba sequence with the change of iterations.

$$\phi = atan2(\sum_{\theta} C_{\theta}, \sum_{\theta} S_{\theta}) \quad (2.22)$$

Where N is the number of different $\theta$ orientations and $E_{\theta}$ is the energy calculated for the orientation $\theta$:

$$E_{\theta} = C_{\theta}^{2} + S_{\theta}^{2} \quad (2.23)$$

Local orientation is calculated starting from mean energy along orientations with some simplification of methods described in [45].

$$\theta_{local} = \frac{1}{2} atan2(\sum_{\theta} E_{\theta} \sin 2\theta, \sum_{\theta} E_{\theta} \cos 2\theta) \quad (2.24)$$

The major benefit for this kind of processing is the quality of filtering operations. In this work we optimize the algorithm for good stereo and optical flow estimation, thus if we share the adopted filters for a further calculation of local features we fix also the precision for this stage. Thus the local features precision depends strongly from the

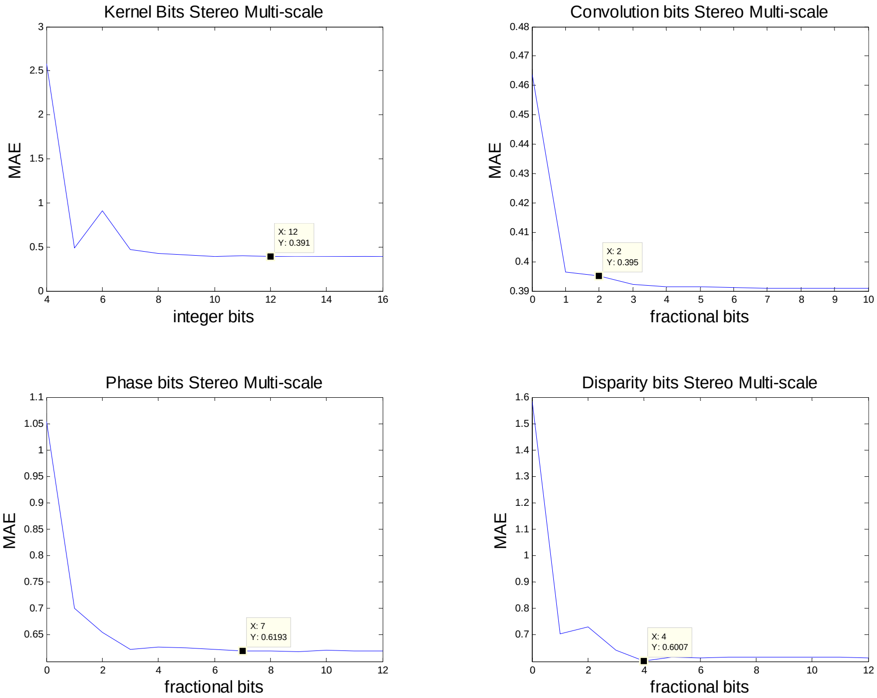


**Figure 2.7: Stereo: bit width study** - a) MAE for varying convolution kernel integer bit width. b) for trigonometric fractional bit width. c) for convolution fractional bit width. d) for fractional part of disparity.

other modalities. Especially we see in next chapters how local features are integrated in the stereo core.

## 2.3 A hierarchical approach

One of the main important contributions of this work is the implementation of the multi-scale version of the algorithm into the FPGA. As difference to hardware devices that include warping circuitry (for instance texture units in GPUs), we have had to develop our own high performance warping circuits. This is a very memory intensive operation for reconfigurable devices that, as we will shown, has motivated the development of a customized memory control units for access scheduling. Due to phase periodicity, phase-based techniques can only detect shifts that do not exceed half the filter wavelength [30]. To extend this range, a coarse-to-fine control strategy can be used [49]. An efficient solution involves the use of an image pyramid, in which the image resolution is halved at each level [50]. Specifically, a coarse-to-fine Gaussian pyramid [51] is constructed, where each layer is separated by an octave scale. Accordingly, the image is increasingly blurred with a Gaussian kernel and sub-sampled to build the image pyramid. At each pyramid level k, the sub-sampling operation reduces the image resolution to a half in height and width respect to the previous level k-1, reducing also the values range presented at this scale and helping the filters to properly tune their response. By applying the original filters to each level of the pyramid, the detectable range of shifts is doubled each time. The control strategy starts at the lowest resolution and uses stereo estimation obtained with a mono-scale method to warp the images at the next higher resolution so that the estimated values are removed [52]. The residual estimation is then within the range of the filters applied at that level. The algorithm which we use is particularly suitable for this warping strategy since it uses strictly local information. Propagation of reliable values along scales is different depending on the algorithm. For example in stereo implementation, only disparity values that can be reliably computed at the highest resolution are retained. In other words, if the refinement made at the highest resolution to a lower resolution estimate (that was reliable at that lower resolution) is unreliable, the disparity value is discarded and not included in the density counts of the next section. The procedure starts at the lowest resolution. The phases computed at this level and the estimations for this level are estimated accordingly to the algorithm. Estimated values are then transformed (multiplied by two) to the next scale and the filter outputs at that level are warped to compensate for the effects of these phase differences. All estimation is recalculated with new information and is propagated to the next level; the procedure is repeated until the original resolution is obtained [53]. Moreover, we have included a multi-oriented analysis of the image that computes the values with filters tuned at 8 different orientations, in the case of stereo we have only 7 (vertically oriented filters cannot be used because they do not have projection on the horizontal axe where the displacement takes place). All operations can be summarized as follows:

1. Selection of the number of scales according to the image resolution and expected values range presented in the scene. A pyramid representation is produced for the left and right images: only left if we are not calculating disparity.

2. Processing iteration for each scale:

   - Computation of estimated values using the proper algorithm: stereo or optical flow.
   - Merge with previous values (not necessary in the first step).
   - A simple median filter is used to regularize the results (see Appendix A).
   - Expansion of merged estimations multiplying values by two. Bilinear interpolation of values during up-sampling operations (this step is not operated for the last scale).
   - Warping pyramid images with expanded images to cancel already computed values reducing ranges (this step is not operated for the last scale).

These operations are implemented at each scale and propagated from coarser scales to finer scales along the multi-scale pyramid. It is worthwhile to mention that the warping operation uses a backward scheme with sub-pixel accuracy (computed using bilinear interpolation). Each warped pixel generally requires 4 neighbour pixels and therefore, a smart memory access scheme will be necessary for the algorithm implementation in order to avoid performance penalties. This multi-scale extension allows increasing the estimations range compatible with our approach. Moreover, the multi-oriented scheme allows to properly tune (optimize) the phase estimations according to the image structure presented at each pixel and, therefore, to increase the system accuracy.

## 2.4 Multi-scale operation vs temporal oversampling approach

Such multi-scale approach represents a high computational effort especially in the compensation of values along scales. It is possible also some mono-scale versions of algorithms but they present limitation in accuracy. In the case of optical flow, for example, the accuracy loss of this simplification can be compensated with a high frame rate camera operation. In Fig. 2.8, we visually illustrate how the Motion Range per Second (MRS) for a mono-scale algorithm with a high frame rate is equivalent to a multi-scale one with a low frame rate. We assume that accordingly with the size of the spatial filters, the Motion Range (MR) between two images is approximately 1.8 pixels. We also remind that multi-scale operations such as warping, image pyramid, and expansion increase the computational errors by introducing outliers in wrong matching values. The equivalence of multi-scale and mono-scale range is approximately valid up to an implementation with three scales for an oversampling temporal factor of around 6 (Fig. 2.8). If we require a higher detection range with a mono-scale version we need a very prohibitive frame rate. While one scale more allows us an MRS of 675, for the same MRS we need a frame rate of 375 frames per second. Furthermore a good behavior of a mono-scale algorithm in optical flow depends from several factors as sensor speed, final application and sequence scenario.

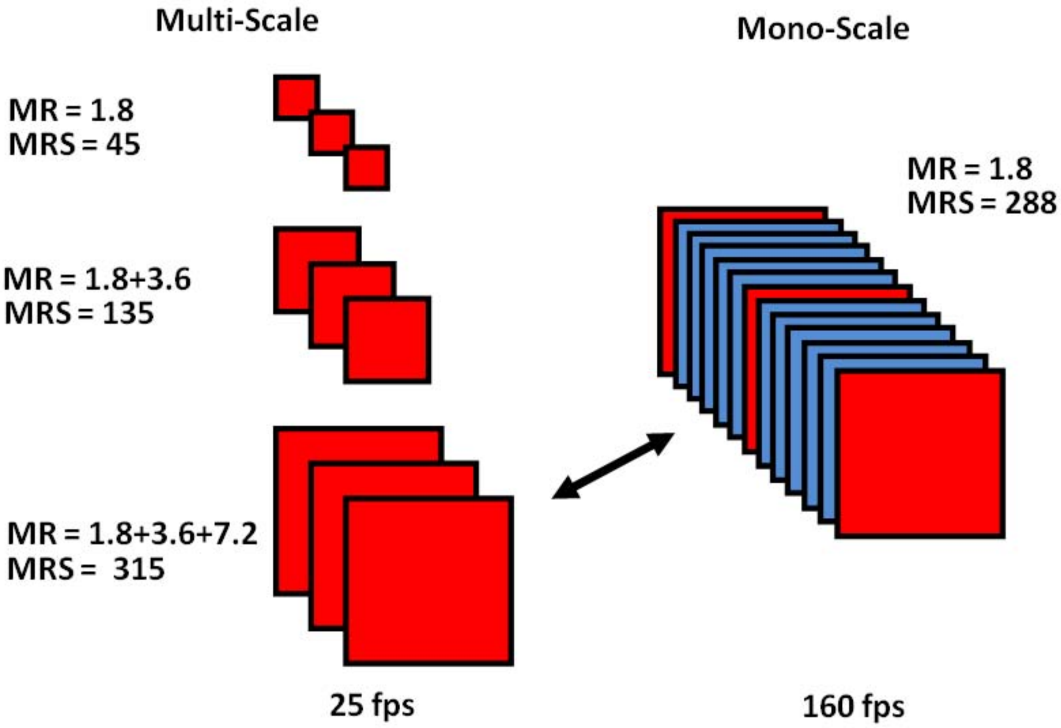


**Figure 2.8: Motion Range for multi-scale methods versus mono-scale ones** - MR and MRS stand for the Motion Range and Motion Range per Second, respectively. Using 3 scales, the MR of a multi-scale implementation is 12.6 pixels but only 1.8 for the mono-scale version. Nevertheless, to properly compare both approaches, we should use MRS that includes the information about the different frame-rates. A mono-scale algorithm working at 160 frames per second is approximately equivalent to a multi-scale one working with 3 scales at 25 fps.

In the case of the stereo algorithm a mono-scale version needs a proper choice of filters to cope with the disparity ranges. Large filters can detect high disparities but at cost of spatial resolution lost. Thus a mono-scale approximation of the disparity algorithm is not viable for unconstrained scenarios and therefore a multi-scale scheme is preferred. Fig. 2.9 shows how the error for the "Tsukuba" sequence decrease with increasing the filter size but it doesn't reach the accuracy level of a hierarchical approach and loses the spatial resolution. At the same time bigger filters require a higher computational effort that grows proportionally with the size of the kernel. On the other hand a multi-scale approach maintains the same computational effort increasing only the number of iterations: the processing takes more time.

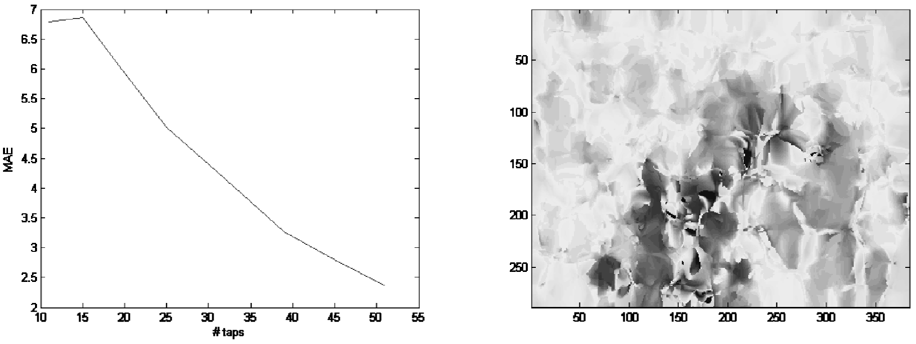


**Figure 2.9: MAE variation for different filter sizes** - Error decreases increasing the size of filters (left). On the right a qualitative result with a 51 taps filter: spatial resolution is clearly affected.

## 2.5 Conclusions

After a comparative study between existing algorithms we can state that the phase-based methods for vision on chip represent good trade-off between accuracy and computational effort. In addition, the harmonic decomposition base of the reference models allows the exploration of many sharing resources approaches as well as the implementation of many low and middle level vision features in a single framework [54]. Note that high performance approaches like variational methods are not hardware friendly and can be unsuitable for an FPGA implementation. A comparative study with other approaches demonstrates quantitatively that phase-based method is robust to scene variations in contrast and brightness, as well as small affine distortions. They are key elements for real-world systems that need to work on unconstrained environments and significantly contribute to choose phase based approach as our target algorithm. After the choice of the candidate for our vision system, different modifications of the original models towards their implementation on hardware are described and evaluated (in terms of their impact on the final error). All changes are simulated with software tools and accuracy loss is measured quantitatively. Algorithm simplifications reduce the system accuracy, final results maintain good quality that allows for utilization in most of the target applications. After these studies we are ready to properly start the design stage of the hardware system. It will be described in next chapters.

# 3

# Stereo and Local Features architecture

## 3.1 Abstract

In this chapter, we present a real-time implementation of a stereo algorithm on FPGA. The approach is a phase-based model that allows computation with sub-pixel accuracy. The algorithm uses a robust multi-scale and multi-orientation method that optimizes the estimation extraction with respect to the local image structure support described in Chapter 2. With respect to the state of the art, our work increases the on-chip power of computation compared to previous approaches in order to obtain a good accuracy of results with a large disparity range. In addition, our approach is specially suited for unconstrained environments applications thanks to the robustness of the phase information, capable of dealing with severe illumination changes and with small affine deformation between the image pair. This work also includes the rectification images circuitry in order to exploit the epipolar constraints on chip and avoid software preprocessing of input images or complex (and low accuracy) manual camera calibrations. The dedicated circuit can rectify and process images of VGA resolution at a frame rate of 57 fps. The implementation uses a fine pipelined method (also with superscalar units) and multiple user defined parameters that lead to a high working frequency and a good adaptability to different scenarios. In the chapter, we present different results and we compare them with state of the art approaches.

## 3.2 Introduction

Stereoscopy has been always a wide faced problem. Current dense stereo techniques are mainly divided into two categories: local approaches and global approaches. As described in Chapter 2 we use a phase-based which belongs to local approaches. The advantage of phase-based approaches has been pointed out in previous sections. Currently, one important goal is the disparity computation on real time with a stable and robust technique able to extract this feature in multiple unconstrained scenarios. Real-time computation leads to its applicability on very diverse scenarios (robotic platforms, driver assistance, etc.). The architecture proposed in this chapter wants to realize a real time system useful also for driving scenarios as proposed in the EU project DRIVSCO [22]. For this reason, we need a good choice for the system implementation capable of working on very difficult illumination scenarios as well as dealing with multiple image artifacts (as the ones caused by rain, foggy days, motion blurred images, etc). With the increasing computational power of machines, software approaches have improved their computational performance. In previous works such as [55], authors achieved a real time stereo system using parallel computation. Current alternatives are GPU implementations [41] that can provide a fast and accurate result. Unfortunately, all these systems require considerable power consumption and their applicability on portable embedded systems is questionable. A good alternative towards on-chip implementation is the use of FPGA based approaches. As FPGA devices progressed both in terms of resources and performance, the latest FPGAs have come to provide "platform" solutions that are easily customizable for system connectivity, DSP, and/or data processing applications. As platform solutions are becoming more and more important, leading FPGA vendors are coming up with easy-to-use design development tools. Carefully designed systems do not loose accuracy as compared to a software implementation and at the same time, feature a high data rate and portability. Further advantages are the low power consumption and reduced size of the whole system as pointed out in the introduction. These important advantages allow a large use in robotic platforms or in an automotive scenario. Previous works proof the validity of FPGAs in stereo computation with very high frame rate [56, 57] but with restricted accuracy compared to other software approaches. Our purpose is to maintain a good accuracy due to the use of a phase-based method and robustness against image artifacts and illumination problems and at the same time, a high data throughput. In [19], a similar implementation of an algorithm in a mono-scale version is described. The problem with this system is the limited disparity range and its strong relationship with the filter size. Our work improves previous works with a multi-scale and multi-orientation approach that allows a very large disparity range and fits properly the image structure thanks to the different oriented filters employed. Our system performs a multi-orientation and a multi-scale technique with a very complex design that is justified in order to reach a robust solution. This approach is a novel hardware system for calculating disparity using a phase-based algorithm with the latest design techniques. The design strategy is based on the construction of deep pipelined data paths composed by heterogeneous computing units and a combination of high level abstraction descriptions as well as RT level ones, in order to keep performance and at the same time keep short the design times of very algorithmic system descriptions.

### 3.2.1 Algorithmic vs. RTL description

In a traditional design flow, crafting the hardware architecture and writing VHDL or Verilog for RTL synthesis requires a considerable effort and time. The code must follow a synthesis standard, meet timing, implement the interface specification, and function correctly. Without time constraints (very unrealistic situation in real-time systems), a design team is capable of meeting all these constraints. However, deadlines imposed by time to market often imposes pressure and forces designers to compromise in area by re-using blocks and IPs that are over-designed for their application. The complexity of our target design leads us to use faster algorithmic hardware description languages (HDLs). As studied in [58], algorithmic languages allow a faster circuit and system definition but achieve slightly less efficient systems than others defined with traditional RTL descriptions using VHDL or Verilog. In order to optimize some critical parts of the design, the system architecture is described in a hybrid form. Generally, physical interfaces such as memory access and external communication represent the bottleneck for hardware architecture due to the inherent sequential operation at the interfaces. Therefore, high clock frequencies and an optimized circuit description are required. Often, processing stages of an architecture have to interact directly with off-chip RAM memory (for temporal results storage) and it reduces speed due to physical constraints of this communication. For this reason, a specific memory controller has been defined in

VHDL in order to abstract the complex memory access in a shared memory scheme and to deliver high-level ports to connect process-stages to physical memory [59]. Bearing in mind multiple accesses on shared memory, we adopt a memory controller architecture with multiple Abstracted Access Ports (AAPs) for accessing to external memory chip which is completely designed in VHDL by M. Vanegas (see 8).

## 3.3 Hardware architecture

The whole design has been implemented for a Xilinx Virtex 4 xcv4fx100 FPGA [3]. The chip includes 94896 configurable logic cells and different embedded recourses: two Power PCs, 160 DSP blocks and 6768 Kb of Block RAM (divided in blocks of 18 Kb). The architecture has been described with a high level hardware description language (Handel C [4]) which optimizes work at an algorithmic description but has proven to be competitive to lower level abstraction levels [58]. Critical stages as the memory controller unit (MCU on Fig. 3.1) or communication with PCI-Express interface have been implemented with VHDL language in order to optimize the performance of these critical elements as previously explained. As described in Chapter 2 and in Fig.3.1, the system could be decomposed in two different parts: pyramid and processing (disparity, merge, median filter, expansion, and warping). The design strategy consists of a fine pipelined circuit which takes full advantage of the high parallelism of FPGA; nevertheless, some parts need a sequential execution due to the iterative multi-scale extension. Loop unrolling of these stages is a valid technique for performance improvement but has the drawback of high latency penalty and therefore, is not feasible for our target applications. This section focuses on different circuit stages explaining this hybrid (parallel-sequential) structure.

### 3.3.1 Rectification and image pyramid

In the first stage, rectification consists in a bilinear interpolation of input images with a LUT of $\Delta$ values previously calculated off line for the camera system configuration. For calculating each rectified pixel it is necessary to read from the x-matrix and y-matrix the pair (x; y) that corresponds to each point P. The integer part of (x; y) is used for retrieving from memory the four pixels of the original image that together with the fractional part of (x; y) produce the rectified pixel with a weighted bilinear

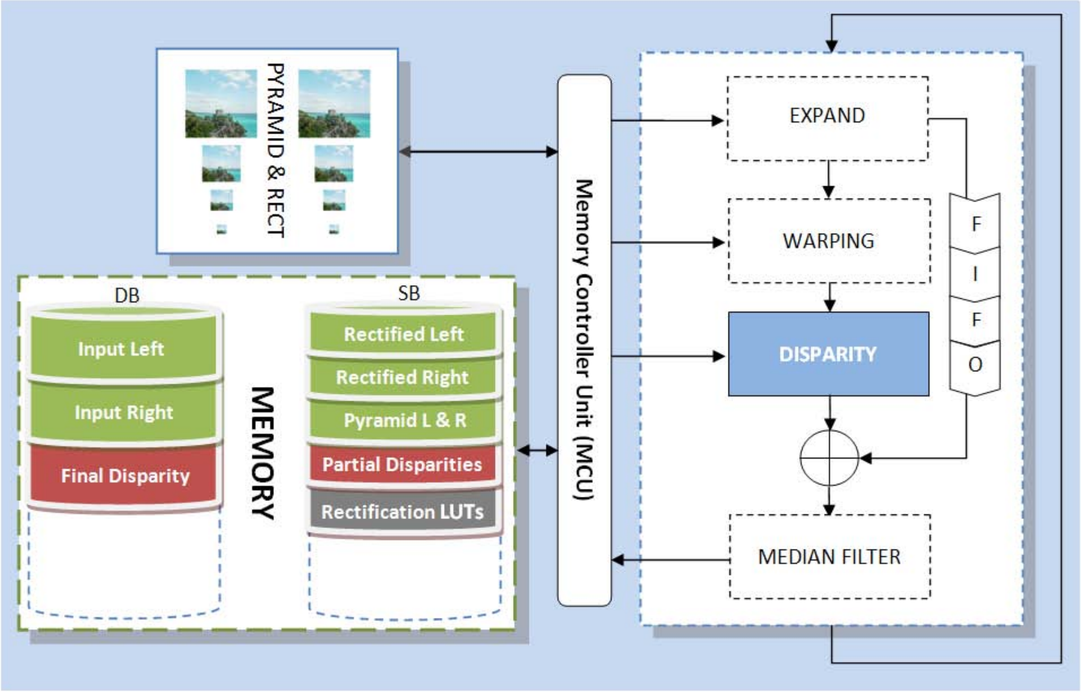


**Figure 3.1: Multi-scale architecture of the complete stereo system.** - The design is divided in two sequential basic steps: the pyramid stage (on the left upper part) and the processing loop (right block). Memory banks are organized in two different kinds: two Double Buffer (DB) banks and a Stereo Buffer (SB) bank. All directions have 36 bits; soft gray data have 8 bits per pixel and are stored in groups of four, hard gray data have 12 bit per pixel and are stored in pairs (PCI interface uses only 32 bits), middle gray data have 12 bits and are stored in groups of three.

interpolation. The rectification process needs to do four memory accesses per clock cycle for calculating one warped pixel in order to achieve the maximum throughput. This is another reasons for the choice of the specific memory controller unit (MCU) created by Mauricio Vanegas [59] that manages data with different Abstract Access Ports (AAP). This critical architecture is optimized in VHDL and described in [59]. The rectification architecture uses a reading AAP of the MCU for access to the original image. In total, two reading-AAPs are used by the two different rectification blocks: left and right image. The MCU provides 36-bits of bus-length allowing a four pixels per memory access in the case of image data. X-matrix and y-matrix are provided from the expand circuit through two blocking FIFO. Bearing in mind that the rectification needs a neighborhood of four pixels and that the number of data available per memory access is limited to four in a same line, it is evident that in the best case, one access to image data brings two pixels of the same line; nevertheless, it is impossible to access the four pixel window in a single memory access. In the worst case, we access four different memory words, this occurs each four consecutive memory accesses on image data. This hardware constraint restricts the performance up to 4 pixels each 10 memory accesses. After rectification, the pyramid is built by a smoothing and down-sampling circuit (see Fig. 3.2). Left and right circuits are replicated and work in parallel. Each pyramid scale is obtained sequentially one after the other (mainly, due to limitations of the sequential access to the external memory). Rectification and first image reduction are executed in parallel, input and output images are directly read/stored into an external RAM memory. The main operations at this step are the bilinear interpolation locally calculated in 2 by 2 windows for the rectification and a 2D convolution with a low pass Gaussian filter of 5 taps before every down-sampling in order to smooth input images. The kernel is a 5 by 5 matrix decomposed in two vectors like:

$$K = [1\ 4\ 6\ 4\ 1]/16 \tag{3.1}$$

Thus, convolution with the input image is separated into x and y operation in order to benefit the FPGA parallelism. Five different image lines are stored in an embedded multi-port BlockRAM which is used like a FIFO. After the pre-processing, we send to output (external SRAM) a pixel every two clock cycles: one pixel is discarded (sub-sampling).

### 3.3.2 Stereo core

Stage two (right part of Fig. 3.1) starts after the first one and is sequentially repeated for every scale. Disparity, merge, expansion, warping, and median filter circuits work all in parallel. For the smallest scale, the merge circuit is omitted and the disparity block works directly on the pyramid output. Disparity calculation, as described in Section 2.1.2, is divided in three main steps and benefits from a fine pipelined design. It uses two (left and right) Gabor filters of 11 taps, 7 atan2 cores with CORDIC core [3, 60] for calculating (2.5) and a simple median circuit for choosing final disparity between 7 different orientation-based estimations. On Fig. 3.3, we describe these three different steps with specification of pipeline stages. The filtering part also takes another 4 (half

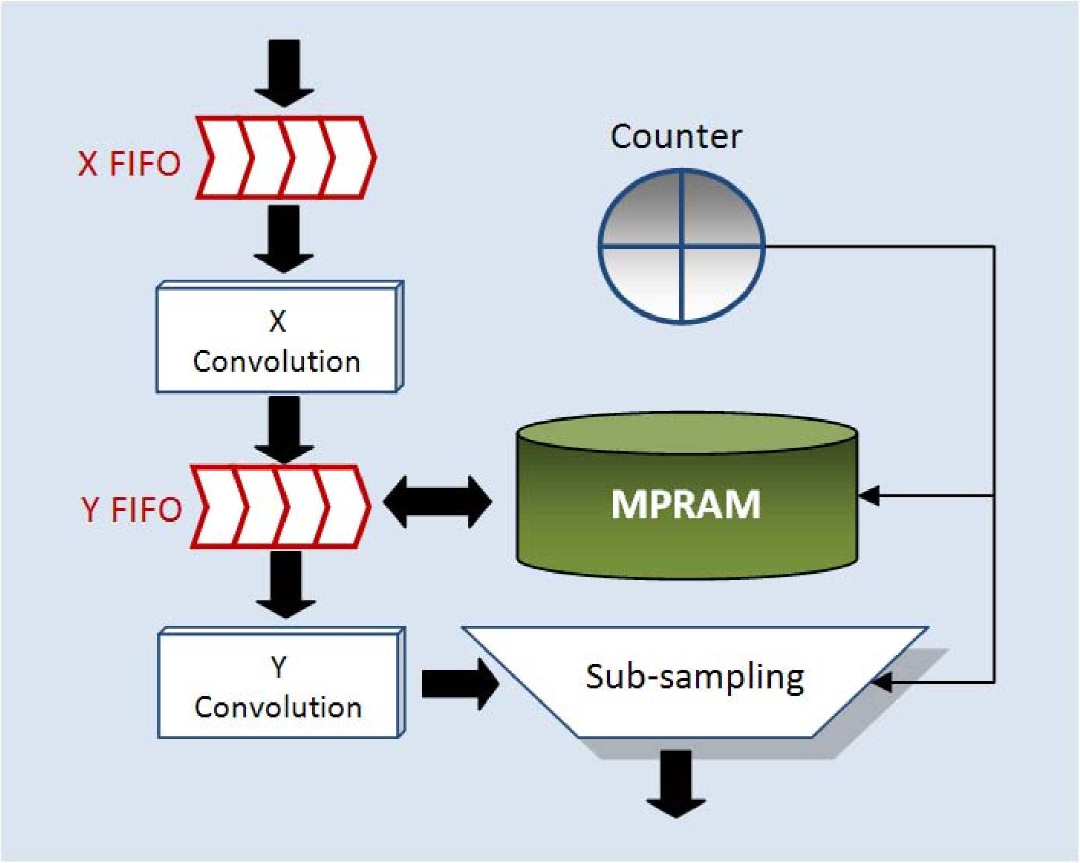


**Figure 3.2: Circuit architecture for the image reduction.** - This circuit is required for building the image pyramid and is used sequentially along scales.

filter size) image line cycles to fill temporary FIFO convolution, which this time will be added as latency to the total disparity computation.

### 3.3.3 Multi-scale architecture

The following merging circuit is simply the result of adding old and new disparity values provided respectively from a FIFO and from the disparity circuit (Fig. 3.1). Non valid values at the smallest scales are discarded; only in the last step we consider also the invalid values of the finest scale. The rest of blocks in the processing stages interact with memory trough a memory controller unit that multiplexes in time the huge amount of data to read/store [59]. In detail, the memory data flow is displayed in Fig. 3.1 and can be summarized as follows:

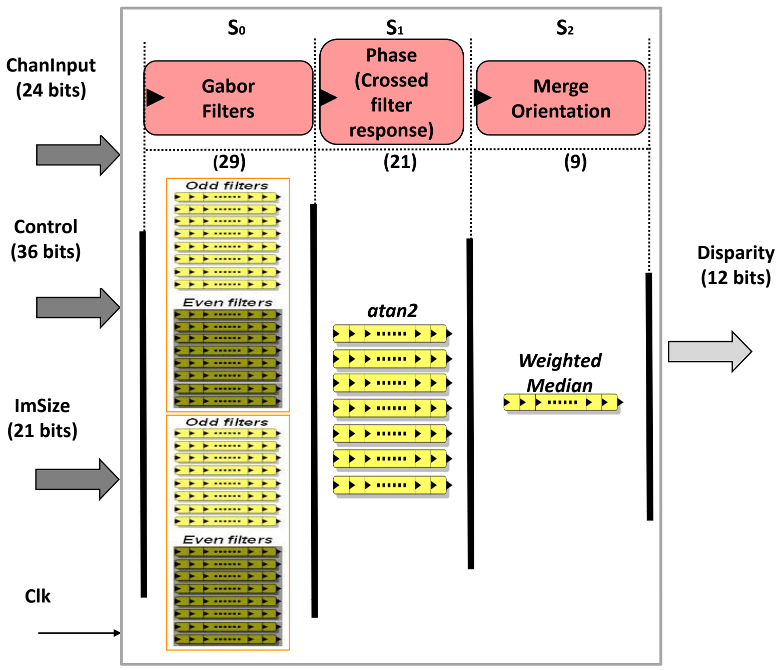


**Figure 3.3: Pipeline stages for a mono-scale system.** - Filters need four more lines of latency to fill half blockRAM of convolution (at the beginning of the computation).

1. Expand circuit, read old partial disparity and up-sample it with a bilinear interpolation (new values are multiplied by 2 to adapt disparity values to the new scale).

2. Warping circuit, read pyramid images and shift those using expanded disparity as $\Delta$ LUT.

3. Median filter stores the partial/final result.

As we can see, the interaction with memory is a very critical problem that needs a dedicated circuit and a specific memory mapping (Fig. 3.1). Parallel accesses to RAM are allowed with a multiple bank strategy and sequential operations.

### 3.3.4 1-D warping for disparity range extension

In order to get a further reduction of memory accesses, the warping architecture has also been optimized with the use of embedded memory (on-chip memory resources). The nature of stereo algorithms is to compute distances between positions of corresponding points only along the direction of the epipolar lines (x direction) and therefore, we do not need a random memory access of the whole image at every cycle, but only a partial access to some specific lines; thus, we use a multi-port embedded RAM as cache to store two input lines and obtain in this way a fast access to pixel and disparity (LUT) values. Values are continuously refreshed as in a circular FIFO buffer. A double buffer technique is used to operate multiple accesses and optimize data throughput. We have a line of latency necessary to store the first values. Processing takes 7 clock cycles more of latency for the search of the new pixel and the bilinear interpolation. Due to the fine grain pipeline, the circuit can process a pixel every clock cycle at a maximum frequency of 55 MHz (Table 3.1).

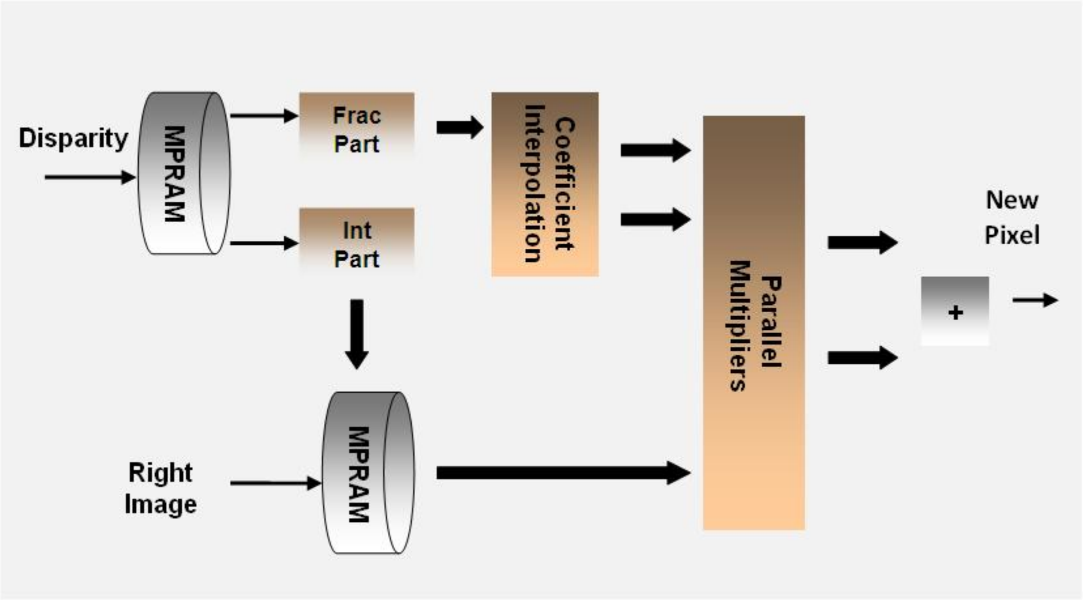


**Figure 3.4: Warping architecture for the stereo case.** - The circuit processes a pixel every clock cycle as input data is previously stored in a multi-port RAM.

On Fig. 3.4, we present the basic architecture of the warping block for the stereo case. MPRAM memories are embedded in the Virtex 4 and they do not use any logic

**Table 3.1:** HW resources for different system parts (functional blocks) in an xcv4fx100 device.

| Circuit | Total 4 input LUTs | Slice Flip Flops | Slices | DSP | Block RAM | Freq. |
|---|---|---|---|---|---|---|
| | (out of 84352) | (out of 84352) | (out of 42176) | (out of 160) | (out of 376) | (MHz) |
| Board interface, undistorsion and rectification | 9943 (11%) | 9097 (10 %) | 9894 (23 %) | 32 (20 %) | 43 (11 %) | 50.8 |
| Mono-scale disparity (sharing) | 19835 (23 %) | 10421 (12 %) | 13783 (32 %) | 115 (71 %) | 20 (5 %) | 42.2 |
| Mono-scale disparity (no sharing) | 30238 (35 %) | 11882 (14 %) | 17975 (42 %) | 97 (60 %) | 20 (5 %) | 52.6 |
| Mono-scale disparity (sharing) + Local features | 33459 (39 %) | 20011 (23 %) | 20789 (49 %) | 115 (71 %) | 20 (5 %) | 41.6 |
| Total system | 58374 (69 %) | 30273 (35 %) | 35715 (84 %) | 131 (81 %) | 99 (26 %) | 50.5 |
| Total system sharing | 47971 (56 %) | 28824 (34 %) | 31169 (73 %) | 149 (93 %) | 99 (26 %) | 41 |

resources. Parallel multipliers are also implemented in embedded DSPs. The other operations are simply shifts and bit selection to separate integer from fractional part in the LUT values; finally we have an adder which sums the interpolated values to obtain the final average value (a further shift operation).

### 3.3.5 Implementation and hardware utilization

According to the bit width choice and architecture described in Section 3.3, we synthesize different circuit parts to estimate the hardware consumption for each functional block. On Table 3.1, we report hardware consumption for an xc4vfx100 Virtex 4 chip. For the whole system, we have to replicate filter consumption by 2 (left and right image), atan2 core by 7 (orientation number minus the vertical one); being total occupation reported on the last rows. In order to reduce hardware consumption, it is possible to operate in different ways. First of all, we can reduce bit depth with a consequent loss of precision already described in Section 2.2.2. Another possible way to avoid precision loss is sharing recourses. On Fig. 3.3, we can see how some blocks are repeated in the architecture; the idea is to share a block by multiplexing input data in time. Sharing the same circuit for two different processes can save up to 50% of hardware resources for this processing unit, but it restricts the final system data throughput: one data for every two clock cycles. Possible targets for sharing recourses are the pre-processing filters and cross phase blocks; in the first case, the same kind of filter is repeated for the left image and right image for filtering, the most expensive operation can be shared for the image pair. In Fig. 3.3, it is also evident the repetition in the second step of mono scale block disparity: seven different atan2 cores are used for computing phase (one for each orientation). In this case, we can use only 4 cores and share them for all seven processes. With these two optimizations, the system requires 13% less in available LUTs and 10% less in slices. We can also move on the other side of optimization and dedicate larger hardware effort to achieve higher accuracy. A previous study on software states that some kind of regularization at the end of each scale iteration can drop a lot of erroneous values and get a more homogeneous output. Our process is a simple bi-dimensional median filter. We apply regularization directly after merging output; median filter is operated on a 3x3 window and discards non valid numbers depending on a user defined threshold: if non valid values in a window are higher than the threshold median output, it will be a non valid number; otherwise, the block provides as output the median value among valid numbers. This precision oriented change increases requires 2% more LUTs and 4% more slices with respect to the non sharing system. On Section 3.4, we discuss the qualitative and quantitative improvements.

### 3.3.6 Local features integration

As described in [54] the phase-based method presents high analogies for multiple features extraction. In particular is possible share the harmonic representation and the first filtering stage for a further local features extraction [54]. As previously explained in Section 2.2.3 we adapt some of the algorithms described in [45] to a hardware friendly implementation. In the same time we integrate it in the mono-scale disparity core in order to share the onerous filtering stage and save hardware resources. In this way, for each processing scale we obtain a local estimation of energy, phase and orientation a part from the disparity. Local energy is directly extracted from the phase difference thresholding of the stereo computation: in fact energy is already obtained for each orientation in the second stage of Fig. 3.3 as a confidence measure. Local orientation is obtained also starting from magnitudes calculated in phase difference block. After a first multiplication by cosine and sine value (respectively with squared even and odd part of filter output), orientations are merged with a summation and final orientation is given by an arctangent module. Local phase is simply calculated as the output of an arctangent of the summation of even filter responses and odd filter responses. This further processing generates three output channel more in the disparity core of Fig. 3.3. New data output are stored directly in external RAMs in three different memory addresses and with words of 32 bits: four contiguous pixel informations of 8 bits in each address. The hardware utilization of the complete core with local features integration is reported in Table 3.1. It is possible to integrate the local features core also in a phase-based optical flow processing but as we will see in the Chapter 4, the huge quantity of memory accesses for this computation limits any further data transfer.

## 3.4 System results

The final circuit is optimized and set up to run on a XircaV4 board produced by Seven Solutions [61]. Although in this work the platform is used as co-processing board, the great advantage is the possibility of using it in a stand-alone mode for robotic issues and for driving scenarios. We have implemented a hardware/software platform to work as an interface between the external world (sequences captured from on-line cameras) and the FPGA platform. It provides the input images and shows the results of the disparity processing on-line. The whole system consists of a co-processing FPGA board and a host computer connected through the PCI Express interface (for further details see Appendix 8). The communication using the PCI Express interface is performed with a simple handshaking protocol. The double buffer scheme optimizes the system speed and regularizes the read and write operations on external SDRAM banks. For each couple of frames, the application and the FPGA board wait for each other to commute the memory bank. The application writes the input image in the first memory segment (reserved for the input images in the memory map, Fig. 3.1) or reads the previous results. FPGA manages the bank memory map, allocates a bank for its processing and commutes the other bank to the application. This software (created by Francisco Barranco) is available as Open Source at `http://code.google.com/p/open-rtvision/`

### 3.4.1 System accuracy

In order to evaluate our system, we have used some benchmarking images. This allows to compare the results with other approaches and with ground truth if it is available. Unfortunately, for the majority of road sequences there is not ground truth, so results can be evaluated only qualitatively. We have used a software version (with double precision floating point arithmetic) to compare the results. We have used the previously described software interface to process stored sequences in different approaches. For

**Table 3.2:** Mean Absolute Error (MAE), standard deviation, density, and percentage of error major than 1 for different image pairs

| Image | MAE | SAE | Dens | Err≥1 |
|---|---|---|---|---|
| Tsukuba | 0.84 | 1.44 | 91.72 | 18.8 |
| Venus | 0.74 | 1.38 | 86.96 | 13.1 |
| Cones | 2.52 | 6.25 | 85.4 | 26.8 |
| Teddy | 2.97 | 5.89 | 68.22 | 36.8 |
| Monopoly (E0-E0) | 1.05 | 1.94 | 31.64 | 18.4 |
| Monopoly (E2-E2) | 1.73 | 4.5 | 31.64 | 23.9 |
| Monopoly (E0-E1) | 1.02 | 1.86 | 31.64 | 18.6 |

error estimation, we use MAE (Mean Absolute Error), SAE (Standard deviation of Absolute Error), density, and percentage of error major than one as described in [15]. We have processed different image pairs downloaded from the Middlebury database [2]; processed disparities are compared with ground truth for a multi-resolution computation which changes depending on image resolution. Quantitative results are provided in Table 3.2. Comparing the system with other approaches in the literature [2], we obtain a lower accuracy of results, but this is still high enough for our target application on the driving scenario [22] and they are good taking into account the constraints of a hardware implementation on a embedded device. In addition, it is relevant to remember that the model is especially robust and stable for real world unconstrained scenarios (Fig. 3.5) which is of crucial importance for outdoor-scenario applications. In fact the robustness of the phase signal allows the same accuracy under different illuminations. The same sequence has been processed with different conditions available from [2], moreover left and right frames has been chosen with different illumination. Results maintain a good accuracy level also in very complicated conditions (fourth row of Fig. 3.5).

### 3.4.2 Performance analysis

Our hardware implementation is optimized for the Xirca platform [61] but FPGA external interface with memory and PCI interface can be easily adapted for other kind of board and system maintaining the power of processing cores. On the other hand, if we have fewer resources on the target platform, the solution is dropping frequency and sharing resources as described in Section 3.3.

For the specific platform, we have used a third additional memory bank as stereo buffer (SB) just to obtain parallel access to memory. Memory mapping is generated from FPGA at initialization time and depends on the image size (user defined parameter). Memory space is organized as indicated in Fig. 3.1, DB bank is only used for input and output images, while rectified images, pyramid, partial disparities, and rectification LUTs are stored in SB. Thus, the operation sequence described in this section writes at time t the correspondent output image in DB. All circuits work at a data rate of one pixel per clock cycle. For this reason, processing time takes a number of clock cycles equal to:

$$c = (RI + sI) + (s + 1)I = I(2s + 1 + R) \tag{3.2}$$

where R is the inverse of data rates for the rectification circuit and is R=2,5; I is the image resolution (x size multiplied by y size) and s is the scale factor and depends on the number of scales as indicated in (3.3).

$$s = \sum_{n=1}^{N} 2^{-2n} \tag{3.3}$$

Where N is the number of scales. With (3.2) and (3.3), we can calculate the frame rate for different image sizes simply by dividing the frequency of the circuit by C. We discuss now some speed up for the processing system in a specific board with at least four memory banks. For the described architecture, we are using now only 3 banks, so we can implement a second double buffer for SB in order to run in parallel rectification/pyramid with disparity calculation. (3.2) becomes now:

$$c = \max(\{RI + sI\}, \{(s + 1)I\}) \tag{3.4}$$

The problem for this new architecture is the frame of latency, in fact, at time t, FPGA writes on DB bank output image for time t-1.

For the three different versions of our architecture and for embedded purposes we also analyzed power consumption with the Xpower tool and report the results in Table 3.3.

On Table 3.4, we provide the final frame rates for some usual image resolution and for the faster system architecture described: using 4 memory banks and without any sharing. On the other hand, if the target platform includes larger hardware

**Table 3.3:** Power consumption for stereo circuits in a Virtex 4 xc4vfx100: estimated with Xpower tool.

| System | Power (W) |
|---|---|
| Total System (sharing) | 3.64 |
| Total System (not sharing) | 3.78 |
| Total System (Gabor filters sharing) | 4.34 |

resources and more memory banks available, the design can take full advantage of a multi-core implementation. It is important to take into account that the system is a real-time processing module which is faster than most existing systems, see for instance [62, 63]. It is important to remark that, except for the mono-scale phase-based version presented in [19], our approach overcomes the performance of all the other available contributions. The image rectification stage together with the coarse-to-fine nature of the algorithm limits the frame rate performance, but as we indicated in the previous section, provides reliable results especially in non controlled environments and no image preprocessing is required. The mono-scale phase-based implementation such as [19] achieves a higher frame rate but is not capable of solving high range disparities and many times produces oversmoothed results with reduced spatial localization capabilities. Note that our approach achieves a higher Point × Disparity per Second (PDS). This measure is commonly used in the literature as performance metric (see for instance [16, 19, 21, 56, 63]) and evaluates the throughput and the disparity range at the same time. It is the multiplication of the frame rate per the image resolution per the disparity range. In literature, we can also find a multi-resolution architecture [35] that simply computes disparity for different scales in parallel but does not propagate information among them and as a consequence, they reduce the hardware resource requirements. It is worthwhile indicating that the warping operations have a very high complexity mainly due to the non-deterministic data access scheme (this is why they are rarely implemented in the literature) but they are justified for the accuracy improvements. In Table 3.4, we compare our best implementation with some works in the state of the art. Obviously, this solution is the most expensive configuration and can be replaced with a more economic one based on hardware sharing depending on the final application and the platform constraints. We achieve a very high processing speed (17.6 Megapixels per second) which is the second fastest approach in the table.

**Table 3.4:** Comparison with other approaches described in the literature. We indicate the throughput in MegaPixels per Second (MPPS) and the Point × Disparity per Second (PDS=MPPS X D)

| Work | Image resolution | Frame rate | MPPS | PDS x $10^6$ (Disp. range) | Method | Processor type |
|---|---|---|---|---|---|---|
| Presented work | 640x480 | 57.5 | 17.6 | 2252 (128) | Phase-based | Custom FPGA, Virtex 4 (50 MHz) |
| | 800x600 | 36.8 | 17.6 | 4505 (256) | | |
| | 1024x768 | 22.4 | 17.6 | 4505 (256) | | |
| Sang-Kyo Han et al. [21] | 320x240 | 140 | 10.7 | 707 (64) | SAD | ASIC at 150 MHz |
| Gibson et al. [64] | 450x375 | 5.9 | 0.99 | 63.7 (64) | Semi global matching | NVIDIA G80 GPU |
| Li [65] | 640x480 | 31.2 | 9.58 | 2875 (300) | Spherica | Sony PC VGNK704 with a 2.8 GHz Processor |
| Diaz et al. [19] | 1280x960 | 52 | 63.89 | 1885 (29) | Phase-based | Custom FPGA Virtex 2 (65 MHz) |
| Murphy et al. [66] | 320x240 | 150 | 11.52 | 230.4 (20) | Census transform | Custom FPGA Spartan 3 (26 MHz) |
| Gong et al. [63] | 384x288 | 11.3 | 1.2 | 30-60 | GORDP | ATI Radeon X800 |
| Wang et al. [62] | 320x240 | 43 | 3.3 | 52.8 (16) | Dynamic prog. | 3.0 GHz PC with an ATI Radeon XL1800 |
| Darabiha et al. [35] | 360x256 | 33 | 3 | 55.2 (20) | Phase-based | Custom FPGA, 4x Xilinx Virtex 2 |
| Masrani et al. [56] | 640x480 | 30 | 9.21 | 1179 (128) | Local weighted phase correlation | Custom FPGA, 4x Altera Stratix S80 |
| Woetzel and Koch [67] | 704x576 | 3.57 | 1.44 | 28.9 (20) | TSSD | P4 with a NVIDIA GeForce FX5600 |
| Miyajima et al. [68] | 640x480 | 20 | 6.14 | 1228 (200) | Matching | Custom FPGA Virtex 2 (40 MHz) |

But the most important comparative result is that we obtain 4505x$10^6$ PDS which is the best comparative mark (more than 35% better than the second best approach) of the table and takes into account the large disparity range achievable by our multi-scale approach. Finally note that if the undistorsion and rectification stage is not used (it is the case in most of the literature solutions), the processing speed achieves up to 35.7 MPPS.

## 3.5 Conclusions

This chapter describes a stereo system of high complexity and performance implemented on a reconfigurable device. We have improved existing FPGA approaches with a higher accuracy and a larger disparity range thanks to an optimized data bit-width utilization and a coarse-to-fine multi-scale processing scheme. The final system can process video sequences at a high frame rate (for instance SVGA resolution at a frame rate of 36 fps), achieving real time for large resolution images. A comparison with the literature asserts that our approach is among the fastest approaches in terms of pixels per second and the best approach in terms of PDS (i.e. taking into account the disparity range covered). Furthermore, it is an on-chip approach which is very well ranked among other approaches addressed with diverse technologies. Since we use reconfigurable hardware, we can exhaustively adapt the architecture to different chip sizes and application domains. Thanks to a sharing strategy, the same algorithm can be synthesized for different devices and platforms. This sharing strategy affects the system data throughput but not its accuracy. The described stereo system is robust against illumination variations between the two cameras and local contrast differences since it is based on phase and uses multi-orientation estimations to better optimize accuracy for different local contrast structures. This feature is obtained by the multi-orientation phase-based stereo model. The outstanding computing power of the system in terms of Megapixels per second has been achieved by adopting a fine grained pipelined processing data-path with superscalar units at several stages. This represents a global circuit with more than 2000 processing elements working in parallel. Such a complex design has been carried out by adopting a modular design strategy. We have validated the stereo processing engine in the framework of a co-processor solution that uses a software interface as frame-grabber and real-time visualization tool. The same board can be used to implement a stand-alone system and work on embedded applications with reduced occupation in space and low power consumption compared to other approaches as high performance processors, GPUs, etc. This makes the presented processing engine of specific interest for a wide variety of application fields, which include robotic issues, driving scenarios, mobile video surveillance, etc. As future work, we will deal with the stand-alone version of the system, developing an Ethernet-based frame grabber with a direct connection of cameras on this interface. This structure would be adapted for a car system and integrated with feedback signals from/to the vehicle and the driver.

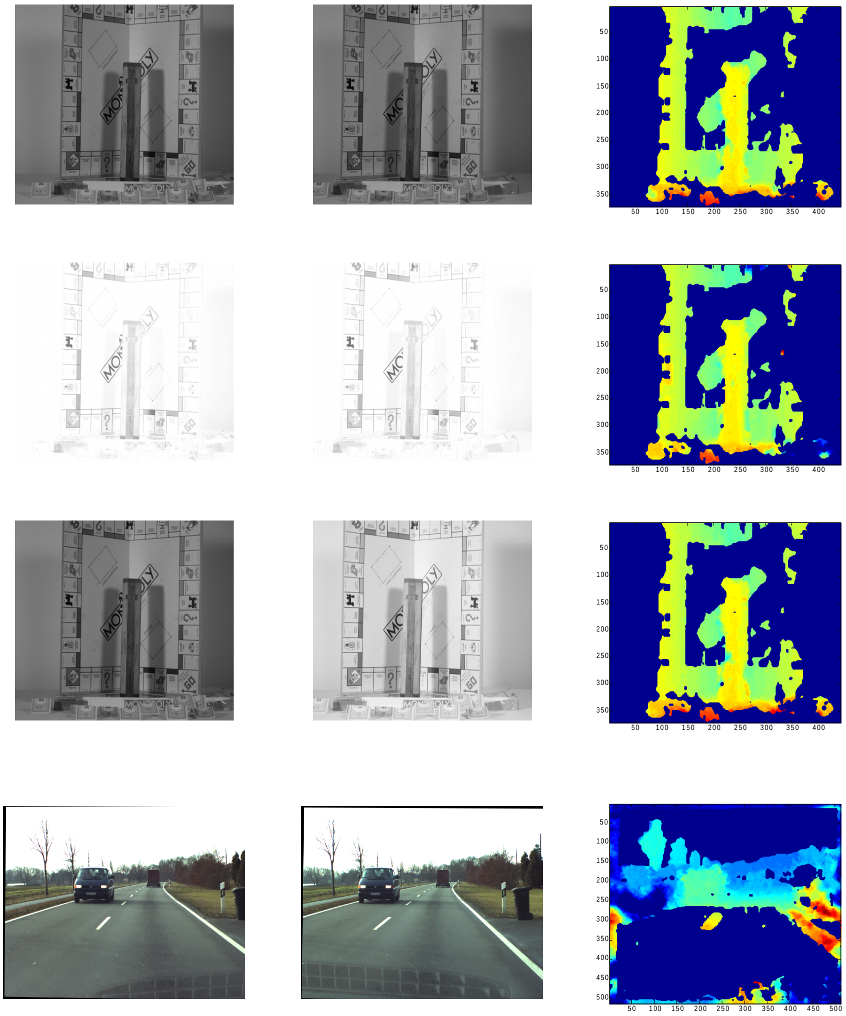


**Figure 3.5: Qualitative results of the stereo system.** - Left image input (first column), right image input (center) and disparity results are displayed. The three first rows shows the results for the "Monopoly" stereo pair with different expositions. Special consideration should be made for the third row where two different expositions have been used for the left and the right images. In the last row is presented a result for a real driving scenario.

# 4

# Optical Flow architecture

## 4.1 Abstract

The accurate estimation of optical flow is a problem widely experienced in computer vision and researchers in this field are devoting their efforts to formulating reliable and robust algorithms for real life applications. These approaches need to be evaluated, especially in controlled scenarios. We describe here the implementation of a phase-based optical flow in a FPGA device. The system benefits from phase-information stability as well as sub-pixel accuracy without requiring additional computations and at the same time achieves high-performance computation by taking full advantage of the parallel processing resources of FPGA devices. Furthermore, the architecture extends the implementation to a multi-resolution and multi-orientation implementation, which enhances its accuracy and covers a wide range of detected velocities. Deep pipelined datapath architecture with superscalar computing units at different stages allows real-time processing beyond VGA image resolution. The final circuit is of significant complexity and useful for a wide range of fields requiring portable optical-flow processing engines.

## 4.2 Introduction

Beginning in the early 1980s, with important contributions such as that provided by Lucas and Kanade [37], the study of optical flow has generated high interest in computer-vision research and the potential multiplicity of its applications. Since then diverse optical-flow algorithms have been developed and many studies published concerning motion estimation [1, 23, 38, 69]. Motion perception is useful in very diverse real-life environments. Some potential fields of application are video surveillance [70], autonomous robot navigation [71], driving assistance [72], sports analysis (http://www.spinsight.co.uk/index.htm) [73, 74] and so on. Furthermore, current global computer-vision approaches include motion estimation as one of the low-level cues used for higher level vision stages and for the extraction of more complex features such as independent moving objects (IMOs) [75] or motion in depth [34]. For all of these applications we obviously need a real-time processing engine, a problem that has been partially solved as far as modern commodity processors are concerned [13] but remains an open issue for embedded applications in which computing resources are constrained. To avoid this problem some implementations simplify the model to increase processing speed; for instance, mono-scale versions are often used [20], but they only work accurately over a restricted working speed and furthermore, this also limits their adaptability to different scenarios. This has motivated the search for high-performance customized computing architectures with a high level of parallelization in order to achieve real-time processing and robustness to deal with input instability due to limited scene-contrast structure. Within this context we have proposed different solutions that represent valid technologies which can be used to implement high-performance approaches. Thus it is possible to take full advantage of commodity processor parallelism [13] or use some specific co-processor devices such as GPU boards [41] or a cluster of processors to achieve real-time [24]. These options adapt well to different environments that require simulation accelerators, but are unsuitable for many industrial products that demand reduced power consumption, size, and price. In this case a better alternative is offered by processing architectures customized for embedded systems (based on FPGA devices for instance) [20, 46]. We describe here the implementation of a phase-based multi-scale algorithm in a FPGA platform, the optical-flow model of which is described in [40]. Phase information allows us to reach the desired stability, as described and justified in [29]; furthermore, with reconfigurable hardware capabilities and on-chip high parallelism we achieve real-time processing. The final system described here can process images very accurately at a resolution of 512 x 512 up to 36 fps over a wide speed range.

## 4.3 Implementation on FPGA

We have implemented the algorithm described above using different hardware description languages. Critical parts of the design, such as memory interfaces [memory control unit (MCU) designed by M. Vanegas [59] in Figure 4.4] and PCI-express interface with FPGA, are described in VHDL language. For the optical-flow algorithm, on the other hand, we use an algorithmic hardware description language (Handel-C) because, as pointed out in [58], this abstraction level does not affect hardware requirements or performance significantly and the design process is faster than with VHDL. According to Ortigosa et al. [58] the same algorithm can be described using less development time with a C-like language. Furthermore, we used IP cores from Xilinx CoreGenerator [3] to optimize such complex mathematical operations as divisions, trigonometric functions etc. The whole circuit is synthesized for a Xilinx Virtex 4 xc4vfx100. The device used includes 94,896 configurable logic cells, 2 embedded PowerPCs that work up to 300MHz, 160 embedded DSP and 6,768 Kb of embedded Block RAM memory (each elementary block has 18 kb). Although the FPGA includes PowerPCs we do not make use of them in this specific low-level vision algorithm: such an approach needs massive parallel processing, which is better suited for FPGA logic resources. An additional middle-level vision processing can run in embedded processors, as stated in [76] but this issue is not addressed in this work.

## 4.4 System architecture

For such a complex system we chose to adopt a modular design strategy based on a long pipelined datapath capable of processing one pixel every clock cycle. Nevertheless, there are iterative stages (due to the multi-scale approach) and because of the limited hardware resources available we need some sequential parts that communicate with the other components through an external RAM. In this way, memory access increases and the implementation of a MCU is essential. In the architecture we have adopted a specific MCU that manages memory access by using abstract access ports (AAPs) as described in Appendix 8. These allow us to handle the different memory modules as multi-port ones with as many ports as the number of configured AAPs. Arbitration circuitry distributes access equally so as to optimize memory bandwidth. The AAPs save the effort of dealing with problems related to memory-access scheduling, thus increasing memory-use bandwidth and helping to reduce coding times significantly. As can be seen in Figure 4.1, the MCU synchronizes the following blocks:

1. Pyramidal calculation of input frames
2. Optical-flow processing for each scale: the main processing part.

Both stages are repeated as often as the scale number: the pyramid stage is a top-down implementation and the processing of the optical flow is a bottom-up one.

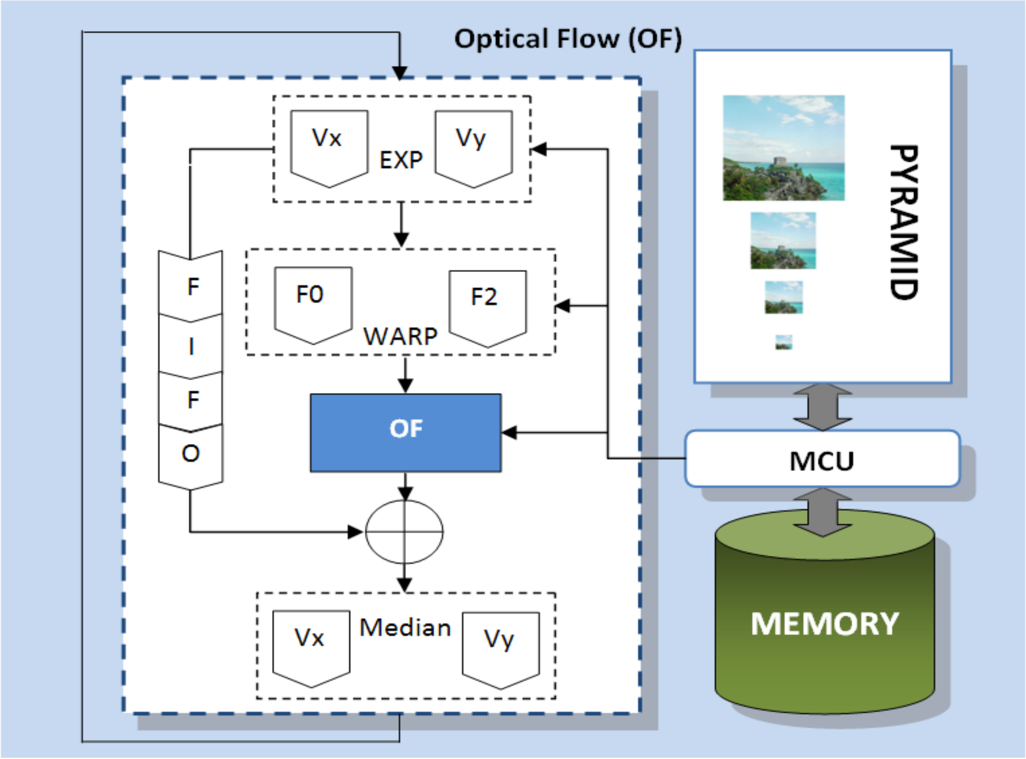


**Figure 4.1: Optical flow system architecture.** - The pyramid iteration appears on the upper right-hand side, and the optical-flow processing stage is on the left-hand side. Both stages communicate with memory through the MCU.

Stage 1 finishes just before the iteration in Stage 2, executing an anti-aliasing filter and a sub-sampling of input frames. The anti-aliasing filter consists of a Gaussian smoothing with a 5x5 kernel. The sub-sampling circuit reads frames from an external memory and stores them in an embedded memory (circular buffer) for convolution with the kernel; this takes 2 lines of latency. Output values are sub-sampled and stored on the same external memory in another location next to the previous one. Stage 2 includes the five blocks described in Section 2.1.1: expansion circuit, warping circuitry, optical-flow computation core, merge core and a median filter. The first block expands previously computed optical-flow values to adapt them to the new image size, i.e. it enlarges the image by a factor of two using bilinear interpolation. On output, this block multiplies values by 2 and also adapts velocity vectors to the new scale. In following subsections we focus on the warping circuit and on the mono-scale optical-flow core of Fig. 4.2.

### 4.4.1 Warping module

There are two different warping circuits, one for the previous and one for the future frame; the current (middle) frame is not warped. Warping adds the motion estimation obtained from the previous scale in order to reduce the movement range in the input frames and adapt it to the filter size of the optical-flow core. This block takes the new pixel and operates a bilinear interpolation with neighboring pixels in a 2x2 sized window. The motion compensation (warping) operation in the multi-scale algorithm is a problematic issue for an FPGA circuit. This kind of devices are in fact well suited for local memory accesses but present some difficulty with random memory accesses. The neighborhood of four pixels and the limited memory accesses (four pixels in the same line) make evident that in the best case one access to image data brings two pixels of the same line; nevertheless it is impossible to access to the four pixels window in a single memory access. In the worst case, this operation needs four different accesses to memory. This is another reason for the development of the specific MCU [59]. The warping architecture for optical flow uses a reading AAP of the MCU for access to the original image. In total, two reading-AAPs are used by the two different warping blocks: previous and future frame of the temporal sequence. The sequential accesses restrict the performance down to 4 pixels each 10 memory accesses. A similar circuit is used for rectification and undistortion in the stereo circuit.

### 4.4.2 Filtering stage

The first operation for the processing is the spatial filtering with quadrature pair filters. This part of the circuit is further divided in two steps. The first one operates the convolution with seven different kernel masks: three of them that represent the Gaussian basis function and four of them, the Hilbert transforms as described in [42]. Each kernel convolution mask has 9 different taps that are previously hardware wired. Due to the symmetry (or asymmetry) of basic functions, we use only five of them and define for the convolution circuit a flag for the different case: a value 1 means a symmetric vector while a -1 represents an asymmetric one. Multiplication and sum for convolution are optimized with the embedded resources of the FPGA: DSP48 for a Xilinx Virtex4 and MULT18x18 for a Virtex2 [3]. The second stage of this block makes a linear combination of the convolution outputs for each of the 8 orientations using trigonometric polynomials up to second order. This operation steers the filters and builds the quadrature filters for the different orientations. Again, values of the polynomial are hardware wired. The convolution operations use embedded FPGA memory as circular line buffer with a total latency for the whole filtering part of 2589 clock cycles (4 lines and 29 clock cycles for a VGA image). But this latency depends on the number of columns of the image. Since we are using 9x9 pixels filters, the first four lines of the image results are discarded. The input for stage $\mathrm{S}_0$ is a channel containing the three pixels of the temporal sequence. Filter banks are replicated three times: one for each input pixel. On the output, the circuit produces for each pixel 8 real values and 8 imaginary values (a complex value for each orientation): fixed point representation for this stage has 8 bits for the integer part and 2 for the fractional part.

### 4.4.3 Phase computation

Phase for each orientation is calculated starting from real and imaginary values of the previous step. This operation needs an arctangent core. We use a Xilinx [3] IP core generated with the Core Generator tool and based on the CORDIC algorithm [60]. For this stage, we need 24 different cores, one for each orientation and input frame. Every core has a throughput of one pixel per clock cycle and a latency equal to the output bit width plus 4 more clock cycles. Arctangent cores have 10 bit width variables on the input and 10 bit width on the output; consequently, we have 14 cycles of latency. On the output of this core, the 2QN fixed point representation is used (1 bit of sign, 2 of integer part, and the rest, for the fractional part) and values in the range $[-\pi, +\pi]$ are provided: in our case, for the fixed point representation, we use 1 bit for the sign, 2 bits for the integer part, and 7 bits for the fractional part.

### 4.4.4 Unwrapping the phase values

In this stage, we compute a temporal filter among the phase of the three temporal input frames. With this operation, we eliminate discontinuities of the phase due to its periodicity. The first temporal frame phase value is not changed, the other two are transformed by applying a subtraction operation that subtracts $2\pi$ at points with discontinuities. Basically, we apply the following basic operation

```
for n = 2 to N do
    if $\phi(x, y, \theta, f_n) - \phi(x, y, \theta, f_{n-1}) > \pi$ then
        $\phi(x, y, \theta, f_n) = \phi(x, y, \theta, f_n) - 2\pi$
    end if
end for
```

Where N is the number of frames of the temporal sequence: we indicate for each variable spatial position (x,y), orientation $\theta$ and frame f. This circuit consists of 32 adder circuits (4 for each orientation) and 16 comparators (2 for each orientation). For this block, we implement a pipeline of 6 clock cycles of latency. Output values of this stage use 5 bits for the integer part and 5 bits for the fractional part.

### 4.4.5 Component velocities estimation

The fourth stage computes component velocities at each orientation as described in Section 2.1.1. Basically, temporal information is used to determine velocity magnitudes and velocity orientation is provided by the filter orientation. The temporal phase gradient is computed from the previous unwrapped data. This information is processed to compute local velocities. A linear threshold is calculated for each pixel and assigns a reliability weight to each output value. This block takes 24 values as inputs (8 unwrapped phases for each input frame), the block produces 8 values (one for each orientation) and another 8 that represent the related thresholds for each orientation-based estimate. Divisions in this block are avoided using the following strategy. Considering that for denominator we have constant values, we approximate division with a multiplication and a right shift operation in order to save logic resources and benefit from the embedded arithmetic resources of the FPGA. If we have to divide by D, then the multiplication factor is computed by 2N/D, being N the value to shift the data that properly approximates the division operation. The latency of this stage is 9 clock cycles as indicated in Fig.4.2.

### 4.4.6 Final velocity vectors

Information of different orientations is finally merged in the fifth stage. For each component velocity computed in stage four, we apply the reliability threshold and consider as good velocity vector values the ones that have, at least, a number of reliable components higher than a defined threshold. A sum among valid orientations is operated as indicated in (2.4). At this stage, we have variable denominators, thus, we need the implementation of two hardware dividers (one for the x component and one for the y one). We use Xilinx IP dividers from the Core Generator. This pipelined stage has a latency of 5 clock cycles plus the divider latency: 5+ (D+4) where D is the bit depth of the divider input. Non reliable values are sent to the output as NaN; we codify NaN values with the reserved signed value of ”0b100000000000” assuming that output values have 12 bits.

## 4.5 Coprocessing and interface to standard PC

The complete system has been implemented in a Xirca V4 board described in the Appendix 8. The system described in this section works as a co-processing board where input images are written in the SRAM memory banks from a host PC, which receives the sequence from a camera. All these operations are managed by software developed for this specific application but this software is easily adaptable to other real-time vision applications: a screen capture of the user interface and a real-time processing is shown in Figure 4.3. The processing system uses an additional memory bank (Fig. 4.4) to store the pyramid information and synchronize data between this block and the one that computes optical flow (see Fig. 4.1). The memory map is calculated when processing begins and depends on the host parameters. The input parameters can define the size of the input image and various processing thresholds. This software is currently available as open source software (built up by F. Barranco) at http://code.google.com/p/open-rtvision/.

## 4.6 Architecture optimizations: tuning for low resources or high performance

Depending on the chip size and the platform used, especially the number of banks, the circuit can be optimized towards different targets: the first involves an optimization of the frame rate and the second an optimization of hardware use. This allows us therefore to adapt the basic design to high-performance application or low-cost application requirements. The former is obviously constrained by the resources available on the target chip. For instance, a simple way of increasing processing performance is to replicate the processing circuit and synthesize it for a larger FPGA. As pointed out in [3], our chip is not the most powerful device available on the market. The design can be adapted to a Virtex 6 device, which can provide up to 8 times more resources than the Virtex 4 device in our board. In this case, the multi-core architecture can simply replicate the processing cores and split the input image into a number of parts equal to the replication of cores (with a slight overlap to handle image boundaries). Although this option is feasible and easy to address in the future we do not intend to focus on it here because prices and power consumption increase significantly and these are very valuable resources for embedded devices. Instead we will concentrate on architectural modifications. The most critical part of the complete circuit is the warping block. All the circuits except this one work at a data rate of 1 pixel per clock cycle, but warping takes on average 2.5 clock cycles to process a single pixel because it depends on the number of memory accesses needed for reading the neighboring pixels, and this is an unpredictable process, depending as it does on the fractional optical-flow values of the x and y components. Improvements during this stage, which is the bottleneck of the system's performance, will improve the overall performance of the circuit considerably. Furthermore, warping occupies all the memory channels available in the OFB bank and therefore we need an additional memory bank if we want two new warping cores. If we take 2.5 clock cycles as a reference, we can improve on this processing by a factor of 2. The FPGA processing time takes a number of clock cycles that can be estimated using:

$$C = (1+s)I + (s+R)I = I(2s+R+1) \tag{4.1}$$

where R is the inverse of the data rate for the warping, I is the image resolution (height image resolution multiplied by width image resolution) and s is the scale factor, which depends on the number of scales, as indicated in:

$$s = \sum_{n=1}^{N} 2^{-2n} \tag{4.2}$$

where N is the number of scales. The first term in Eq. (4.1) is related to pyramid computation (right-hand block 1in Fig. 4.1) and the second is related to the processing (left-hand block in Fig. 4.1). These equations do not take into account the computing times necessary for priming (filling in) the pipeline because these terms are fairly small compared to the processing time and are therefore negligible. Using Eq. (4.1) and taking the worse case of R, we can calculate the frame rate for different image sizes simply by dividing the frequency of the circuit by C. Thus, in the original version for image resolution of 512x512 and with R=2.5 we obtain a frame rate of approximately 36 fps; if R=1.25, on the other hand, the frame rate increases to 48.2 fps. Now, if we want to optimize the use of hardware resources we need an architecture that exploits the resource-sharing capacity of this computing scheme. This means that we use some blocks in a further sequential code, thus increasing the processing time but reducing hardware resources. As an example we developed a shared version of the mono-scale core which removes the repetition of similar blocks. As shown in Figure 4.2, the architecture contains the repetition of the same filtering circuit for each frame and also the atan2 cores. If we share the same block for the three different frames we reduce hardware use by approximately 22% and increase processing time by a factor of three. In Table 4.1 we describe hardware use and compare the basic design with this sharing optimization. With the last low-cost version, Eq. (4.1) becomes:

$$C = (1+s)I + (s+3)I = I(2s+4) \tag{4.3}$$

in which we replace R with 3 because we know that n processes running in parallel take as long as the slowest process, which in this second case is optical-flow computation. The final frame rate calculated with Eq. (4.3) will be one of approximately 31 fps. This represents a very moderate reduction in performance at a lower cost (in terms of hardware resources).

Table 4.1 also shows the total hardware resources required. The whole system uses 86% of the available slices in a Virtex 4 xc4vfx100 for the fastest version. The sharing version saves about 9% of total resources (close to 4,000 slices) allowing a possible

**Table 4.1:** Hardware use for a Virtex 4 xc4vfx100. We describe the recourses used for different circuit parts. Here we compare three different architectures: the first is the original implementation, the second is an alternative with a sharing strategy and the third is a high-accuracy version with Gabor filters. The table is divided into four different parts: total system resources, hierarchical blocks and interface, mono-scale parts and total mono-scale cores.

| Circuit | Total 4 inputs LUTs (out of 84352) | Slices Flip Flops (out of 84352) | Slices (out of 42176) | DSP (out of 160) | Block RAM (out of 376) | Freq. (MHz) |
|---|---|---|---|---|---|---|
| Total System (Sharing) | 45415 (53%) | 36468 (43%) | 32744 (77%) | 132 (82%) | 106 (28%) | 43.6 |
| Total System (not sharing) | 60564 (71%) | 40073 (47%) | 36603 (86%) | 132 (82%) | 106 (28%) | 45.05 |
| Total System (Gabor, sharing) | 51368 (60%) | 38905 (46%) | 35023 (83%) | 147 (91%) | 112 (29%) | 41.3 |
| Board Interface | 4774 (5%) | 5195 (6%) | 5288 (12%) | 0 | 36 (9%) | 112.4 |
| Interface + warping | 9943 (11%) | 9097 (10%) | 9894 (23%) | 32 (20%) | 43 (11%) | 50.8 |
| Reduction | 364 (1%) | 244 (1%) | 235 (1%) | 0 | 4 (1%) | 106.7 |
| Expansion | 413 (1%) | 270 (1%) | 367 (1%) | 0 | 1 (1%) | 85.7 |
| Filter | 3904 (4%) | 2540 (3%) | 2343 (5%) | 59 (36%) | 8 (2%) | 77.9 |
| Phase | 2544 (3%) | 2715 (3%) | 1814 (4%) | 0 | 0 | 68.9 |
| Unwrapping | 341 (1%) | 240 (1%) | 256 (1%) | 2 (1%) | 0 | 123.4 |
| Component velocity | 587 (1%) | 442 (1%) | 391 (1%) | 9 (5%) | 0 | 82 |
| Full velocity | 1227 (1%) | 2153 (2%) | 1412 (3%) | 13 (8%) | 0 | 85.6 |
| Mono-scale Optical Flow (sharing) | 18439 (21%) | 17651 (20%) | 15691 (37%) | 100 (62%) | 24 (6%) | 51.9 |
| Mono-scale Optical Flow (not sharing) | 32569 (38%) | 22645 (26%) | 20024 (47%) | 100 (62%) | 24 (6%) | 59.7 |

**Table 4.2:** Power consumption for optical flow circuits in a Virtex 4 xc4vfx100: estimated with Xpower tool.

| System | Power (W) |
|---|---|
| Total System (sharing) | 4.35 |
| Total System (not sharing) | 4.17 |
| Total System (Gabor filters sharing) | 4.47 |

implementation in cheaper devices such as Virtex 4 xc4vfx60. It should be noted that the sharing option slightly decreases the total clock rate, which is coherent with the mono-scale optical-flow results. This can be easily understood when bearing in mind that arbitration logic is required for the shared modules and that this increases logic depth. Table 4.1 also shows a further high-accuracy version of the algorithm that uses the separable Gabor approach for the filtering stage; the quantitative results for synthetic benchmarks using this version are set out in Section 4.7. For the three different versions of our architecture and for embedded purposes we also analyzed power consumption with the Xpower tool and report the results in Table 4.2.

On the other hand, it is possible to reduce hardware resources in a mono-scale version, which saves all the hierarchical circuitry. The architecture is reduced to a single mono-scale core and the board interfaces. Because of its high parallelism, this mono-scale architecture can achieve a frame rate of 160 fps for VGA images but is unable to compute large displacements between subsequent frames. Within the context of this idea, the mono-scale architecture of the LK algorithm is described in [20], in which an explanation of high-frame-rate architectures is proposed. Due to its inherent limitations the mono-scale approach is suitable only for high SNR and high-frame-rate cameras, which are not frequently used in normal conditions. With this kind of camera the algorithm tunes the higher spatial frequencies better than the more complex multi-scale architecture proposed in this work but it loses estimation density in low-textured areas and general adaptability in sequences involving large pixel movement (fast motion).

**Table 4.3:** Frames per second at different image sizes and number of scales. In a mono-scale approach the system can process up to 186.26 fps using images of 512x512, but this approach is limited in range.

| Size | # scales | Frame per second<br>(sharing) | Frame per second<br>(not sharing) |
|---|---|---|---|
| 512x512 | 4 | 29.61 | 36.8 |
| 640x480 | 4 | 25.25 | 31.5 |
| 640x480 | 5 | 25.1 | 31.09 |
| 720x576 | 4 | 18.75 | 23.26 |

## 4.7 System results

As seen in Section 4.3, the system's architecture combines both parallel and sequential parts and so the platform processes the final image at a lower speed than the corresponding data rate of each individual processing block. The final performance (Table 4.3) will depend on the number of scales (computed sequentially) and on the image size as described in Section 4.3. This drawback can be solved by loop unrolling but this involves the disadvantage of significantly increasing system latency, which may not be very suitable for real-time applications. We measured the frame rate experimentally by simply computing the processing time to transfer 5,000 images; note that the frame rate was higher than the value expected from (4.3) in which R was set to the worst case. In fact we measured a real case: access to memory for the warping operations benefited from spatial data coherence and therefore the average time was slightly lower than 2.5 cycles.

The system's accuracy was assessed with tests on the synthetic "Yosemite" sequence because this is widely used in the literature. For this sequence we have the ground truth information so it is possible to measure errors in the algorithms and compare them with the existing methods, as explained in [77]. Our errors do not include the area of the sky. In Table V we describe the angular error in degrees (AE), the standard deviation from the angular error and the density of the valid values (eliminating the total number of pixels in the sky area).

A comparison of our work with some real-time implementations in the literature [20, 78] reveals how the final system is more accurate but results in a lower frame-rate performance for the multi-scale case. In some solutions, such as in [20], the high frame

**Table 4.4:** Angular errors for different Middlebury sequences [2]. We use four scales for all sequences.

| Sequence | AE | Std | Dens |
|---|---|---|---|
| Yosemite | 4.69° | 7.01 | 82.81 |
| RubberWhale | 11.2° | 20.6 | 79.09 |
| Grove3 | 12.08° | 19.34 | 92.62 |
| Urban3 | 13.72° | 25.94 | 49.28 |

rate is the result of the mono-scale origin of the algorithm, with an evident loss in accuracy and density. For the Yosemite sequence and a dense optical-flow map in [20] the system has an AE of 18.3°and in [78] the authors achieve 6.7°, whilst our system achieves 4.69°, as shown in Table 4.4. Our work improves both the accuracy and motion range of these contributions. More importantly, our algorithm in the mono-scale version can also achieve a very high frame rate even when using the PCI-express interface restrictions. Compared with these other works, our architecture surpasses them in complexity (Section 2.1.1), which allows it to be used in complex scenarios thanks to the proven stability of the phase information against affine deformations and illumination changes [29]. A comparison of our system with the GPU-based implementation of the same algorithm provided by Pauwels et al. [41] is extremely interesting. They achieved a similar processing speed with a smaller error of 2.7°. The increase in error engendered by our hardware system compared to theirs is mainly due to the fact that they use floating-point arithmetic and five temporal frames. Nevertheless, this difference is acceptable if we bear in mind that our approach can be included in embedded systems, where parameters such as size, power and certification capabilities are key elements that are unachievable by GPU-based approaches. There is also a novel contribution for commodity processors [13]. Despite the impressive performance that Anguita et al. achieved in software, their system involves the same drawbacks as those in [20] as far as accuracy and motion range are concerned because of their using a simple mono-scale engine. The qualitative results of our system for different benchmarks are set out in Fig. 4.5. Input images are chosen from an exhaustive dataset available in [2].

Additional results for a real sequence are shown in Figure 4.3 together with the software interface and demonstrate the validity of our system for non-controlled scenarios as well; in this case we cannot evaluate the numerical errors but the satisfactory motion

**Table 4.5:** Comparison with other approaches described in the literature. Our mono-scale engine achieves 24.8 MPPS.

| Work | Image resolution | Frame rate | MPPS | Method | Processor type |
|---|---|---|---|---|---|
| Presented work | 640x480 | 31.5 | 9.6 | Phase-based | Custom FPGA,<br>Xilinx Virtex 4 (45 MHz) |
| Presented work<br>mono-scale | 640x480 | 81 | 24.8 | Phase-based | Custom FPGA, Xilinx Virtex 4 (45 MHz)<br>Xilinx Virtex 4 (45 MHz) |
| Botella et al. [79] | 128x96 | 16 | 0.19 | Multi-Channel Gradient | Custom FPGA<br>Xilinx Virtex 2 |
| Anguita et al. [13] | 1280x1026 | 68.5 | 86.8 | LK | Core 2 Quad<br>Q9550 (2830 MHz) |
| Diaz et al. [20] | 800x600 | 170 | 82 | LK | Custom FPGA,<br>Xilinx Virtex 2 (82 MHz) |
| Pauwels et al. [53] | 640x512 | 48.5 | 15.8 | Phase-based | NVIDIA GeForce<br>8800 GTX |
| Wei et al. [78] | 640x480 | 15 | 4.6 | Tensor-based | Custom FPGA,<br>Xilinx Virtex 4 (100 MHz) |
| Murachi et al. [80] | 640x480 | 30 | 9.2 | PLK | VLSI<br>(332MHz) |
| Bruhn et al. [24] | 160x120 | 63 | 1.2 | Variational | Intel P4<br>3060 MHz |
| Sosa et al. [81] | 256x256 | 30 | 1.9 | Change-Driven | Custom FPGA,<br>Altera EP20K1000C (33 MHz) |
| Martin et al. [82] | 256x256 | 60 | 3.9 | H&S | Custom FPGA,<br>Altera EP20K300EQC240-2 |
| Niitsuma et al. [83] | 640x480 | 30 | 9.2 | SAD | Custom FPGA,<br>Xilinx Virtex 2 |
| Correia et al. [84] | 256x256 | 25 | 1.6 | LK | MV200 [85] |

estimation for a driving scenario in which environmental factors are not controlled is evident. Apart from the accuracy results in the various benchmark sequences set out in Table 4.4 and Figure 4.5, we also compared our system with some other approaches in the literature to emphasize the validity of our contribution: a brief resume is set out in Table 4.5.

## 4.8 Conclusions

We describe here the implementation of a co-processing system that is capable of achieving a frame rate above that of traditional software implementations in a conventional processor. Thanks to the stability of the phase information and the high parallelism of FPGA devices we have been able to implement a motion-estimation system that works in real time with high performance and is also useful for various different applications and problems. Since the architecture is customized for FPGA devices it represents a very valid alternative for embedded image-processing systems. The system's architecture uses a highly complex multi-scale, multi-orientation algorithm which involves the implementation of 1,750 parallel processing units for the mono-scale core alone, and this number increases to more than 2,000 basic processing elements with the multi-scale implementation. Due to the high complexity and computational effort which our system represents it is, as far as we know, one of the most complex motion-estimation systems implemented in FPGA devices to date. This is justified by its great robustness both in synthetic scenes and real sequences. The multi-scale extension allows the detection of movements of more than 10% of the input image size. The final circuit processes VGA images up to 31 frames per second with 4 scales. The main advantage of the FPGA implementation is its eminent portability; in fact the same co-processing board used in this work can be used as a very useful standalone platform in industrial systems, such as in assisted or autonomous navigation in automotive fields. Most other approaches defined in FPGAs only implement a mono-scale model. The implementation of a multi-scale approach is indeed complex because the warping operation requires the displacement of pixels along the different axes (depending on the estimated motion in previous scale). This represents an operation requiring a large amount of random memory access. Therefore, an efficient access to external off-chip memory resources is critical in this multi-scale approach. The implementation of a multi-scale approach renders the system described in this chapter easily adaptable to different scenarios with high motion ranges. We are currently working on the use of the Gigabit Ethernet interface as camera input in order to obtain a completely autonomous system. Future work will also include the use of embedded PowerPC processors to explore more complex system-on-chip (SoC) and Hw-Sw co-design strategies to explore the reconfiguration capabilities of the processing engine, as well as to be able to extract complex motion information such as estimates of ego-motion, time to contact etc., all using advanced co-design techniques.

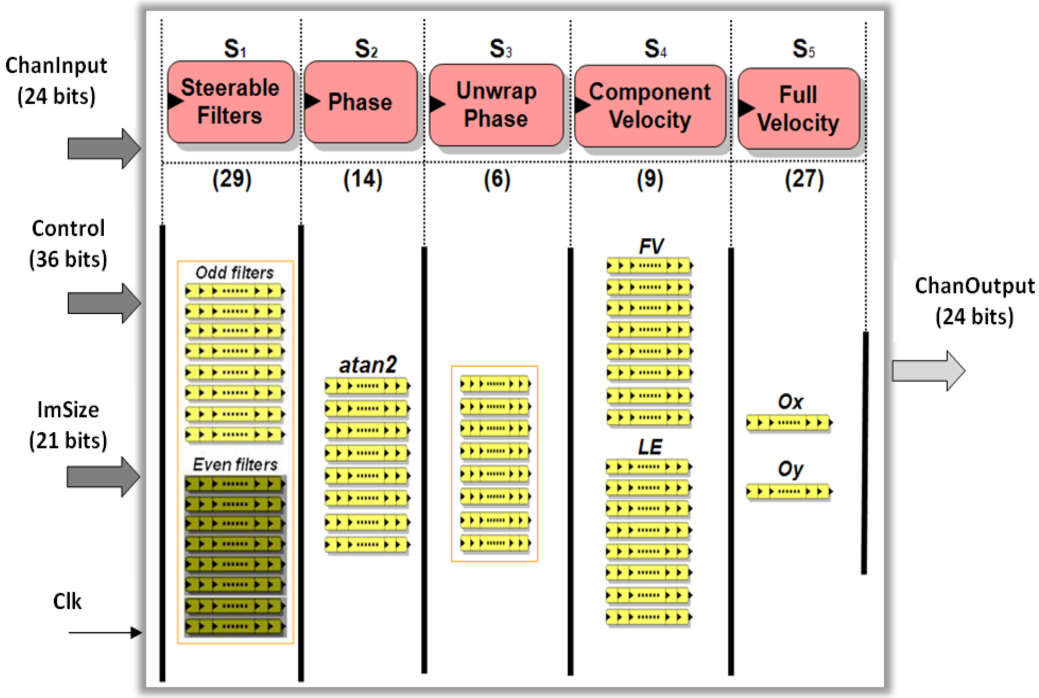


**Figure 4.2: Optical flow processing core.** - The mono-scale optical-flow core is divided into 5 different stages, as detailed in Section 2.1.1: S0 is the filtering stage, S1 the phase calculation, S2 the phase unwrapping, S3 the determination of component velocities and S4 the choice of final velocity. The S0, S1 and S2 data-paths are repeated for each temporal frame. Horizontal arrows represent graphically the super-pipeline depth (the exact number of pipeline stages for each part is specified in brackets) and the vertical repetition of data-paths represents the super-scalar depth (each line corresponds to a different orientation). The final design contains a significant number of parallel processing units and is fully pipelined to achieve high performance. The number of pipelined stages for each block presented is shown in brackets. The total number of processing units is 1,750 (the number of pipelined stages multiplied by the number of scalar units). This illustrates the high complexity of the system architecture.

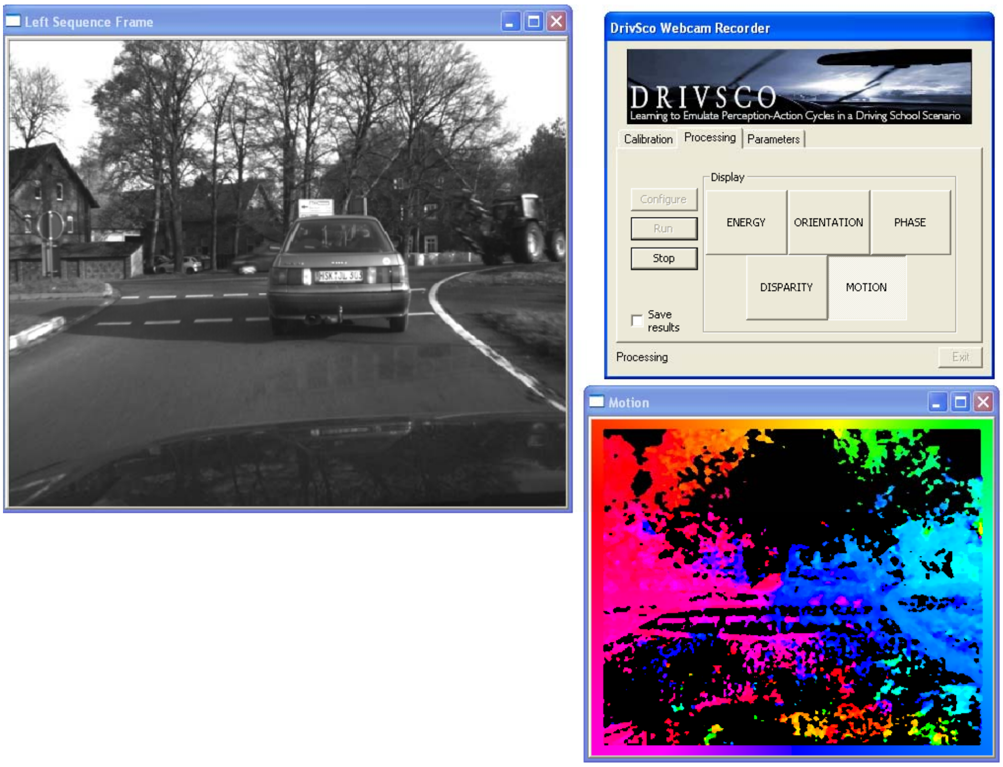


**Figure 4.3: Screen capture of the optical flow application.** - The optical-flow computation in real-time for the left input camera is displayed in the lower right screen.

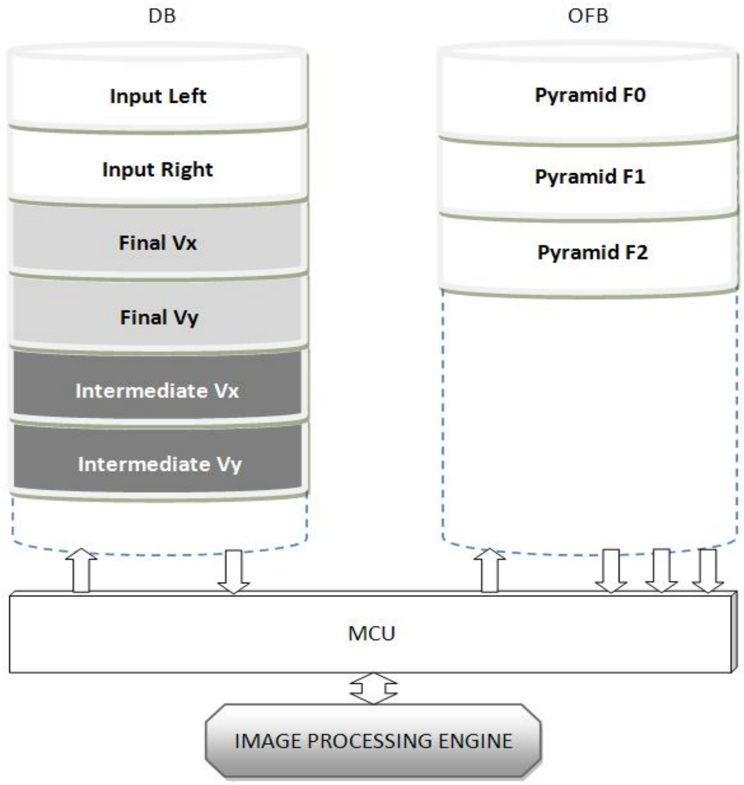


**Figure 4.4: Memory mapping for optical flow.** - Memory mapping, including the double buffer (DB) used for data transfer with the PC and the optical-flow buffer (OFB) used as support for the main processing. The DB bank (on the left) contains: input images (first/second white rows), the final result (third/fourth, light-gray rows) and information about previous scales (fifth/sixth, dark-gray rows). The pyramid of the input frames is stored in the OFB. The SRAM banks work with words of 36 bits. White data have 8 bits for each pixel and are packed in vectors of 4 pixels per memory address. Light gray represents data with 12 bits, which are packed in vectors of 2 data, and finally, dark-gray data have 12 bits and are packed in vectors of 3. Arrows between MCU and memory banks represent the different AAPs used: 1 read and 1 write for the DB bank, 1 read and 3 write for the OFB bank.

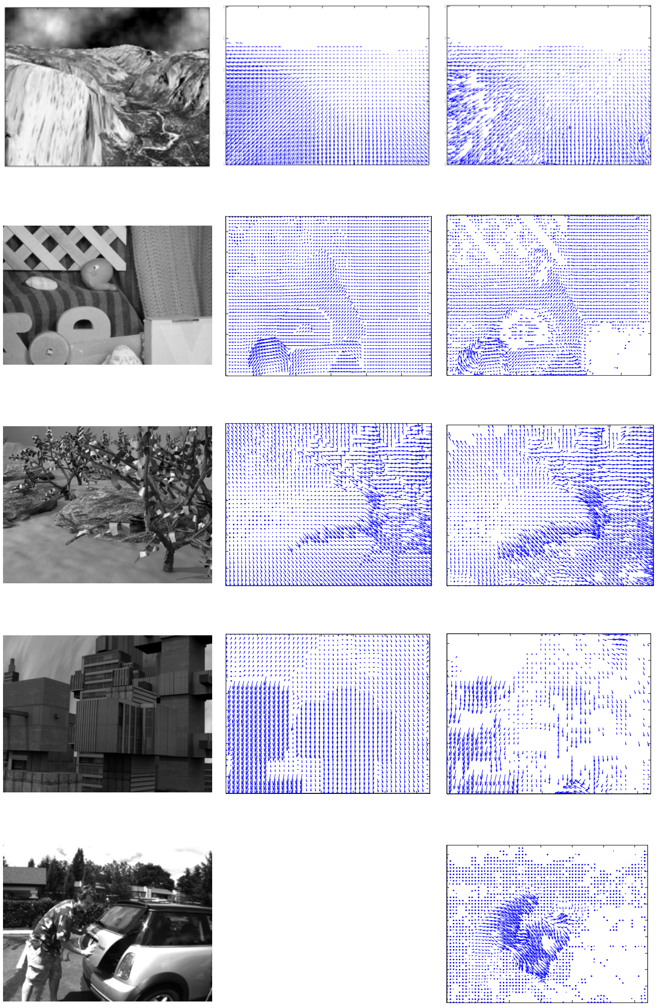


**Figure 4.5: Qualitative results for different Middlebury sequences.** - On the left we display one of the input frames, in the middle the ground truth, and on the right our system's results.

# 5

# Low level vision engine

## 5.1 Abstract

In this chapter we illustrate how FPGA devices and novel design techniques and methods could be applied for high performance low level vision systems. As previously justified in Chapter 2 we adopt a phase-based algorithm and integrate architectures of Chapters 3 and 4 to extract multi-scale optical flow, disparity, energy, orientation and phase. Depending on the system target application, we demonstrate that it is possible to develop diverse hardware implementations with different performances, resources utilization and accuracies trade-offs tuned according the required specifications (power, price, performance, accuracy, etc.). Making use of all these strategies, we implement a high performance low level vision engine that achieves real-time processing on the same chip. The system computes multi-scale optical flow, stereo and local contrast descriptors (energy, orientation and phase) at 28.6 fps (image resolution of 512x512). Rather than a single implementation, this chapter presents a design and integration strategy, it evaluates different resources sharing techniques and illustrates the versatility of FPGA based system that can be easily adapted to diverse target platforms to fulfil cost vs. performance specifications. We analyze multiple hardware sharing strategies as well as different memory resources utilization alternatives. Finally we study the impact in the design of different architectural decisions. The exploration of the design space allows us determining the best solution for a target application and generalizing our results to other vision algorithms.

## 5.2 Introduction

The Field Programmable Gate Array introduction in the large scale industry allows that a lot of application fields benefit its easy customization of massive computing resources. Important works on video surveillance [86], medical imaging [87], robotic [88], biology [89, 90, 91] and on many other areas reveal the great importance acquired by FPGAs. But how many kinds of architectures can be designed for the same target? Reconfigurable devices make possible an easy prototyping of hardware designs and accurate preliminary studies for complex systems. Beyond this idea, current FPGA devices are a stable platform for many final system applications. In this chapter we address the implementation of a complex low-level vision engine which will illustrate novel design methodologies for systems with high computational effort. In the machine vision community, a huge amount of contributions state how parallelism of FPGA devices suites very well for pixel-wise operations as in low-level vision. Previous works (previous chapters) implement different features extraction as optical flow [20] and disparity [19] or image processing as denoising [92], deblurring [93] or segmentation [94]. Generally most of them propose a single vision task and does not compel with hard requirements in terms of device size or power consumption. For this reason architectures offered are often focused on a specific device where the design fits without problem and does not need of specific algorithm adaptations. In order to present an exhaustive study of different possible implementations of the same architecture we choose to integrate in the same chip the extraction of optical flow, disparity, energy, orientation and phase. A phase-based algorithm described in [40] has been adopted for its robustness and performance. To the best of our knowledge, this is the first time that a low level vision engine with these features computed in real-time has been developed in a single chip and it is one of the most complex image processing systems described so far on FPGA. After a study of a proper algorithm adaptation to hardware, we examine the sharing capabilities of the complete system and we demonstrate the possibility of changing the architecture according to different system requirements. The main contribution of this work is a new design methodology independent from hardware platform and generally applicable to other vision algorithms. The final system implemented represents a great advance in the area of the vision on chip. The architecture is able to process up to 420 Mb/sec achieving the real time for images of 512x512. It benefits from all FPGA capabilities (reduced size, high performance, low power consumption, etc.) and can potentially be converted in a system for industrial purposes. Efficient integration of different high performance cores on a single chip is not straightforward. In this chapter we describe how a complex architecture including the on-chip computation of different vision primitives can be addressed adopting different resources sharing strategies. The system includes vision engines such as optical flow (originally composed of 1810 parallel processing elements, description of Chapter 4), stereo (originally composed of 1145 parallel processing units, description of Chapter 3) and local features (originally using 312 parallel processing units as described in [95]). We study different implementation alternatives, adapting the system to a low, medium and high cost version arriving at different performance vs. cost trade-offs.

## 5.3 Design strategies for a hardware implementation

Depending on final system requirements, the architecture can be tuned to fulfill different target specifications. Main interests are performances and hardware costs, of course the first one depends directly on the second one. Thus if we need high performance we have to increment the hardware recourses. Performance can be estimated in terms of processing speed and accuracy. The first one can be improved with a fully parallel implementation and at the other hand is limited with sequential execution or lower frequency clock. As displayed in Fig. 1.3 a sequential execution allows a recourse sharing and it can optimize the hardware utilization. Hardware utilization can be reduced also by reducing accuracy. Main strategies for this aim are the reduction of bit width in the fixed point representation and an algorithm simplification based on simpler or hardware friendly operations. The high complexity of the system developed allows to explore multiple approaches in a deeper way than previous works (and chapters) about on-chip computer vision architectures. In this section we analyze all these variables starting from those which affect the accuracy and we report how they affect the system.

As previously explained in Section 2.2, we study different modifications with respect to [40] and how they affect the computation. Different filter banks study described in Section 2.2 give us a general idea of accuracy achieved using different strategies. Basically we study two solutions: a separable implementation of Gabor filter as in [47] and a Gabor-like version based on the second order derivative of Gaussian as explained in [42].

The first solution is a more accurate approach but with a considerable computational cost where we need 25 1D convolutions with a kernel of 11 taps. As an alternative, with the second derivative approximation it is possible to reduce convolutions to only 13 units with 9 taps. Another important simplification is the reduction of the number of frames in the temporal sequence from five to three, this implies saving two filter banks and consequently a minor memory effort. Cost reduction in filter stage is quite significant on the overall system. A further algorithm simplification is in the choice of the number of orientations: again with this approach are reduced the convolution units. All main modifications include a change in the filtering stage, as we can see from Table 5.1 there is a very high difference from the high performance version to the low one. Table reports at the same time the computing power saving and the accuracy on final system in both, stereo and optical flow processing for "Yosemite" and "Tsukuba" sequences.

### 5.3.1 Analysis of sharing capabilities

After the analysis of simplification on accuracy we address now to the hardware reduction analysis by sharing resources. In general a sharing strategy implies the storing of partial results for a future utilization. The alternative is a parallel replication of circuits. The choice depends obviously from the availability of memory in the first case or hardware recourses in the second. For this reason, known the algorithm and the hardware platform, it is mandatory a proper analysis of sharing capabilities and the accurate evaluation of different solutions for the final requirements. Basically after a first block diagram we focus our attention on block repetitions. For example in our case we find the parallel processing for filtering stage along temporal sequence and evaluation of different orientations along the same frame. The idea is to generate a hybrid parallel/sequential system where expensive operations are shared by multiplexing of input data in time into a single processing core. As previously underlined the most critical operation for our example is the first harmonic analysis and the band-pass filtering, thus it is a good candidate for sharing. For this purpose we have three filter banks for optical flow, two for stereo and another one for local features which can be reduced to only one. In a fully shared approach (Fig. 5.1) the filter core will attend four different data streams in a correspondent number of clock cycles; the local feature core can reuse the output of one of them and the input is shared between stereo and optical

**Table 5.1:** Computing resources and accuracy for different kinds of filter banks: Steerable stands for the second order Gaussian derivatives steerable filters and Gabor for the Gabor filters. Hardware utilization is estimated for a Xilinx Virtex xc4vfx100 [3] with the Agilent DK5 tools [4]. Accuracy columns reports the optical flow Angular Error (AE) for the "Yosemite" sequence and the stereo Mean Absolute Error (MAE) for the "Tsukuba" sequence. In brackets we indicate the density. Hardware utilization is referred to only one bank, depending on the algorithm (stereo or optical flow) this will be approximately multiplied by the number of input frames.

| Filter | Orientations | Convolution units | Slices | Accuracy (Dens.) |
|---|---|---|---|---|
| Steerable | 2 | 13 | 995 | AE=11.62(82) |
| | | | | MAE=1.92(95) |
| Steerable | 4 | 13 | 1974 | AE=8.82(86) |
| | | | | MAE=1.3(95) |
| Steerable | 8 | 13 | 2754 | AE=8.04(85) |
| | | | | MAE=1.26(95) |
| Gabor | 2 | 7 | 1624 | AE=8.45(77) |
| | | | | MAE=1.88(94) |
| Gabor | 4 | 15 | 2816 | AE=5.5(82) |
| | | | | MAE=0.99(95) |
| Gabor | 8 | 25 | 6273 | AE=4.58(79) |
| | | | | MAE=0.79(95) |

flow: in this case the rectified image is used also for optical flow, otherwise different inputs imply a further cycle for the central frame of optical flow. Two different filter sharing mechanism are possible:

1. Complete filter sharing by storing intermediate data on external memory (full filter sharing option).

2. Arithmetic filter sharing (each input stream keeps their local memory buffers and only arithmetic units are shared).

The second option has the advantage of no additional writing/reading operation, therefore, circuit performances can be computed as $F_{clock}/N$. We use this option because current FPGAs are plenty of on-chip memory resources. Note that the first approach (typically used in processors), do not fit properly here because it increases the number of memory accesses that is one of our bottleneck due to the slow interfaces. Obviously the used approach increases logic resources to manage and store temporary data but save hardware compared with the fully parallel approach. In Fig. 5.1 a brief clock cycles diagram is shown for three different methods: the first one is a fully parallel implementation, the second is a sequential approach that share filters only inside different processing cores (stereo and optical flow separately) and the last is another sequential one with sharing of data between different processing cores.

From a frequency point of view, using sharing strategies, we have a new block that works at f/N MHz where f is the system clock frequency and N the number of cores shared. Thus we need a proper signal control to order data in input and output from slower block. We also need specific circuits responsible of the sorting of data and the temporal storing in further pipeline stages that prevent a mixing of data. If the shared block operates with more adjacent pixels, the internal circular buffers are repeated a number of time equal to shared processes in order to differentiate paths. After these critical and accurate synchronization strategies the final throughput will be divided by 4 in the slower case or by 3 in the balanced solution and the hardware utilization will be reduced as displayed in Table 5.2. Compared with a traditional sharing strategy, some logic is added for the management of the multiplexing in time and for the temporal buffers. Various intermediate solutions especially for phase processing can be adopted for specific speed vs. data throughput trade-offs. In the second solution in Fig. 5.1 we split the filtering stage in two parallel blocks, one for the optic flow that shares 3 filter

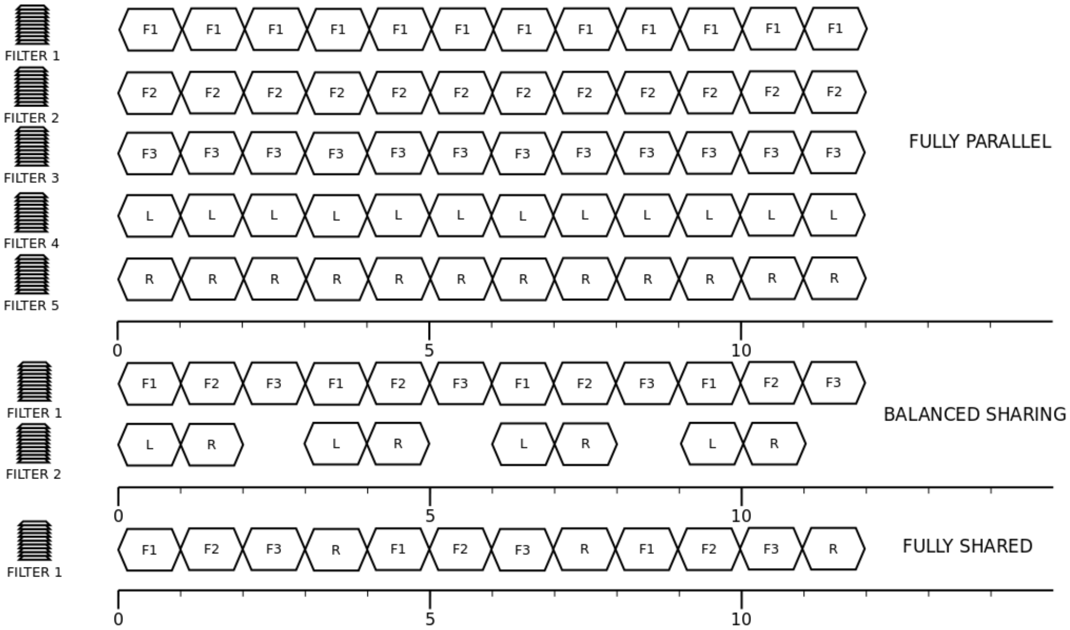


**Figure 5.1:** Different sharing strategies are shown with a timing diagram. A fully parallel implementation provides one data per clock cycle throughput while the slower sequential approach achieves only one pixel in four clock cycles. The fully parallel architecture works for 3 temporal images of optical flow and 2 images (left and right) for the stereo case. The local features could use the input from the middle frame of the flow and therefore no additional filter bank is required.

**Table 5.2:** Hardware utilization for different number of steerable shared filters. Results are provided by DK synthesis tool for 4 orientations filters.

| # Banks | Slices |
|---|---|
| 1 | 1855 |
| 2 | 2829 |
| 3 | 3925 |
| 4 | 4275 |

banks and one for the stereo algorithm that shares 2 banks. With these intermediate approaches the throughput will depends on the slower process, in this case we obtain one pixel every three clock cycles (Fig. 5.1).

Note that, if required, the sharing strategies may also be extended through the different filter orientations (using the same FIR filter circuitry but modifying the filter taps values). In our case the reduction of hardware resources is only marginal because sharing across orientation requires fully implementation of multipliers for the FIR filter operation. Using fixed multipliers coefficients (no sharing), allows using FPGA logic to implement simple multiplications (for instance for factor power of two). Consequently, we do not get a significant hardware resources reduction by FIR filters sharing across orientations and therefore this option has not been taken into consideration (in fact, depending on the filters coefficients, there are situations where the "sharing approach" could even be more expensive). Going on with the analysis of sharing strategies we realize from the first block estimation that a further parallelism of blocks is present along orientations and frames. Both, optical flow and stereo, need arctangent cores for the phase calculation with a total repetition of this core in 24 units for the first case and 8 for the second (we are using the approximation of equation (2.5) described in [43]). Adopting a proper management of data as previously described, the hardware saved can achieve a maximum factor of 32 but with a drastic limitation in the global throughput. In this case we can achieve a high number of sharing processes but the maximum is not the optimum choice. First of all we opt for a first division in two big cores, one for optical flow and another for stereo: arctangent for stereo has a different input (crossed components) from optical flow. This allows a hardware saving in the input data management. With this first separation we can share up to 24 arctangent processing units (Fig. 5.2). Now for a further simplification on logic it is possible to

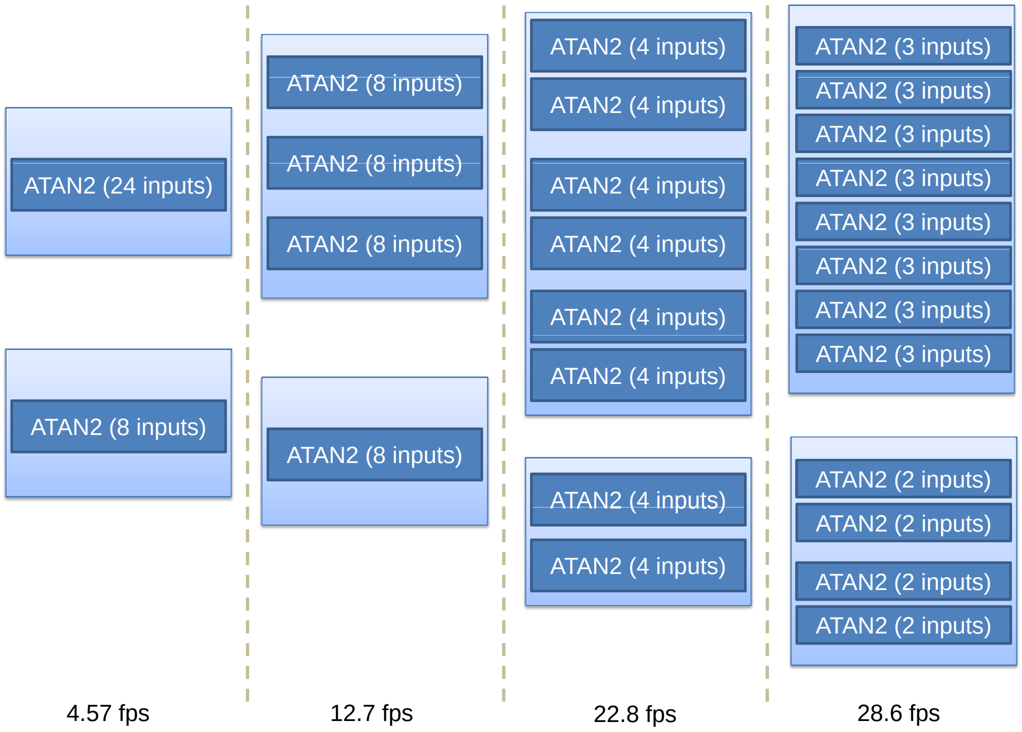


**Figure 5.2:** Different sharing phase solutions: atan2 cores are the sames of stage 2 in Fig. 3.3 for stereo and in Fig. 4.2 for optical flow. Frame rates for a 512x512 resolution are reported. Note that without sharing strategies the system uses 32 modules and achieves 57 fps.

adopt a sharing strategy along orientations or along frames, thus the maximum sharing factor is 8 (number of orientations). In order to maintain the throughput obtained with the shared filters, we choose a sharing along frames also for phase computation in optical flow, while for stereo we share along orientations always maintaining the throughput: only 8+4 (optical flow + stereo) arctangent cores are used.

## 5.4 System architecture exploration

As described in previous sections the whole system which includes the three different functional blocks of Section 2.2 can be designed with multiple designer choices. After the first general study we take advantage of the reconfigurable capabilities of FPGAs and we try to define a general solution that can be fitted in diverse platforms. This can be customized in specific solutions, we focus on three different approaches: a low cost approach that can be fitted in an economic platform, a balanced version with a good cost vs. performance trade-off and a high performance version for system with very competitive requirements. The implementation is synthesized for three different chips in order of economic value and size.

### 5.4.1 Balanced implementation

The implementation of the algorithms presented in Section 2.2 is a significant design challenge itself. If we want parallelize them and synthesize all of them in the same chip, the complexity of the goal becomes even higher. A simple integration of existing cores as the ones described in Chapter 3 and 4 is not affordable for a single device as a Virtex4 xc4vfx100: our target device for a balanced medium cost implementation. The previous analysis of Section 5.3 leads us to a performance vs. accuracy trade-off with the following main choices:

- Sharing resources. This strategy is applied to stereo and optical flow separately with a partial replication of filters paths.
- Use of 8 orientations for each algorithm.
- Steerable filters.
- 8 integer and 2 fractional bits limitation in convolution.

An important problem for this complex architecture is the memory utilization. Storing the final output results requires 60 bits for each pixel: 12 bits for disparity, 24 bits for optical flow, 8 bits for energy, 8 bits for local orientation and 8 bits for phase. Taking into account that external memory addresses use words of 32+4 bits and that we need reading and storing input images and partial results, it is clear that the memory management becomes a very complicated task. In the optimal case we access to an external memory address using only one clock cycle, but wrong memory management scheduling can degrade this performance significantly. Therefore, we conclude that it is definitely necessary a special memory controller unit (MCU) to manage the huge quantity of data accesses. As described in [59], we adopt a special architecture with different Abstract Access Ports (AAP). This critical circuit is optimized in VHDL. The whole system includes two main cores, one for the optical flow and the other for the disparity; the local features (local contrast descriptors) processing is wrapped in the stereo core (Fig. 5.3). The multi-scale architecture is replicated twice. The optical flow expansion and warping are replicated, one module for the x component and one for the y; the pyramid stage is shared between circuits and operates just before the processing. It reads/stores images directly from/to external memory where processing circuits (expansion and warping) read input images. Each block of the Fig. 5.3 represents a complex design; in following sections we analyze some of them, especially the mono-scale cores (gray boxes). All system modules have been described in detail in Chapters 3 and 4 and are now integrated adopting a very fine design strategy as described in Section 5.3. The first overview of the complete scheme in Fig. 5.3 illustrates the complexity of this novel architecture. A fine synchronization is needed to interconnect different parts. We adopt basically special structures such as blocking channels and FIFOs to connect in a large pipeline most of the processing circuits. Only the synchronization between different scales is obtained by memory operations. This is a further justification for the intensive use of the MCU module.

### Warping module

The motion compensation (warping) operation in the multi-scale algorithm is a problematic issue for an FPGA circuit as we have seen in previous chapters. This kind of devices are in fact well suited for local memory accesses but present some difficulty with random memory accesses. The sequential accesses to memory restrict the performance down to 4 pixels each 10 memory accesses. A similar circuit is used for rectification and undistortion in the stereo circuit while warping for the stereo case can be simplified as described in Chapter 3.

### Optical Flow core

As described in Chapter 4 the optical flow core receives as input 3 different frames, a control word, the clock and the image size; it computes velocity vectors and sends them in an output channel (Fig. 4.2). It is divided in 5 different ”functional” stages that run in parallel. To limit the hardware utilization we need to share the filtering stage and the phase computation, so the processing core achieves a data rate of 1/3 pixels per

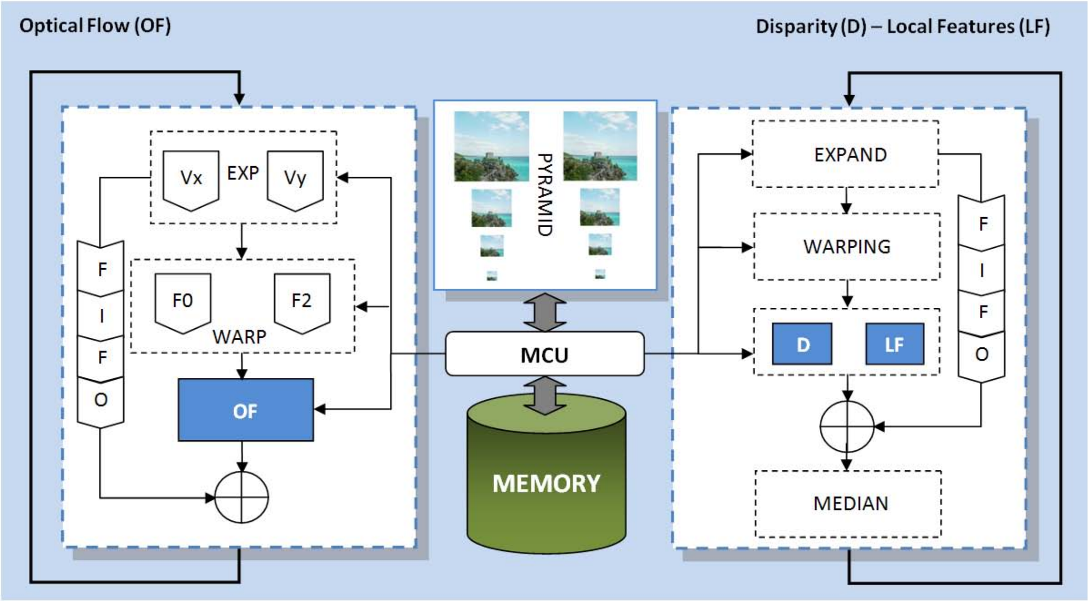


**Figure 5.3:** Whole system architecture.

clock cycle: we have finally only one filter bank (Fig. 4.2) shared in three different clock cycles and 8 arctangent modules for the phase. Summarizing the different processing stages of the core, we have:

S1 Convolution with Quadrature pair filters.

S2 Phase calculation for each orientation with an arctangent core.

S3 Temporal filter: wrapping of phase values considering their periodicity.

S4 Velocity component computation as described in [40] and according to (2.2).

S5 Threshold operation and combination of valid values for each orientation for the final velocity vector estimation according to (2.4).

In Fig. 4.2 we can appreciate different pipeline stages of the optical flow computing core circuitry. In S1 and S4 we use some IP cores from Xilinx [3] to compute complex operations such as arctangent (S1) and division (S4). Every line of arrows in the figure represents a logic path that generally is related with an orientation. All the different paths/orientations are computed in parallel. Parallel processing units adopting a sharing strategy are reduced by 47% with respect to a fully parallel implementation as presented in [96].

### Stereo core

The disparity core, as described in Chapter 3, is divided in three main steps and similarly to the previous circuit it benefits from a fine pipelined design:

1. Even (C) and odd (S) filtering with quadrature filters pairs of left and right images.
2. Disparity computation using equation (2.5) at each orientation and threshold operation assuming $k(x) \approx K_0$
3. Choice of final disparity estimation between different orientations: median value among orientations.

For a fully parallel architecture, this system needs two (left and right) second derivative Gaussian filters of 9 taps, 7 atan2 cores with CORDIC algorithm [60] for the calculation of equation (2.5) and a simple median circuit for choosing final disparity between 7 different orientations-based estimations. The filtering part takes also further 4 (half filter size) image lines cycles to fill temporary FIFO convolution, this time represents an extra latency to the total disparity computation. As in the optical flow system, we have to share the filtering stage in time (multiplexing it) for hardware recourses optimization, thus we finally have only one filter bank that processes left and right input frames. Therefore, processing takes two clock cycles. Arctangent modules are also shared: the final implementation uses only 4 cores.

### Local features: energy, orientation and phase

Local features are embedded as a block included in the stereo core (Fig. 5.3). The choice is motivated by the resource utilization strategy. Phase based local features and stereo allow a high sharing strategy for these two cores. In fact energy, orientation and phase need the same filtering stage; furthermore energy of images is used in stereo pipeline to detect reliable disparity values. Orientation uses also energy components modulated by trigonometric functions; a further arctangent core computes the final orientation from the sum of real and imaginary parts modulated as in (2.24). The phase is a hardware friendly version which consists in two adders (for real and imaginary sum of filter output) and in an arctangent module (2.8). A similar approach for local features in FPGA is described in [72].

### 5.4.2 Low cost version

A low cost/performance implementation for a smaller Virtex 4 will need a reduced accuracy and a different design strategy solution. We chose a xc4vfx60 chip with a reduction in economic cost (price) of about 65% and in logic resources of 40%. This implies a stronger sharing of resources and a consequent reduction of frame rate. In this case only one bank filter is used for the whole processing. Filter output is stored in circular buffers that provide data to three different cores: stereo, optical flow and local features. We reduce also the number of orientations to only 4 accepting the loss in accuracy studied in Section 5.3. The throughput is represented by the slowest case of Fig. 5.1 and is equal to four pixels per clock cycle. As shown in Table 5.3 the system achieves a frame-rate of 22.8 fps working with 512x512 pixels resolutions and save a considerable amount of logic resources.

### 5.4.3 High performance version

If requested by a specific application and allowed by technologic and economic availability, it is possible to increase the computational power and the accuracy of the system. First of all, this approach can benefit from a fully parallel architecture as discussed in section 5.3.1 and achieves a throughput of one pixel per clock cycle. A modification to the warping module is necessary in order to preserve this high processing capability as described in next section. This approach is not possible in an economic platform in which memory banks are not sufficient. A modern chip as a Virtex 5 is equipped with a large amount of logical resources compared with a older technology as a Virtex 4: approximately up to 4 times more in logic cells. An improvement in accuracy is affordable with a change of steerable filters for the more accurate and expensive separable Gabor filters. Accuracy gains in synthetic and real sequences obtaining errors equal to system described in [96] and [97] and reported in Table 5.3.

#### Improved warping

A fully parallel circuit for warping also in optical flow can be obtained with a major use of external memory. The original circuit operates in a sequential approach for a random external memory access. A parallel access can be adopted with a previous allocation of four pixels neighborhood (window for the bilinear interpolation) in the same

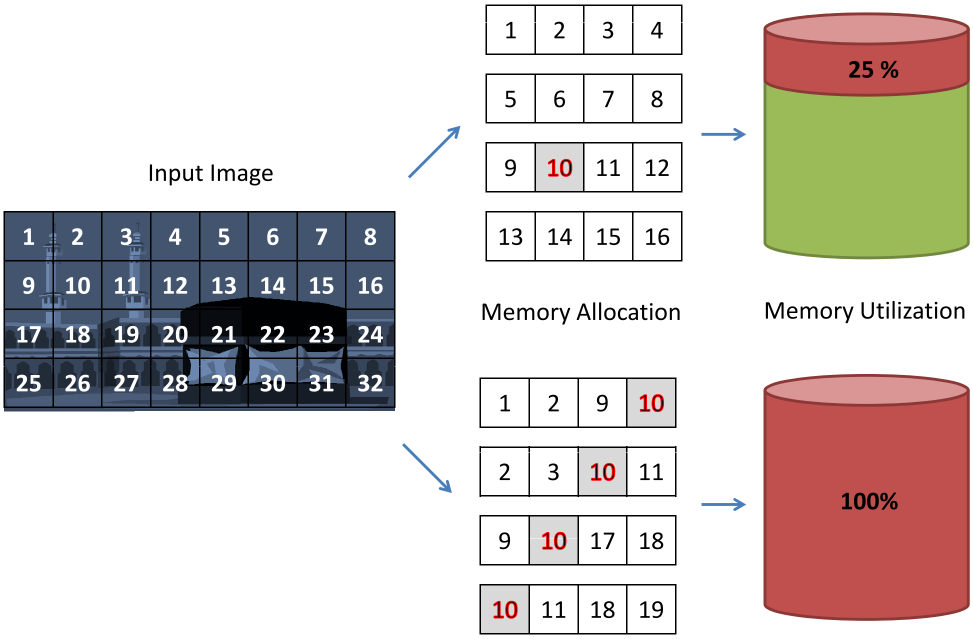


**Figure 5.4:** Warping improvement. For each pixel of the input image we store in external memory all the values of its 2x2 neighborhood. In the traditional approach (upper part) consecutive pixels are stored. In the improved warping (bottom) the same pixel is repeated 4 times.

memory address. This approach needs more external memory resources: input data are replicated four times. The preallocation is operated before the warping operation. Pyramid data are previously stored in internal circular buffers and 2x2 windows are reallocated in the same memory address. In this way the warping process need only one clock cycle to access to them. In this way the fast access of FPGA to local data benefits the future random processing. As indicated in Fig. 5.4 the cost in terms of memory addresses increases with a factor 4 for all the image pyramid.

## 5.5 Case study: Implementation on a XircaV4

We present in this section an example of co-processing board that uses the balanced architecture of Section 5.4.1. The circuit runs on a XircaV4 board produced by Seven

**Table 5.3:** Comparison for different platform implementations. Frame rates are for 512x512 resolutions and power consumption is estimated by Xpower tool. Errors are reported for ”Tsukuba” and ”Yosemite” sequences (further validations are in Chapters 3 and 4). Prices are taken from [5].

| FPGA | LUTs occupied | Power Consumption | Frame rate | Freq. (MHz) | Price (US Dollars) | GigaOPS | Accuracy (MAE / AE) |
|---|---|---|---|---|---|---|---|
| Virtex4 xc4vfx60 | 46202 | 3.8 W | 22.8 | 42 | 904 | 25.9 | 1.31 / 12.9 |
| Virtex4 xc4vfx100 | 76828 | 7.2 W | 28.6 | 42 | 2,084 | 92.3 | 0.97 / 9.99 |
| Virtex5 xc5vlx330t | 168534 | 5.5 W | 57.2 | 53 | 12,651 | 165 | 0.84 / 4.69 |

Solutions [61] and described in Appendix 8. We use the hardware/software interface described in [98] for communication between the user, the cameras and the FPGA platform. The whole system consists of a co-processing FPGA board and a host computer connected by the PCI Express interface.

### 5.5.1 Memory organization and access scheduling

A proper memory mapping has been realized to maximize the parallel access to all memory resources. A balanced and optimized solution is that the MCU provides 3 reader AAPs and 2 writer AAPs for each external memory bank. This memory mapping detailed in Fig. 5.5 has been optimized for the XircaV4 platform and for 4 memory banks. Two banks are used for the double buffer (DB1 and DB2), a third bank is used as Optical flow buffer (OFB) and the last one as Stereo buffer (SB). Green data in Fig. 5.5 use 8 bits for each pixel, red and gray data use 12 bits, the first ones store 2 pixels per address and the second 3 pixels per address. This scheme can be improved with a different platform where more banks are available. Concurrent accesses are possible thanks to the use of multiple banks and to the faster clock domain of the MCU.

### 5.5.2 System performance

We adopt a sharing strategy in which some parts of the circuit work in a sequential mode (see Section 5.3) for saving power consumption and fit into a Virtex 4: a fully parallel implementation can fit in a modern FPGA (Virtex 5 or 6). This leads to a limitation in the processing speed. For both cores we share the input filters stages and limit the data rate to one pixel every two clock cycles for stereo and to one pixel every

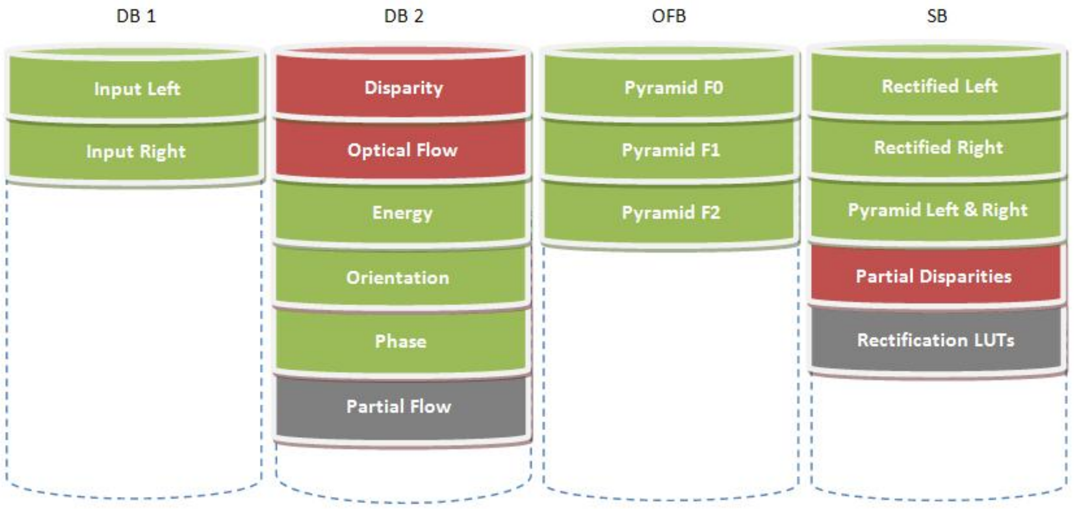


**Figure 5.5:** Memory mapping for a XircaV4 platform. Note that an MCU is mandatory for a parallel access to all this data.

three clock cycles in the optical flow. The two cores run in parallel with different data paths; the worst one is the optical flow for which the processing takes:

$$C = 1 + sI + (s+3)I \tag{5.1}$$

Where I is the image resolution (width by height) and s is the scale factor and depends on the number of scales as indicated in (3.3). First term in (5.1) is related to the pyramid computation operation (first operation before starting the flow computation) and the second one is related to the processing engine: the number 3 is because the processing core takes 3 clock cycles because of sharing the filter bank between input frames. Circuit latency is negligible compared to the number of cycles of the whole processing data-path. With equations (5.1), (3.3) and the clock frequency we can calculate the frame rate for different image sizes simply dividing frequency by C; if we choose the maximum clock frequency allowed by the circuit (42 MHz) we obtain a frame rate of approximately 28 fps for a 512 by 512 image. Table 5.4 reports the frame rate for different image resolution and number of scales. The frame rate of the first row (a mono-scale version) is limited by the PCIe interface bandwidth that for our platform has 130 MB/s (empirical measure). For this reason compression techniques or more advanced schemes as the condensation mechanism presented in [99] can be realized in order to reduce the transferred data and improve the frame rate. Due to the novelty of this

**Table 5.4:** Frames per second at different image size and number of scales (results experimentally measured with the balanced system running). Clock set to 42 MHz. A mono-scale processing will be faster but is not accurate for large/fast movements and significant disparities.

| Size | # scales | Frames per second |
|---|---|---|
| 512x512 | 1 | 53.4 |
| 512x512 | 4 | 28.6 |
| 640x480 | 4 | 24.2 |
| 640x480 | 5 | 24 |
| 720x576 | 4 | 17.8 |

work it is difficult to make a direct comparison with the state of the art. There are no other approaches including all these different primitives on the same chip. Nevertheless we can remark the main features of the system and its significant computation power. Existing systems as [20] include a considerable number of parallel processing units. This approach was a processing engine with a computational power of 12.4 Giga-Operations per second (GigaOPS). Our system, with 2221 parallel processing units and a frequency of 42 MHz is able to compute 93.3 GigaOPS (see Table 5.3). Furthermore if we take in account the low power consumption, near 7.2 W (estimation obtained by Xilinx Xpower analysis), we have 12.9 GigaOPS/W. Other engines as DaVinci from Texas Instruments or Blackfin from Analog Devices obtain respectively 4 MegaOPS/W (4.8 MIPS / 1.2 W according to datasheet [100]) and 2.16 GigaOPS/W (1512 MMACs / 1.4 W according to datasheet [101]). Their computation power is far below the our system performance, therefore several of these engines would be necessary to process all the primitives included in our system. This would lead to a multi-core implementation of high complexity. Taking into account that the low-level vision represents the higher computational cost in a vision system, it becomes clear the importance of this high performance vision engine.

### 5.5.3 Results in real and synthetic sequences

As we saw the system processes at the same time, optical flow, disparity, energy, phase and orientation. In the literature is not available a sequence with ground truth for

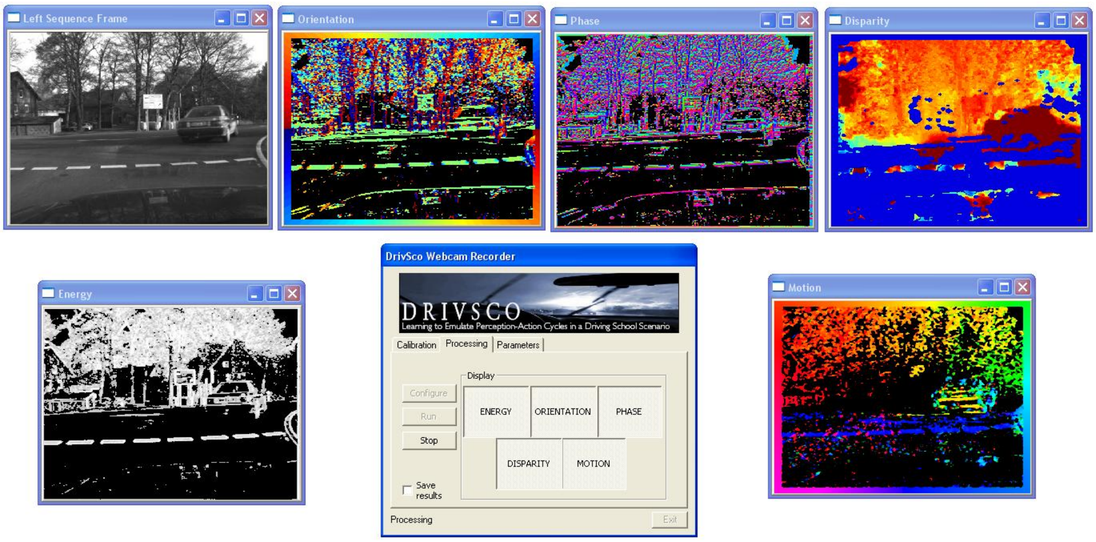


**Figure 5.6:** Screen shot of software interface while running a full processing in a driving scenario. First row, left to right, input left image, local orientation, phase information and stereo information. Second row, local energy, user interface and optical flow direction according to the color code at the image frame. Note that all these features are running simultaneously on the same chip.

all these primitives. So we adopt existing data sets for separate features, especially stereo and optical flow. Data sets and their respective ground truth are available in [2] where is possible a comparison with the state of the art approaches. For each single modality we achieve the same accuracy detailed in previous chapters. Due to the limitation in hardware recourses (Table 5.3), for the XircaV4 it is not possible to operate any kind of post-processing or regularization on-chip, therefore we have raw results worst in quality than the evaluation rates reported in [2]. Nevertheless, just using simple techniques such as median filters (see Appendix 7) and a high performance architecture we can significantly reduce the error rate (Table 5.3). Therefore, median regularization operations will be included, if more resources are available (as in the high performance version on larger devices). Fig. 5.6 gives a qualitative representation of a real sequence result for all modalities running in parallel. It is possible to see also the software interface; in the main window the user can choose on the loop the primitives to display.

### 5.5.4 Results discussion and potential applications

This work is the result of research and development in FPGA platforms. We can underline the long time-to-market for this complex architectures especially if compared with the methodology adopted in GPUs [102]. The design strategy addresses a fine pipeline architecture in opposition to a multi-core implementation, typical in modern approaches (GPUs, general purpose processors). Our approach produces large pipelines affording glitches and reduces the power consumption. Despite working at low frequencies (42 MHz for the balanced implementation) the architecture can achieve a great processing power. Advantages of a multi-core architecture can be added in more expensive chips: with a throughput higher than one pixel per clock cycle. Novelty of work does not allow a comparison with other approaches but we have estimated and compared the achieved on-chip computing power with other alternatives. Our system can be included in smart cameras for industrial applications where our device is able to handle massive pre-processing operations such as local features, disparity and motion. A secondary stage (based on commodity processors) could process the produced data stream for a specific application (pick and place, quality inspection, motion analysis, etc). Currently in the industry we have low level vision solutions integrated as smart cameras specialized or image processing tools [103, 104]. Local features information (mainly energy for edge detection) is provided for many systems. Our system improves current systems in terms of processing speed. There are systems that provide disparity computation information [105, 106]. These approaches mainly represent a low accuracy processing based on simple algorithms as Sum of Absolute Differences (SAD). Our approach overcomes these existing solutions in terms of accuracy and robustness (a critical issue for industrial applications) because the adopted model is based on phases. Finally, no smart cameras are extracting optical flow in real-time. As alternative, we could compute it in software using a PC with vision libraries [107, 108] but due to the large processing time required, this approach drastically constraints the final feasible applications. The presented architecture is able to provide at the same time diverse primitives easy to integrate with a software application running on a PC. This facilitates the development of a powerful vision system for many industrial applications, from vision inspection systems (working as a high performance smart camera), to more complex system (for instance for dynamic robotics applications). In fact, with this architecture is even possible to control complex and novel industry processes such as the fluid control that require real-time constraints and that are not feasible for state of the art industry processors [109, 110].

## 5.6 Conclusions

A complete study for hardware design techniques is presented. The main purpose is the presentation of different strategies for a complex low level vision processing engine with different requirements and available resources. This work demonstrates that with specific techniques it is possible to target different system specifications. Limitations are defined by user and depends on available technology, cost and time-to-market. Strict requirements need obviously more developing time. In order to proof our hypothesis we choose a complex algorithm of the computer vision field especially well suited for real-world applications due its robustness to noise or illumination changes. The algorithm adopted is a high performance phase-based approach previously studied in various works [54].

The chapter describes a high performance low level vision engine implemented on a single chip. It computes in real-time different vision modalities: motion, stereo and local contrast descriptors such as energy, orientation and phase. Integrating processing cores of these different modalities on a single architecture is a complex task. The simple adding of the single modality cores would require 3268 parallel processing units and an unaffordable memory access rate. This work describes resources sharing techniques that allow the implementation of specific architectures addressing different performance vs. cost trade-offs. A balanced system uses only 2221 parallel processing units and represents an approximate hardware resource reduction of 32% with respect to the simple add-on of the different cores, while maintaining high accuracy: 50% of performance reduction in terms of Megapixels per second and only 28% of accuracy reduction (joined stereo and optical flow).

A lot of parameters can be set for an *ad hoc* solution. Detailed studies for bit-width, sharing strategies and algorithm modifications are illustrated in this work. A balanced architecture that achieves real-time in 512x512 images is presented with a deeper detail level. At the same time two further designs are evaluated in order to fulfil other user requirements as low power/resources consumption or high performance. The high performance implementation computes 165 GigaOPS (with a system clock at 50 MHz) at low power consumption (approximately 30 GigaOPS/W). This represents an outstanding processing power vs. energy consumption trade-off. Rather than a single implementation, this work presents a design and integration strategy, it evaluates different resources sharing techniques and illustrates the versatility of FPGA based system that can be easily adapted to diverse target platforms to fulfil cost vs. performance specifications. The high number of parameters and their combination lead to a very complex study, but it is useful to illustrate the versatility of the designs and its easy adaptation to different design constraints. The methodology adopted in this example can be generalized to many other processing tasks. The experience of the designer plays an important role in this cases and affects the time-to-market and the quality of adopted solutions. To the best of our knowledge this study and this architecture represent a novel contribution. In fact, it represents one of the most complex vision engines ever designed on a single FPGA device. It can cover an important role as well in industrial applications as research fields. The natural evolution of this novel low-level vision engine in the oncoming future is to study its integration with with middle-level vision in hybrid hardware/software co-design. Further optimization in pipelines and synthesis efforts can be adopted for future improvements in the maximum clock frequency. This future study implies a larger hardware utilization but allows in the same time a better throughput.

# 6

# Discussion

## 6.1 General work motivation

The main aim of this work is the study and the implementation of a low level vision system on chip. Preliminary studies and experiments demonstrate that a phase-based approach is a good candidate for a complex and accurate multi-modal vision engine. Despite the algorithm is not the best option in terms of accuracy amongst all the works proposed in the literature, our choice is justified not only by the robustness of the phase in unconstrained scenarios but also by the general purpose of the phase for different features detection as stereo, local energy, local orientation and local phase besides optical flow computation, ego-motion, phase congruency, etc. [54]. Thus this multi-orientation algorithm allows addressing all these vision cues computations with the possibility of sharing a significant amount of circuits for the early and middle vision processing. Previous works address the single modality problem (motion or stereo) with very high frame rates or accuracy but the majority of them do not conjointly study the complementary benefits of computation accuracy and high frame rates (for instance in optical flow processing) on the same approach. For this purpose the work methodology is based on a fine grain pipeline and on a novel multi-scale architecture. The first one is rarely used in literature (since it requires a very structured design) and it is substituted for multi-core approaches, more expensive in terms of power consumption. The second one, as far as we know, has been never implemented in reconfigurable devices. It represents a novel contribution and the definition of this architecture can be reused for different kinds of image processing models or algorithms. After an exhaustive study and the implementation of different single modalities such as stereo (Chapter 3), optical flow (Chapter 4) and local descriptors (Chapter 3), the work focuses on the study of a global low level vision engine that assembles all the previous architectures. This ambitious aim has been never afforded in the literature and represents a novel contribution to the research and the development of architectures based on FPGA devices. The design of the system includes the study and the definition of many parameters and architecture choices that can generate a widespread range of solutions. We demonstrate in Chapter 5 that an architecture can be defined following different user requirements and its final implementation will be significantly different in terms of hardware utilization and power consumption. Our initial aim of a global low level vision engine in the same chip has been completely solved and an accuracy vs. costs trade-off solution is implemented as a co-processing board. The system computes multi-scale optical flow, stereo and local contrast descriptors (energy, orientation and phase) at 28.6 fps (image resolution of 512x512). The balanced implementation offered for a Virtex 4 chip consists of 2221 parallel processing units on the same chip, computing 92.3 GigaOPS (with a system clock at 42 MHz) at low power consumption (approximately 12.9 GigaOPS/W). A further high performance architecture for Virtex 5 has been studied and achieves a throughput of 165 GigaOPS (30 GigaOPS/W). Such solutions may be of interest in many application fields as medical imaging, robotics, industrial and automotive areas.

## 6.2 Future works

As we know the low level vision stages require the major part of the computational power for a vision system. With this massive processing implemented in a single chip it will be possible integrating diverse middle or high level vision tasks at a reduced cost. These tasks may be addressed with the co-design HW/SW of novel and powerful embedded systems: current FPGAs include embedded processors as PowerPCs. Furthermore it is possible to build more complex visual descriptors based on second-order motion properties. For example the perception of motion in depth, the computation of heading, ego-motion or detection of Independent Moving Objects (IMOs) in which are absolutely necessary the first-order primitives computed by our low level vision system. In order to improve the system portability it is possible the development of a stand-alone board starting from the defined architecture. New interface with cameras and video output can be included in the adopted platform for robotics issues. In terms of circuitry, further optimizations of logic and major efforts in synthesis can improve the working clock frequency. This implies more hardware utilization but in the same time a major throughput.

## 6.3 Publication of results

Our research work has been evaluated in the framework of international conferences and scientific journals (with impact factor on the JCR).

[1] M. Vanegas, M. Tomasi, J. Díaz, E. Ros, Multiport abstraction layer for FPGA intensive memory exploitationapplications, accepted to Journal of System Architecture (2010)

[2] M. Tomasi, F. Barranco, M. Vanegas, J. Diaz, E. Ros, High performance optical flow architecture based on a multiscale and multi-orientation phase-based model, accepted to IEEE Trans. on Circuits and Systems for Video Technology (May 2010)

[3] M. Tomasi, M. Vanegas, F. Barranco, J. Diaz, E. Ros, Real-time architecture for a robust multiscale stereo engine, submitted to IEEE Trans. on Image Processing (submitted in 2009)

[4] M. Tomasi, F. Barranco, M. Vanegas, J. Diaz, E. Ros, Fine grain pipeline architecture for high performance phase-based optical flow computation, submitted to Journal of System Architecture (Submitted in 2009, reviewed with minor revision)

[5] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, Arquitectura multiescala de cálculo de flujo óptico basado en la fase, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009, 2009, pp. 295–304, IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009

[6] M. Tomasi, J. Díaz, E. Ros, Real time architectures for moving-objects tracking, in: (ARC2007), Lecture Notes in Computer Science, Vol. XXX, 2007, pp. 365–372

[7] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, A novel architecture for a massively parallel low level vision processing engine on chip, ISIE2010. Accepted to IEEE Internetional Symposium on Industrial Electronics, Bari (Italy) (July 2010)

[8] F. Barranco, M. Tomasi, M. Vanegas, S. Granados, J. Diaz, Entorno software para visualización y configuración de procesamiento de imágenes en tiempo real con plataformas reconfigurables, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009., 2009, pp. 327–336, IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009

## 6.4 General Scientific Framework

This scientific work has been done and funded by the European Project DRIVSCO: Learning to emulate perception action cycles in a driving school scenario (IST-016276-2). This has represented an excellent collaborative framework with diverse research groups at other European Universities and research institutions. The presented work represents the major contribution of the University of Granada in this DRIVSCO consortium. Therefore, a high responsibility in obtaining timely the planned results was necessary during the whole investigation process. Besides the required technical reports, presentations for EU scientific reviews, a final demo (proof of concept) was required and implemented. The effort invested in this demo is significant but allows easy evaluating of the system performance and also facilitates the dissemination of results beyond a pure scientific scenario, towards industrial future collaborations and also impact in the media (newspapers, TV, etc).

The work required a very close collaboration with different researchers at the lab. It required a lot of efforts in making the different modules and designs easy to be integrated in other architectures and also the inclusion of modules designed by these collaborators. This has required high collaborative and coordinated work with the working team.

## 6.5 Main contributions

We now summarize the main contributions of the presented work:

- Among different algorithms in computer vision a phase-based approach has been chosen. A comparative study of robustness is done justifying that the approach is suitable for an accurate low level vision computation.

- An algorithm modification is adopted in order to adapt it to a proper hardware implementation. An exhaustive study of accuracy vs. performance trade-off is done.

- The system has been designed as a deep finely grained pipelined datapath (with several superscalar stages) to maximize the processing parallelism. This design strategy is very exigent in terms of synchronization, external memory support and data dependencies.

- A novel multi-scale architecture with warping is presented. Comparison with mono-scale approaches confirms that this algorithm extension is useful for accuracy improvements and working range enhancement. To the best of our knowledge, in hardware this approach has never been addressed before.

- The defined architecture has been applied to different single modality algorithms as stereo, optical flow and local descriptors (energy, orientation and phase). A comparative evaluation study with the state of the art rates the presented implementation as a very competitive one with respect to existing hardware solutions.

- A global early processing system including optical flow and stereo has been implemented. A lot of parameters are analyzed in order to relate structural changes (design decisions) with their impact onto the final system performance. A proper performance vs. cost trade-off is obtained.

- The study of system parameters conclude with an interesting comparison of three different architectures: a low cost one, a balanced one with a good performance vs. cost trade-off, and a high performance version.

- The implementation of the balanced solution in a co-processing board achieves a very high computational power. The system consists of 2221 parallel processing units on the same chip, computing 92.3 GigaOPS (with a system clock at 42 MHz) at low power consumption (approximately 12.9 GigaOPS/W). This represents an outstanding processing power vs. energy consumption trade-off.

- The high performance architecture studied achieves up to 165 GigaOPS (30 GigaOPS/W), but due to its high cost, it can be adopted only in few critical applications.

# 7

# Appendix A

## 7.1 Optical Flow and Stereo regularization

In order to improve results in accuracy, it is possible introduce multiple regularization techniques to properly average values and reject noisy estimations of the algorithm. Many techniques are possible such as simple values averaging, complex iterative schemes such as anisotropic diffusion, spatial Kalman filtering, etc. [34, 116]. A good trade-off between robustness to noise and complexity is possible with a simple median filter. This has motivated our choice as regularization method for the hardware system. Nevertheless, note that depending on the size of the median filter, the spatial accuracy can be affected and the mean error increases. In this section we study the variation of error with different kinds of median filters. A final hardware implementation will depend on available resources and from accuracy level.

## 7.2 Regularization on Optical Flow

To the original algorithm we apply a median filter: along each scale and after the merging of values. The hierarchical process uses four scales of the Gaussian pyramid and Gabor filters with 11 taps. Evaluation tests are made with a software simulator of the hardware system (described on MATLAB code) and for different input sequences. Results on different tables and figures show that starting from a kernel size larger than a 5x5 window over-regularize the results, decreasing the accuracy of the system. The median filter does not take into account rejected low confidences (considered as invalid values or NaN) and calculates the median value only over the valid ones. Note that for

**Table 7.1:** Errors for the Yosemite sequence. HW Matlab model.

| Median | AE | Std | Dens |
|---|---|---|---|
| 0 | 10.93 | 13.7 | 70.37 |
| 3x3 | 4.87 | 9.47 | 83.5 |
| 3x3 (2 times) | 4.5 | 9.07 | 89.12 |
| **5x5** | **4.24** | **8.95** | **89.6** |
| 7x7 | 4.58 | 10.15 | 94.75 |
| 9x9 | 4.91 | 10.48 | 96.82 |
| 11x11 | 5.68 | 12.7 | 98.11 |

**Table 7.2:** Errors for the Urban2 sequence. HW Matlab model.

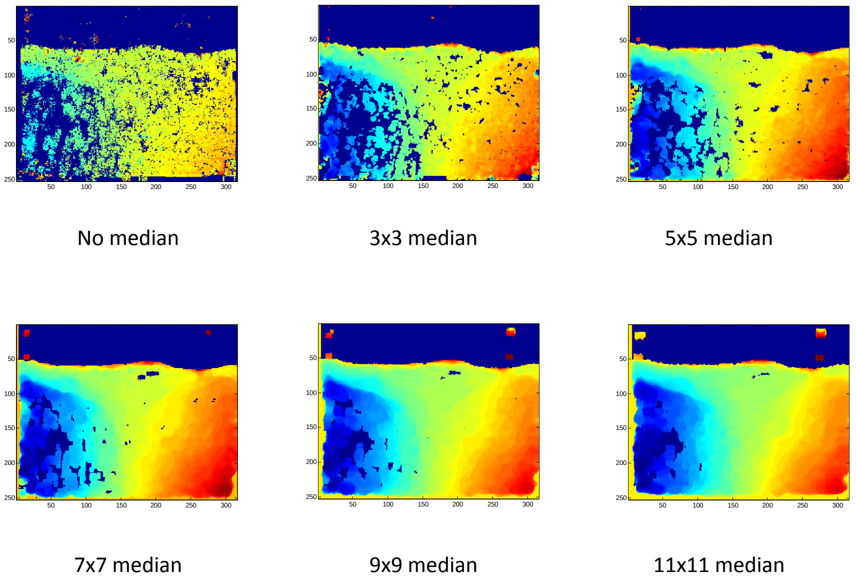
| Median | AE | Std | Dens |
|---|---|---|---|
| 0 | 22.08 | 26.54 | 66.04 |
| 3x3 | 17.38 | 23.26 | 72.52 |
| **3x3 (2 times)** | **16.95** | **22.73** | **79.54** |
| 5x5 | 18.08 | 24.13 | 80.52 |
| 7x7 | 19.34 | 24.82 | 89.2 |

this reason the density increases with the size of the filter. Optical flow quantitative results are offered only for sequence with ground truth, available in [2]. We process also a real driving sequence from [117], in this case is possible evaluate only the qualitative results.

As shown in the figures the quality improvement is remarkable. The best choice is the 5x5 kernel or alternatively a coarse approximation based on two consecutive 3x3 median filters operations.

## 7.3 Regularization on stereo

For the stereo algorithm we repeat the same procedure. According to the results found for the optical flow, we demonstrate that median filter kernel size has an inflection point. Better results are achieved with the cascade of two 3x3 median filters. Note the great improvement from raw results to the regularized ones, especially in real sequences (Fig. 7.6).



No median

3x3 median

5x5 median

7x7 median

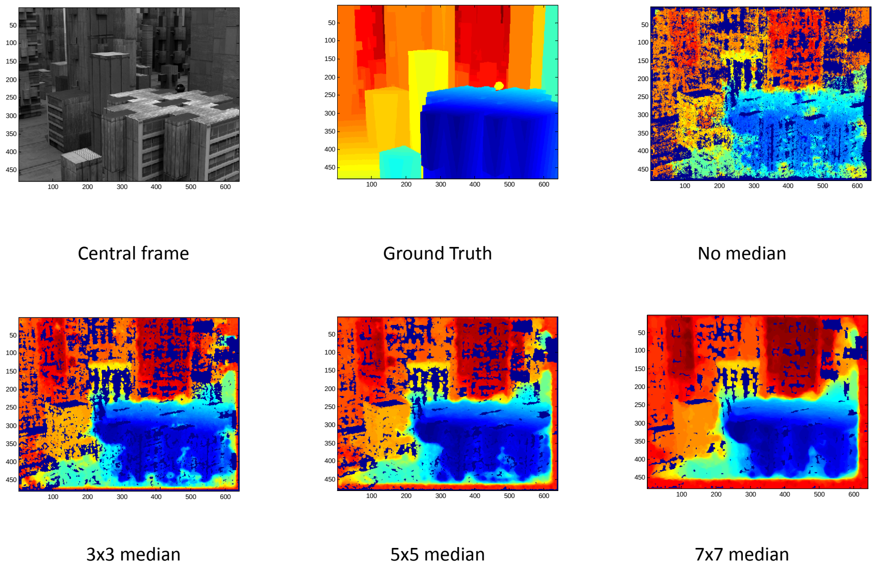
9x9 median

11x11 median

**Figure 7.1: Optical flow regularization: yosemite.** - Vx component for Yosemite optical flow with a HW model computation: red values represent a positive velocity, blue a negative.

## 7.4 Hardware implementation

After a preliminary study we evaluate different median implementation approaches. The best results are found using 5x5 kernels (which requires a large number of FPGA resources) and by the cascade 3x3 filter approach. The bigger computational effort is represented by the sort operation. It obviously grows with the size of the filter. In order to reduce this problem as stated in [118], a 3x3 median filter can be applied recursively and is equivalent to a bigger one depending from the number of iterations. For this reason we implement a median filter composed by two cascade 3x3 median filters. A larger number of cascade filters could be employed in order to better approximate the 5x5 median filter but we do not found relevant improvements in our results thus we do not increase the number of filters beyond two. Two different schemes for the 3x3 median filters have been implemented. The first one is a classic method with a nine



Central frame

Ground Truth

No median

3x3 median

5x5 median

7x7 median

**Figure 7.2: Optical flow regularization: urban2.** - Vx component for Urban2 optical flow with a HW model computation: red values represent a positive velocity, blue a negative.

values sorting and a tree based architecture that swap values in N clock cycles. Properly pipelining techniques are used to translate this delay into latency avoiding performance degradation at cost of large number of flip-flops utilization. The second alternative uses the properties of the sliding windows as stated in [119] and operates with only sorting of 3 values. In this second case, synthesis tools instantiate a multiplexer. Table 7.5 reports the hardware utilization for the two methods.

## 7.5 Conclusions

Experiments with a regularization filter demonstrate its key role in an iterative hierarchical model as the phase-based one adopted in this work to reduce model error due to matching outliers. Note that wrong values produce wrongly warped images and these errors propagate along the image pyramid. The application of a simple median

**Table 7.3:** Errors for the Tsukuba sequence. HW model.

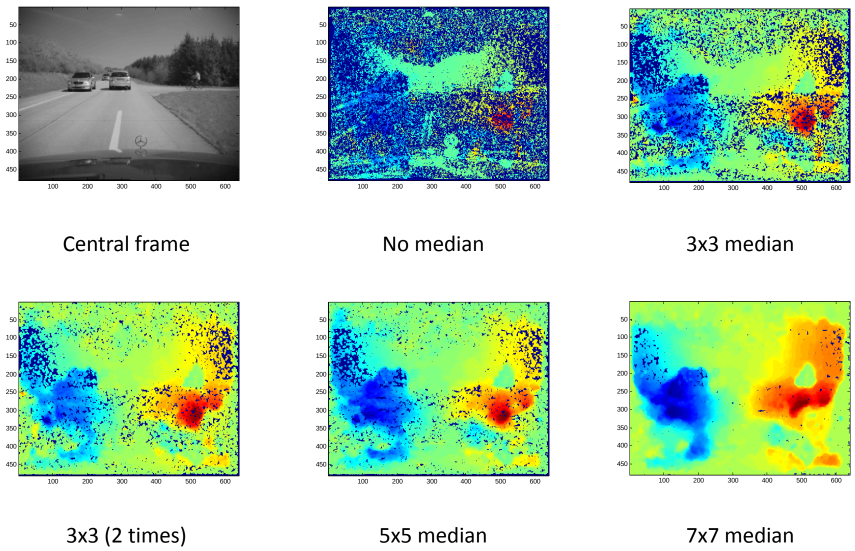
| Median | MAE | Std | Dens |
|---|---|---|---|
| 0 | 1.06 | 1.43 | 97.22 |
| 3x3 | 0.98 | 1.39 | 98.44 |
| **3x3 (2 times)** | **0.96** | **1.4** | **99.14** |
| 5x5 | 1.09 | 1.55 | 99.22 |
| 7x7 | 1.15 | 1.68 | 99.74 |
| 9x9 | 1.32 | 1.9 | 100 |

**Table 7.4:** Errors for the Venus sequence. HW model with 5 scales.

| Median | MAE | Std | Dens |
|---|---|---|---|
| 0 | 1.54 | 2.64 | 97.91 |
| 3x3 | 1.62 | 2.85 | 100 |
| **3x3 (2 times)** | **1.6** | **2.8** | **100** |
| 5x5 | 1.96 | 3.48 | 100 |
| 7x7 | 2.44 | 3.98 | 100 |
| 9x9 | 2.98 | 4.46 | 100 |

**Table 7.5:** Hardware utilization for two different implementations of a 3x3 median filter. The design run in a Xilinx Virtex4 xc4vfx60. The edif netlist generated by the DK Suite 5.

| Circuit | Slices | LUTs | DSP48 | BRAM | Max. Freq. |
|---|---|---|---|---|---|
| Median Sorting 3 32 bit | 1554(6%) | 1848(3%) | 0 | 8(3%) | 125 MHz |
| Median Sorting 3 8 bit | 570(2%) | 697(1%) | 0 | 2(1%) | 128.5 MHz |
| Median Sorting 9 32 bit | 1443(5%) | 1903(3%) | 0 | 8(3%) | 131.5 MHz |
| Median Sorting 9 8 bit | 616(2%) | 827(1%) | 0 | 2(1%) | 159.2 MHz |



Central frame

No median

3x3 median

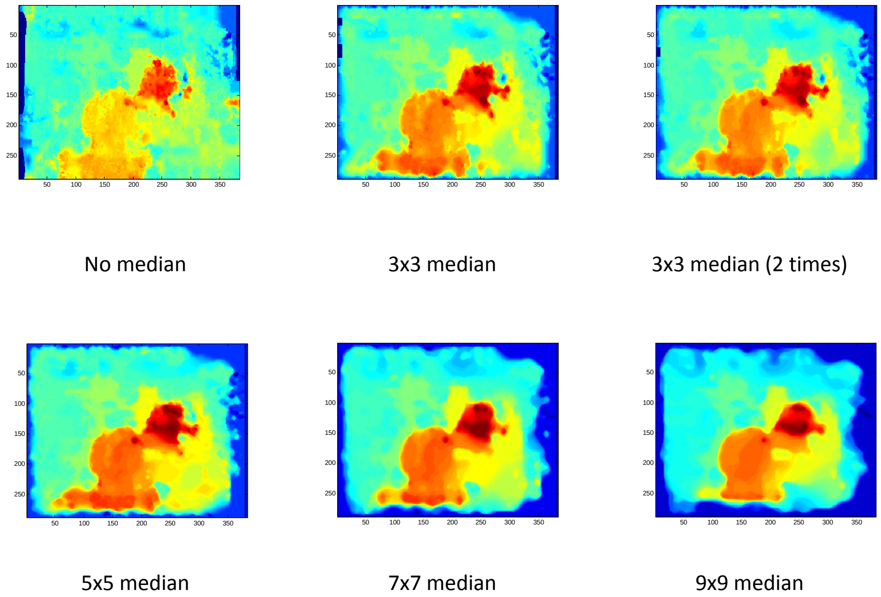
3x3 (2 times)

5x5 median

7x7 median

**Figure 7.3: Optical flow regularization: real sequence.** - Vx component for a real driving sequence optical flow with a HW model computation: red values represent a positive velocity, blue a negative.

filter improves accuracy and density as shown in synthetic and in real scenarios. The quantitative evaluations have been estimated for synthetic sequence with ground truth validating our qualitative hints. Quantitative results indicate that the optimum solution for the majority of analyzed cases is the cascade of two median filters with a 3x3 kernel. Different hardware implementations of a 3x3 median filter have been studied. A recursive application of the same 3x3 median is equivalent to a bigger median filter: depending from the application and the chip size is possible replicate the implemented filter to obtain optimum results. In our case we adopt for the phase-based algorithms the optimum solution found with a previous study. Further replications of the median filter mean a bigger hardware utilization that is not translated in a significant accuracy improvement. The median based regularization can be applied also to different hierarchical approaches. Implemented median filter can be used for these other algorithms after a proper preliminary study as a solution with a good trade-off performance-effort.



No median

3x3 median

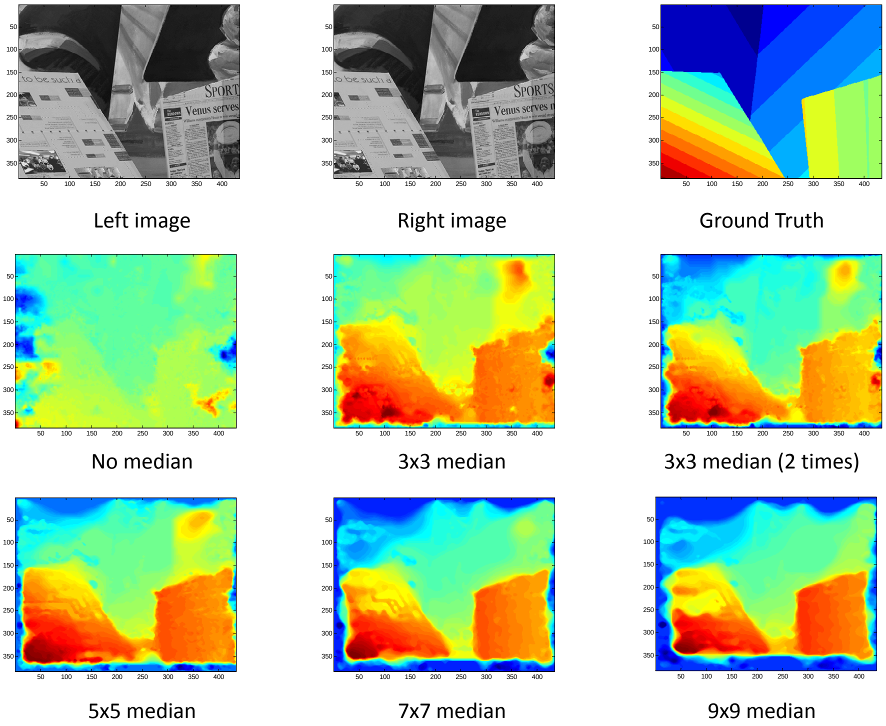
3x3 median (2 times)

5x5 median

7x7 median

9x9 median

**Figure 7.4: Stereo regularization: Tsukuba** - Regularization for the well-known "Tsukuba" sequence: red values represent closer objects.



Left image

Right image

Ground Truth

No median

3x3 median

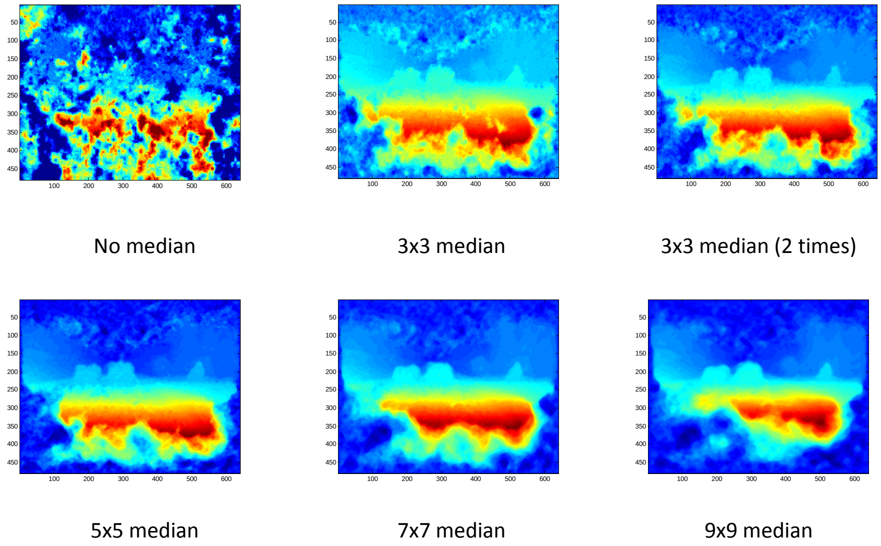
3x3 median (2 times)

5x5 median

7x7 median

9x9 median

**Figure 7.5: Stereo regularization: Venus.** - Regularization for the Venus sequence: red values represent closer objects.



No median

3x3 median

3x3 median (2 times)

5x5 median

7x7 median

9x9 median

**Figure 7.6: Stereo regularization: real sequence.** - Regularization for a disparity calculation of the driving scene of Fig. 7.3: red values represent closer objects.

# 8

# Appendix B

## 8.1 Co-processing platform: software and hardware interfaces

This appendix explains the global environment of the system implementation. In fact in order to obtain such a complex system diverse designers have participated. As previously asserted in the introduction, our work focuses mainly on processing cores and the multi-scale architecture. However for the complete implementation we need other important parts developed by other engineers of the group. For the sake of clarity in this appendix we give a brief description of this parts (Memory Controller Unit, software interface) and of the system platform manufactured by Seven Solutions [61]: spin-off company from the University of Granada. The final system is represented by a co-processing platform connected to a host with a stereo pair of cameras (Fig. 8.1).

## 8.2 FPGA board

The XircaV4 platform used for our system implementation includes as input/output interfaces:

- 1 PCIe 1x interface
- 2 ethernet ports
- 1 RS232

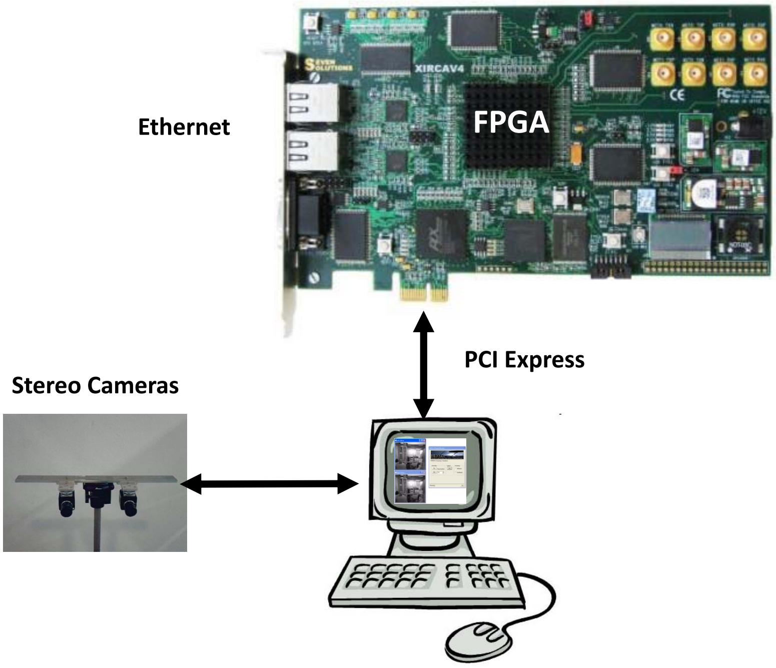


**Figure 8.1: Co-processing platform.** - The XircaV4 is used as a co-processing board and communicates with host through the PCIe interface. Stereo pair of cameras can be adapted to user requirements.

- 1 jtag port
- 2 rocket IO transceivers (SMA)
- 20 expansion pins

It includes a Xilinx Virtex 4 xc4vfx100 FPGA and can include every FPGA with package ffg1152. The platform represent a powerful co-processing board and it can be configured as stand-alone board thanks to its independent DC alimentation. For processing purpose it includes 2 different oscillators (125 MHz, 100 MHz) and one more for the PCI interface at 66 MHz. Furthermore this platform is suitable for video processing thanks to its amount of external memory:

- 4 SRAM memories of 72 Mbits (2Mx36)
- 2 DDR memories of 512 Mbits

- 4Kb IIc EEPROM

One of the advantages is the presence of many memory banks with independent accesses, it allows parallel memory operations. One of the problems is the PCIe bandwidth, experimentally we measure 130 MB/s and this is not sufficient for a fast massive process. Our system saturates the bandwidth at 44 fps for VGA images.

## 8.3 Memory Controller Unit

The Memory Controller Unit (MCU) has been designed by M. Vanegas and is fully described in [59]. It controls the external SRAM banks through different Abstract Access Ports (AAPs). Taking into account that hardware designs on FPGA technology are generally slower than memory chips, the shared memory scheduler uses communication port arbitration based on a fast memory controller with AAPs running slower than the memory controller. Hence the whole design has two clock domains, thus the hardware accessing to data through ports can be concurrent in the AAP clock domain if the ratio between clock frequencies (memory controller frequency over the AAP frequency) is at least the number of AAPs. To achieve a better level of abstraction, the MCU is provided with two kinds of AAPs for reading and writing over memory and consists of several ports (readers and/or writers) depending on the system requirements. The interface scheme is shown in Fig. 8.2, where a block diagram of the MCU with two channels (three request interfaces) is presented. The port interface is based on standard FIFO structures.

AAP arbitration incorporates a scheduler attendant based on hierarchical priority for ensuring global attendance to the same bandwidth per AAP. This does not simply a multiplexing scheme but rather a continuous uniform bandwidth distribution per access. The scheduler attendant just ensures access to memory from all AAPs without trading emerging behaviour necessities of upper levels in the system.

## 8.4 Software interface

The co-processing board solves only processing tasks but in our case does not have a direct connection with input cameras or output devices. Thus a host PC with a software interface is needed to perform this task. The software used (designed by F. Barranco)

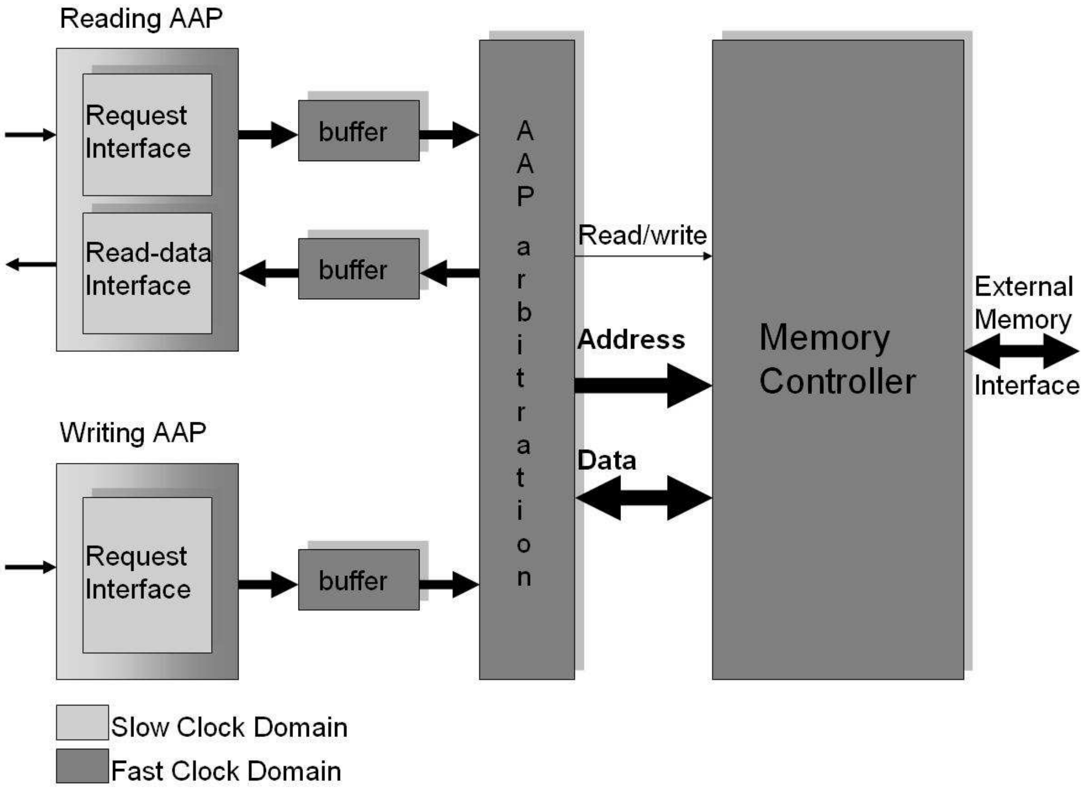


**Figure 8.2: MCU interface scheme.** - Block diagram of MCU architecture for a 2 AAPs configuration case.

is available in `http://code.google.com/p/open-rtvision/` and has been developed in the Granada University for the DRIVSCO project. The application runs in the computer, acquiring the images from the cameras and sending this processed information to the co-processing platform using the PCIe or the PCI interfaces depending from platform. Once, the hardware computation is completed, the application reads the results from the board memory and post-processes the results for an appropriate visualization or storage. Software and hardware interface libraries provides the communication between the computer and the platform. This communication is implemented using two different handshaking protocols: the first one is needed for the parameter pass and the communication channel establishment, the second one performs the transmission using a double buffer or ping-pong scheme. The list of parameters is useful for the processing and for the reuse of the circuits with different environment or camera configurations.

In this way we fully benefits the advantages of a reconfigurable device. The software interface has been developed using the Visual Studio .Net 2003 and 2005 environments. Furthermore, it uses different libraries and software packages for the optimization of the data management: OpenCV, OpenMP, IPP. Further optimization of the code are used for the fulfilling of a real-time communication: minimization of the read and write operations to disk and the cache faults, loops unroll and use of threads for the recording computation.

## 8.5 Camera set-up

Different stereo pair cameras have been adopted to validate the architectures. For all of them an off-line processing for the undistortion and calibration LUTs has been done. The calibration tool for the MATLAB environment has been developed by Jarno Ralli for the DRIVSCO [22] framework. In particular, these three different setups have been validated:

1. Philips SPC 1300NC webcams
2. AVT Guppy F036C firewire cameras (high frame rate)
3. Dalsa Coreco cameras with camera-link interface (high resolution)

They represent different solutions suitable for any kind of application. For low cost applications the Philips webcams are the better choice, for mono-scale optical flow algorithms we used the Guppy cameras (90 fps) and for the validation in the DRIVSCO project we used the Dalsa cameras (mounted also for a car system prototype).

# Bibliography

[1] J. Barron, D. Fleet, S. Beauchemin, Performance of optical flow techniques 12 (1) (1994) 43–77. ix, 6, 25, 55

[2] Middlebury vision, Website, `http://vision.middlebury.edu/stereo/` (2010). x, 6, 8, 14, 49, 68, 93, 104

[3] Xilinx, Website, `http://www.xilinx.com` (2010). x, 40, 42, 57, 60, 63, 79, 86

[4] Handel-c manual reference, Website, `http://www.agilityds.com/products/c_based_products/dk_design_suite/handel-c.aspx` (2010). x, 40, 79

[5] Digikey webpage, Website, `http://www.digikey.com/` (2010). xi, 90

[6] E. R. Kandel, J. H. Schwartz, T. M. Jessell, Principles of Neural Science, 4th Edition, McGraw-Hill Medical, 2000.
URL `http://www.worldcat.org/isbn/0071120009` 2, 4

[7] C. Weems, Architectural requirements of image understanding with respect to parallel processing, Proceedings of the IEEE 79 (4) (1991) 537 –547. `doi:10.1109/5.92046`. 2, 3

[8] C. C. Weems, S. P. Levitan, A. R. Hanson, E. M. Riseman, D. B. Shu, J. G. Nash, The image understanding architecture, International Journal of Computer Vision 2 (3) (1989) 251–282. 3

[9] A. Choudhary, J. Patel, N. Ahuja, Netra: a hierarchical and partitionable architecture for computer vision systems, Parallel and Distributed Systems, IEEE Transactions on 4 (10) (1993) 1092 –1104. `doi:10.1109/71.246071`. 3

[10] Web page of mobile eye, Website, `http://www.mobileye.com` (2010). 3

[11] K. R., M. B., FPGA coprocessing architectures for military applications, Military embedded systems. 3

[12] European machine vision association web page, Website, `http://emva.org` (2010). 3

[13] M. Anguita, J. Diaz, E. Ros, F. Fernandez-Baldomero, Optimization strategies for high-performance computing of optical-flow in general-purpose processors, Circuits and Systems for Video Technology, IEEE Transactions on 19 (10) (2009) 1475–1488. 4, 56, 68, 69

[14] Nvidia web page, Website, `http://www.nvidia.com/object/cuda_home.html#` (2010). 5

[15] D. Scharstein, R. Szeliski, A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, Inter. J. of Computer Vision 47 (1-3) (2002) 7–42. 6, 49

[16] M. Z. Brown, D. Burschka, G. D. Hager, S. Member, Advances in computational stereo, PAMI 25 (2003) 993–1008. 6, 51

[17] M. F. Tappen, W. T. Freeman, Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters, in: In ICCV, 2003, pp. 900–907. 6

[18] D. Bilsby, R. Walke, R. Smith, Comparison of a programmable dsp and a FPGA for real-time multiscale convolution, in: High Performance Architectures for Real-Time Image Processing (Ref. No. 1998/197), IEE Colloquium on, 1998, pp. 4/1 –4/6. 7

[19] J. Díaz, E. Ros, R. Carrillo, A. Prieto, Real-time system for high-image resolution disparity estimation, IEEE Trans. Image Processing 16 (1) (2007) 280–285. 7, 13, 39, 51, 52, 76

[20] J. Díaz, E. Ros, R. Agís, J. L. Bernier, Superpipelined high-performance optical-flow computation architecture, Computer Vision and Image Understanding 112 (3) (2008) 262–273. 7, 24, 56, 66, 67, 68, 69, 76, 92

[21] S.-K. Han, S. Woo, M.-H. Jeong, B.-J. You, Improved-quality real-time stereo vision processor, in: VLSID '09: Proc. of the 2009 22nd Inter. Conf. on VLSI Design, 2009, pp. 287–292. 7, 51, 52

[22] Drivsco project, Website, `http://www.pspc.dibe.unige.it/~drivsco/` (2007). 9, 38, 49, 117

[23] S. S. Beauchemin, J. L. Barron, The computation of optical flow, ACM Comput. Surv. 27 (3) (1995) 433–466. `doi:http://doi.acm.org/10.1145/212094.212141`. 12, 55

[24] A. Bruhn, J. Weickert, T. Kohlberger, C. Schnoerr, A multigrid platform for real-time motion computation with discontinuity-preserving variational methods 70 (3) (2006) 257–277. 12, 13, 56, 69

[25] C. Zach, T. Pock, H. Bischof, A duality based approach for realtime tv- l 1 optical flow, in: DAGM07, 2007, pp. 214–223. 13

[26] T. Huang, J. Burnett, A. Deczky, The importance of phase in image processing filters, Acoustics, Speech and Signal Processing, IEEE Transactions on 23 (6) (1975) 529 – 542. 13

[27] A. Oppenheim, J. Lim, The importance of phase in signals, Proceedings of the IEEE 69 (5) (1981) 529 – 541. 13

[28] B. J., P. M., Z. Y.Y., The importance of localized phase in vision and image representation, in: SPIE, Visual Communications and Image Processing, Vol. 1001, 1988, pp. 61–68. 13

[29] D. J. Fleet, A. D. Jepson, Stability of phase information, IEEE Trans. Pattern Anal. Mach. Intell. 15 (12) (1993) 1253–1268. `doi:http://dx.doi.org/10.1109/34.250844`. 13, 14, 16, 56, 68

[30] A. Cozzi, B. Crespi, F. Valentinotti, F. Wörgötter, Performance of phase-based algorithms for disparity estimation, Mach. Vision Appl. 9 (5-6) (1997) 334–340. `doi:http://dx.doi.org/10.1007/s001380050052`. 13, 16, 30

[31] P. Kovesi, Phase congruency: A low-level image invariant (2000). doi:10.1007/s004260000024. 13

[32] G. Carneiro, A. D. Jepson, Multi-scale phase-based local features, Computer Vision and Pattern Recognition, IEEE Computer Society Conference on 1 (2003) 736. doi:http://doi.ieeecomputersociety.org/10.1109/CVPR.2003.1211426. 13

[33] T. Gautama, M. Van Hulle, A phase-based approach to the estimation of the optical flow field using spatial filtering, IEEE Transactions on Neural Networks 13 (5) (2002) 1127–1136. 13, 16

[34] S. P. Sabatini, F. Solari, P. Cavalleri, G. M. Bisio, Phase-based binocular perception of motion in depth: cortical-like operators and analog vlsi architectures, EURASIP J. Appl. Signal Process. 2003 (2003) 690–702. doi:http://dx.doi.org/10.1155/S1110865703302033. 13, 56, 103

[35] A. Darabiha, J. MacLean, J. Rose, Reconfigurable hardware implementation of a phase-correlation stereoalgorithm, Mach. Vision Appl. 17 (2) (2006) 116–132. 13, 51, 52

[36] B. K. P. Horn, B. G. Schunck, "determining optical flow": A retrospective, Artif. Intell. 59 (1-2) (1993) 81–87. 14

[37] B. D. Lucas, T. Kanade, An iterative image registration technique with an application to stereo vision, in: IJCAI, 1981, pp. 674–679. 14, 55

[38] B. K. P. Horn, B. G. Schunck, Determining optical flow, Artif. Intell. 17 (1-3) (1981) 185–203. 14, 55

[39] T. Brox, A. Bruhn, N. Papenberg, J. Weickert, High accuracy optical flow estimation based on a theory for warping, in: ECCV04, 2004, pp. 25–36. 14

[40] S. Sabatini, G. Gastaldi, F. Solari, J. Diaz, E. Ros, K. Pauwels, M. Van Hulle, N. Pugeault, N. Krüger, Compact and accurate early vision processing in the harmonic space, in: International Conference on Computer Vision Theory and Applications, Barcelona, 2007, pp. 213–220. 16, 18, 19, 21, 22, 56, 76, 77, 86

[41] K. Pauwels, M. M. V. Hulle, Realtime phase-based optical flow on the gpu, Computer Vision and Pattern Recognition Workshop 0 (2008) 1–8. 16, 18, 38, 56, 68

[42] W. T. Freeman, E. H. Adelson, The design and use of steerable filters, PAMI 13 (1991) 891–906. 16, 20, 23, 60, 77

[43] F. Solari, S. Sabatini, G. Bisio, Fast technique for phase-based disparity estimation with no explicit calculation of phase, Electronics Letters 37 (23) (2001) 1382–1383. 18, 82

[44] S. Kalkan, F. Wörgötter, Y. Shi, V. Krüger, N. Krüger, A signal-symbol loop mechanism for enhanced edge extraction, in: VISAPP (2), 2008, pp. 214–221. 19

[45] L. Haglund, Adaptive multidimensional filtering, Ph.D. thesis, Linköping University, Sweden (October 1992). 19, 20, 29, 47

[46] J. Diaz, E. Ros, F. Pelayo, E. Ortigosa, S. Mota, FPGA-based real-time optical-flow system, Circuits and Systems for Video Technology, IEEE Transactions on 16 (2) (2006) 274–279. 19, 56

[47] O. Nestares, R. Navarro, J. Portilla, A. Tabernero, Efficient spatial-domain implementation of a multiscale image representation based on Gabor functions, Journal of Electronic Imaging 7 (1) (1998) 166–173. 23, 77

[48] K. Pauwels, Filter evaluation: optic flow and disparity, Tech. rep., K.U. Leuven (2006). 23

[49] D. J. Fleet, A. D. Jepson, M. R. M. Jenkin, Phase-based disparity measurement, CVGIP-Image. Understand. 53 (2) (1991) 198–210. 30

[50] P. Burt, E. Adelson, The laplacian pyramid as a compact image code, Communications, IEEE Transactions on 31 (4) (1983) 532 – 540. 31

[51] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, J. M. Ogden, Pyramid methods in image processing, RCA Engineer 29 (6) (1984) 33–41. 31

[52] J. R. Bergen, P. Anandan, K. J. Hanna, R. Hingorani, Hierarchical model-based motion estimation, in: ECCV '92: Proc. of the 2nd European Conf. on Computer Vision, Springer-Verlag, 1992, pp. 237–252. 31

[53] K. Pauwels, M. M. V. Hulle, Optic flow from unstable sequences containing unconstrained scenes through local velocity constancy maximization, in: In Proc. British Machine Vision Conference, 2006, pp. 4–7. 31, 69

[54] S. P. Sabatini, G. Gastaldi, F. Solari, K. Pauwels, M. M. V. Hulle, J. Diaz, E. Ros, N. Pugeault, N. Krger, A compact harmonic code for early vision based on anisotropic frequency channels, Computer Vision and Image Understanding In Press, Corrected Proof (2010) –. doi:DOI:10.1016/j.cviu.2010.03.008. URL http://www.sciencedirect.com/science/article/B6WCX-4YR33J9-2/2/4fc06b98fa2221dd7e8908617f7fb72e 35, 47, 95, 97

[55] F. Valentinotti, S. Taraglio, Phase difference stereo disparity computation on a simd parallel machine, in: HPCN Europe '97: Proc. of the Inter. Conf. and Exhibition on High-Performance Computing and Networking, Springer-Verlag, London, UK, 1997, pp. 127–136. 38

[56] D. K. Masrani, W. J. MacLean, A real-time large disparity range stereo-system using FPGAs, in: ICVS '06: Proc. of the IV IEEE Inter. Conf. on Computer Vision Systems, 2006, p. 13. 38, 51, 52

[57] S. Park, H. Jeong, Real-time stereo vision FPGA chip with low error rate, in: MUE '07: Proc. of the 2007 Inter. Conf. on Multimedia and Ubiquitous Engineering, 2007, pp. 751–756. 38

[58] E. Ortigosa, A. Caas, E. Ros, P. Ortigosa, S. Mota, J. Díaz, Hardware description of multi-layer perceptrons with different abstraction levels, Microprocessors and Microsystems 30 (7) (2006) 435 – 444. 39, 40, 57

[59] M. Vanegas, M. Tomasi, J. Díaz, E. Ros, Multiport abstraction layer for FPGA intensive memory exploitationapplications, accepted to Journal of System Architecture (2010). 40, 41, 43, 57, 59, 84, 115

[60] J. Valls, M. Kuhlmann, K. K. Parhi, Evaluation of cordic algorithms for FPGA design, J. VLSI Signal Process. Syst. 32 (3) (2002) 207–222. doi:http://dx.doi.org/10.1023/A:1020205217934. 42, 60, 87

[61] Seven solutions, Website, http://www.sevensols.com (2010). 48, 49, 90, 113

[62] L. Wang, M. Liao, M. Gong, R. Yang, D. Nister, High-quality real-time stereo using adaptive cost aggregation and dynamic programming, in: Proc. of the 3rd Inter. Symposium on 3D Data Processing, Visualization, and Transmission (3DPVT'06), 2006, pp. 798–805. 51, 52

[63] M. Gong, Y.-H. Yang, Real-time stereo matching using orthogonal reliability-based dynamic programming, IEEE Trans. on Image Processing 16 (3) (2007) 879–884. 51, 52

[64] J. Gibson, O. Marques, Stereo depth with a unified architecture gpu, Computer Vision and Pattern Recognition Workshop 0 (2008) 1–6. 52

[65] S. Li, Binocular spherical stereo, IEEE Trans. on Intelligent Transportation Systems 9 (4) (2008) 589–600. 52

[66] C. Murphy, D. Lindquist, A. M. Rynning, T. Cecil, S. Leavitt, M. L. Chang, Low-cost stereo vision on an FPGA, in: FCCM '07: Proc. of the 15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2007, pp. 333–334. 52

[67] J. Woetzel, R. Koch, Real-time multi-stereo depth estimation on GPU with approximative discontinuity handling, in: 1st Conf. on Visual Media Production, 2004, pp. 245–254. 52

[68] Y. Miyajima, T. Maruyama, A real-time stereo vision system with FPGA, in: FPL, 2003, pp. 448–457. 52

[69] A. Verri, T. Poggio, Motion field and optical flow: qualitative properties, Pattern Analysis and Machine Intelligence, IEEE Transactions on 11 (5) (1989) 490 –498. doi:10.1109/34.24781. 55

[70] B. Ni, A. Kassim, S. Winkler, A hybrid framework for 3-d human motion tracking, Circuits and Systems for Video Technology, IEEE Transactions on 18 (8) (2008) 1075 –1084. doi:10.1109/TCSVT.2008.927108. 56

[71] C. McCarthy, N. Barnes, Performance of optical flow techniques for indoor navigation with a mobile robot, in: Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on, Vol. 5, 2004, pp. 5093 – 5098 Vol.5. doi:10.1109/ROBOT.2004.1302525. 56

[72] J. Díaz Alonso, E. Ros Vidal, A. Rotter, M. Muhlenberg, Lane-change decision aid system based on motion-driven vehicle tracking, Vehicular Technology, IEEE Transactions on 57 (5) (2008) 2736 –2746. doi:10.1109/TVT.2008.917220. 56, 87

[73] J. Han, D. Farin, P. de With, Broadcast court-net sports video analysis using fast 3-d camera modeling, Circuits and Systems for Video Technology, IEEE Transactions on 18 (11) (2008) 1628 –1638. doi:10.1109/TCSVT.2008.2005611. 56

[74] G. Papadopoulos, A. Briassouli, V. Mezaris, I. Kompatsiaris, M. Strintzis, Statistical motion information extraction and representation for semantic video analysis, Circuits and Systems for Video Technology, IEEE Transactions on 19 (10) (2009) 1513 –1528. doi:10.1109/TCSVT.2009.2026932. 56

[75] K. Pauwels, M. Van Hulle, Segmenting independently moving objects from egomotion flow fields, early Cognitive Vision Workshop, Isle of Skye. (June 2004). 56

[76] J. Diaz, E. Ros, A. Prieto, F. J. Pelayo, Fine grain pipeline systems for real-time motion and stereo-vision computation. 1 (1) (2007) 60–68. 57

[77] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. Black, R. Szeliski, A database and evaluation methodology for optical flow, in: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on, 2007, pp. 1 –8. doi:10.1109/ICCV.2007.4408903. 67

[78] Z. Wei, D.-J. Lee, B. E. Nelson, J. K. Archibald, B. B. Edwards, FPGA-based embedded motion estimation sensor, International Journal of Reconfigurable Computing. 67, 68, 69

[79] G. Botella, A. Garcia, M. Rodriguez-Alvarez, E. Ros, U. Meyer-Baese, M. C. Molina, Robust bioinspired architecture for optical-flow computation, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on PP (99) (2009) 1 –1. doi:10.1109/TVLSI.2009.2013957. 69

[80] Y. Murachi, Y. Fukuyama, R. Yamamoto, J. Miyakoshi, H. Kawaguchi, H. Ishihara, M. Miyama, Y. Matsuda, M. Yoshimoto, A vga 30-fps realtime optical-flow processor core for moving picture recognition, IEICE Transactions 91-C (4) (2008) 457–464. 69

[81] J. C. Sosa, R. Gomez-Fabela, J. A. Boluda, F. Pardo, Change-driven image architecture on FPGA with adaptive threshold for optical-flow computation, 2006, pp. 1–8. doi:10.1109/RECONF.2006.307775. 69

[82] J. L. Martín, A. Zuloaga, C. Cuadrado, J. Lázaro, U. Bidarte, Hardware implementation of optical flow constraint equation using FPGAs, Computer Vision and Image Understanding 98 (3) (2005) 462 – 490. doi:DOI:10.1016/j.cviu.2004.10.002.
URL http://www.sciencedirect.com/science/article/B6WCX-4F0117W-2/2/f00d6929782b99235f5989d325911079 69

[83] H. Niitsuma, T. Maruyama, High speed computation of the optical flow, in: ICIAP, 2005, pp. 287–295. 69

[84] M. Correia, A. Campilho, Real-time implementation of an optical flow algorithm, Vol. 4, 2002, pp. 247 – 250. doi:10.1109/ICPR.2002.1047443. 69

[85] J. Vallino, Datacube mv200 and imageflow user"s guide, Tech. rep. (1995). 69

[86] J.-W. Gao, K.-B. Jia, Embedded video surveillance system based on h.264, Multimedia Information Networking and Security, International Conference on 1 (2009) 282–286. doi:http://doi.ieeecomputersociety.org/10.1109/MINES.2009.115. 76

[87] H. Bessalah, F. Alim-Ferhat, H. Salhi, S. Seddiki, M. Issad, O. Kerdjidj, On line wavelets transform on a xilinx FPGA circuit to medical images compression, Digital Image Processing, International Conference on 0 (2009) 8–12. doi:http://doi.ieeecomputersociety.org/10.1109/ICDIP.2009.89. 76

[88] Y. E. C. C., C. H. Llanos, W. de Britto Vidal Filho, L. dos Santos Coelho, Fuzzy control for cyclist robot stability using FPGAs, Reconfigurable Computing and FPGAs, International Conference on 0 (2009) 410–415. doi:http://doi.ieeecomputersociety.org/10.1109/ReConFig.2009.53. 76

[89] Y. Liu, K. Benkrid, A. Benkrid, S. Kasap, An FPGA-based web server for high performance biological sequence alignment, Adaptive Hardware and Systems, NASA/ESA Conference on 0 (2009) 361–368. doi:http://doi.ieeecomputersociety.org/10.1109/AHS.2009.59. 76

[90] E. Ros, E. M. Ortigosa, R. Agís, R. Carrillo, M. Arnold, Realtime computing platform for spiking neurons (rt-spike), IEEE Transactions on Neural Networks 17 (2006) 1050–1063. 76

[91] R. Agís, E. Ros, J. Díaz, R. Carrillo, E. M. Ortigosa, Hardware event-driven simulation engine for spiking neural networks, International Journal of Electronics 94 (5) (2007) 469–480. 76

[92] T. Q. Vinh, J. H. Park, Y.-C. Kim, S. H. Hong, FPGA implementation of real-time edge-preserving filter for video noise reduction, Computer and Electrical Engineering, International Conference on 0 (2008) 611–614. doi:http://doi.ieeecomputersociety.org/10.1109/ICCEE.2008.61. 76

[93] M. E. Angelopoulou, C.-S. Bouganis, P. Y. K. Cheung, G. A. Constantinides, Robust real-time super-resolution on FPGA and an application to video enhancement, ACM Trans. Reconfigurable Technol. Syst. 2 (4) (2009) 1–29. doi:http://doi.acm.org/10.1145/1575779.1575782. 76

[94] R. Bannister, D. Gregg, S. Wilson, A. Nisbet, FPGA implementation of an image segmentation algorithm using logarithmic arithmetic, Vol. 1, 2005, pp. 810–813. doi:10.1109/MWSCAS.2005.1594224. 76

[95] J. Diaz, E. Ros, S. Mota, R. Carrillo, Local image phase, energy and orientation extraction using FPGAs, International Journal of Electronics 95 (July 2008) 743–760(18). doi:doi:10.1080/00207210801941200. 77

[96] M. Tomasi, F. Barranco, M. Vanegas, J. Diaz, E. Ros, High performance optical flow architecture based on a multiscale and multi-orientation phase-based model, accepted to IEEE Trans. on Circuits and Systems for Video Technology (May 2010). 86, 88

[97] M. Tomasi, M. Vanegas, F. Barranco, J. Diaz, E. Ros, Real-time architecture for a robust multiscale stereo engine, submitted to IEEE Trans. on Image Processing (submitted in 2009). 88

[98] F. Barranco, J. Díaz, E. Ros, B. n. Del Pino, Visual system based on artificial retina for motion detection, Trans. Sys. Man Cyber. Part B 39 (3) (2009) 752–762. doi:http://dx.doi.org/10.1109/TSMCB.2008.2009067. 90

[99] S. Granados, S. Mota, J. Díaz, E. Ros, Condensación de primitivas visuales de bajo nivel para aplicaciones de procesamiento en tiempo real, in: VIII JCRA. Madrid (Spain), 2008, p. XX, vIII JCRA. Madrid (Spain), Pp.XX. 91

[100] M. Bhatnagar, Tms320dm6446/3 power consumption summary, Tech. rep., Texas Instruments, application report SPRAAD6A (February 2008). 92

[101] J. B., Estimating power for adsp-bf561 blackfin processors, Engineer-to-engineer note ee-293, Analog Devices (June 2007). 92

[102] B. Cope, P. Y. Cheung, W. Luk, L. Howes, Performance comparison of graphics processors to reconfigurable logic: A case study, IEEE Transactions on Computers 59 (2010) 433–448. doi:http://doi.ieeecomputersociety.org/10.1109/TC.2009.179. 94

[103] PtGrey products: http://www.ptgrey.com/products/bumblebee2/index.asp. 94

[104] NI Smart Camera: http://sine.ni.com/nips/cds/view/p/lang/en/nid/205959. 94

[105] ISRA vision webpage: http://www.isravision.com/. 94

[106] OMRON products webpage: http://industrial.omron.eu/en/products/. 94

[107] MVTec webpage: http://www.mvtec.com/halcon/technical-data/. 94

[108] CVB products: http://en.commonvisionblox.de/en/pages/cvb/main.php. 94

[109] W. Zuo, Q. Chen, Fast and informative flow simulations in a building by using fast fluid dynamics model on graphics processing unit, Building and Environment 45 (3) (2010) 747 – 757. 95

[110] K. Wong, R. Kelso, S. Worthley, P. Sanders, J. Mazumdar, D. Abbott, Theory and validation of magnetic resonance fluid motion estimation using intensity flow data, online, pLoS One (March 2009). 95

[111] M. Tomasi, F. Barranco, M. Vanegas, J. Diaz, E. Ros, Fine grain pipeline architecture for high performance phase-based optical flow computation, submitted to Journal of System Architecture (Submitted in 2009, reviewed with minor revision).

[112] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, Arquitectura multiescala de cálculo de flujo óptico basado en la fase, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009, 2009, pp. 295–304, IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009.

[113] M. Tomasi, J. Díaz, E. Ros, Real time architectures for moving-objects tracking, in: (ARC2007), Lecture Notes in Computer Science, Vol. XXX, 2007, pp. 365–372.

[114] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, A novel architecture for a massively parallel low level vision processing engine on chip, ISIE2010. Accepted to IEEE Internetional Symposium on Industrial Electronics, Bari (Italy) (July 2010).

[115] F. Barranco, M. Tomasi, M. Vanegas, S. Granados, J. Diaz, Entorno software para visualización y configuración de procesamiento de imágenes en tiempo real

con plataformas reconfigurables, in: IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009., 2009, pp. 327–336, IX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009.

[116] P. Perona, J. Malik, Scale-space and edge detection using anisotropic diffusion, Pattern Analysis and Machine Intelligence, IEEE Transactions on 12 (7) (1990) 629 –639. `doi:10.1109/34.56205`. 103

[117] Enpeda project webpage, Website, `http://www.mi.auckland.ac.nz/` (2010). 104

[118] G. Arce, R. Stevenson, On the synthesis of median filter systems, Circuits and Systems, IEEE Transactions on 34 (4) (1987) 420 – 429. 105

[119] R. Maheshwari, S. Rao, P. Poonacha, FPGA implementation of median filter, in: VLSI Design, 1997. Proceedings., Tenth International Conference on, 1997, pp. 523 –524. `doi:10.1109/ICVD.1997.568194`. 106

# Arquitectura piramidal de Visión estéreo y estimación de movimiento en tiempo real basada en FPGAs

Matteo Tomasi

Departamento de Arquitectura y Tecnología de Computadores

Universidad de Granada

Tesis para optar al grado de

*Doctor en filosofía (PhD)*

Mayo 2010

1. Revisor:

2. Revisor:

3. Revisor:

4. Revisor:

5. Revisor:

Día de la defensa:

Firma del presidente del tribunal:

# Resumen

En este trabajo de tesis presentamos un amplio estudio sobre motores para procesamiento masivo de algoritmos de visión. Nuestra hipótesis de trabajo se centra en las características y las ventajas de los dispositivos basados en FPGAs (bajos consumos de potencia, altas prestaciones) ademas de los grandes avances en las herramientas de síntesis y los lenguajes HDL. Gracias a estos valiosos medios presentamos y afrontamos la novedosa idea de un sistema de visión de bajo nivel. El estudio que presentamos busca demonstrar las múltiples posibilidades de integración de algoritmos complejos gracias a una atenta adaptación y a nuevas técnicas de diseño. En particular enfocamos nuestra arquitectura a un diseño con cauces de grano fino contrariamente a la implementación de los actuales multi-core muy utilizados hoy en día. El sistema final representa un motor de procesamiento de vanguardia con un consumo de potencia y un tamaño útiles para aplicaciones industriales, robóticas y para investigación. Por primera vez se diseña una arquitectura multi-orientación y multi-escala en dispositivos FPGAs. El carácter iterativo del algoritmo degrada la velocidad de procesamiento pero al mismo tiempo permite una gran mejoría en precisión. Los resultados finales para secuencias sintéticas y reales demuestran el alto nivel de competitividad de nuestro innovador sistema.

A mi familia

# Índice general

# Índice de figuras

# 1

# Introducción

*Historia magistra vitae est.* Si buscamos en las teorías de la antigüedad es posible encontrar rudimentos de filosofía sobre la interpretación de la visión humana ya en la época de la antigua Grecia. Aristóteles en su libro "De los sentidos y de lo sentido"(Parva Naturalia) afirma que *"De las dos últimas, la visión, como exigencia primaria para la vida y sus efectos directos es el sentido superior; ... La facultad de la visión, gracias a que todos los cuerpos tienen color, nos lleva a una gran variedad de diferentes cualidades de todo tipo; por cual es a través de este sentido que somos capaces de percibir las cosas comunes como figuras, tamaños, movimiento, cantidad ..."*. Después de estas primeras teorías, una gran cantidad de filósofos y científicos han contribuido a la fundación de múltiples postulados sobre el sistema de visión humano. Esta notable y larga historia demuestra la importancia que en nuestra vida tiene el comprender como funciona la visión. En el día a día nos encontramos con desafíos que tienen que ver con la visión, por ejemplo en el ámbito deportivo, los entrenadores de un equipo coinciden todos en que un buen jugador tiene que ser rápido en la visión del juego y en la interpretación de la jugada antes que su adversario. Si nos movemos a un entorno más crítico como puede ser la conducción de vehículos en carretera, podemos afirmar que un conductor necesita una reacción muy rápida frente a la presencia súbita de un obstáculo en su camino; para ello el conductor necesita estar entrenado de forma tal que su sistema de procesamiento requiera menos tiempo para detectar y evitar el peligro. En ambos casos tenemos una fuerte dependencia del sistema de visión. Con estos simples ejemplos queremos resaltar la importancia de la visión. En este trabajo se quiere diseñar y construir un sistema de visión que reúna algunas de las características del sistema de visión humano en un sistema de visión artificial. Para realizar esta difícil tarea nos hemos beneficiado de los grandes avances tecnológicos y en la masiva investigación realizada en estos últimos años; para ello hemos hecho un atento y preciso estudio de las soluciones existentes del problema de visión artificial. No obstante la evolución y el incremento de prestaciones en procesadores de propósito general, nosotros proponemos la utilización de pequeños sistemas empotrados que puedan ser rápidos y eficientes en el procesado de vídeo. Cada vez estamos más rodeados de pequeños dispositivos electrónicos equipados con potentes y valiosos chips. En el trabajo propuesto intentamos aportar una nueva contribución al campo de los dispositivos basados en FPGA (Field Programmable Gate Array) e introducir nuevas arquitecturas de procesamiento de vídeo.

## 1.1. Sistema de visión

Un sistema de visión de los más complejos y perfecto es la vista humana, como lo afirma Kandel en su libro [1] y cito textualmente: *Estudios de inteligencia artificial y de reconocimiento de patrones por computador han demostrado que el cerebro reconoce formas, movimiento, profundidad y color usando estrategias que ninguna clase de computador puede alcanzar. Simplemente mirar el mundo y reconocer una cara o disfrutar de un paisaje requiere un enorme esfuerzo computacional mucho más difícil del que se necesita para resolver problemas lógicos o jugar al ajedrez.* Desde los primeros estudios de biología y neurología, los científicos han estado fascinados por ese maravilloso sistema que es la "máquina"humana. Observando y entendiendo los mecanismos naturales, los investigadores intentan reproducir su funcionamiento. El sistema de visión humano en este sentido no representa una excepción. Gracias a esta forma de actuar han nacido la fotografía y las cámaras de vídeo, incluso muchos algoritmos de computo se inspiran en la naturaleza. Al día de hoy el sistema de visión humana queda todavía sin ser entendido por completo. Ya sabemos que el complejo proceso de visión y de interpretación de las escenas se divide en diversos niveles de procesamiento. Como se muestra en Figura 1.1 y basándose en la neurociencia, un sistema de visión se puede dividir en bajo, medio y alto nivel [2]. Comparado con otras partes del cerebro, sabemos que la corteza visual ocupa un gran porcentaje de la totalidad de funciones. Por ejemplo en los monos el 50 % se dedica al procesamiento de la información visual mientras que el 11 % se dedica al tacto y solo el 3 % al procesamiento auditivo. De igual forma sabemos que entre las diferentes partes de la corteza visual las más grandes son las areas V1 y V2 (visión de bajo nivel) que ocupan unos 1100 mm$^2$, mientras que una de la más pequeñas es la MT (Middle Temporal) que ocupa solo 55 mm$^2$ según se puede leer en [1]. De forma similar en vision por computador un sistema de vision artificial necesita un mayor esfuerzo computacional para la parte de bajo nivel comparado con la de medio o alto nivel. Aunque se tiene un buen conocimiento del bajo y medio nivel de visión del sistema visual humano, aún se sabe muy poco de la capa alta de visión. La forma en que se relacionan los objetos que vemos y la información dada por la experiencia pasada (binding problem) aún es poco clara y da lugar a complicados debates entre estudios neurofisiológicos y psicológicos del proceso de percepción humana. En visión por computador los investigadores discrepan con respecto a una exacta catalogación de roles entre los distintos niveles. Weems [2] explica que si una representación esta basada en vectores de datos numéricos que corresponden directamente a cada dato de la imagen (operaciones pixel a pixel) podemos hablar de visión de bajo nivel. Una representación basada en descripciones simbolicas de eventos extraidos de la imagen o en la instancia de modelos conocidos y almacenados en memoria representa el nivel de visión intermedio. Por último las representaciones independientes de la escena o de la visual pertenecen al alto nivel. Por lo tanto el alto nivel se caracteriza por modelos y conocimientos generales, como por ejemplo el conocimiento del entorno actual de una escena.

Actualmente los sistemas de visión adquieren gran importancia en muchos campos de aplicación y nuestro mundo se rodea de cámaras de vídeo y pantallas. Muchos procesos industriales utilizan plataformas robóticas dotadas de visión artificial para controlar sus cadenas de producción. Todos los edificios públicos están equipados con sistemas de visión con función de vídeo vigilancia para la seguridad. La industria del automóvil estudia la posibilidad de un nuevo concepto de sensor basado en cámaras de vídeo inteligentes que alerten al conductor sobre la presencia de peligros [3]. Muchas otras aplicaciones, en el campo militar [4] o el médico, hacen uso de la visión por computador [5].

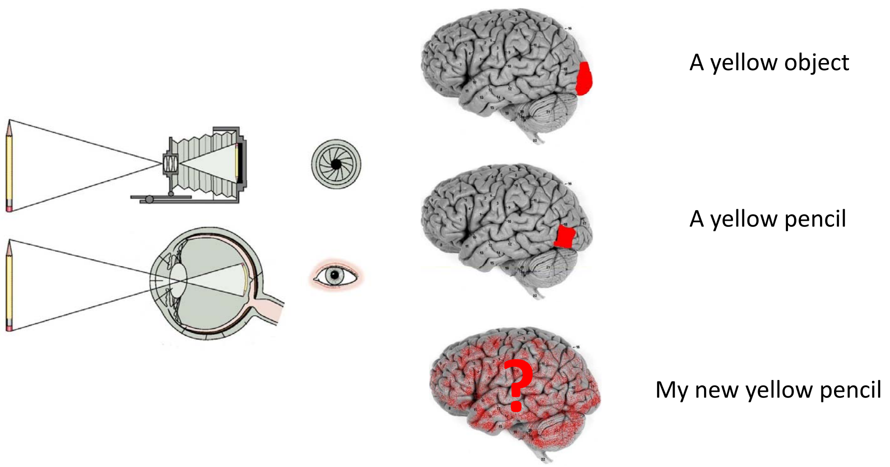


**Figura 1.1: Diferentes niveles de visión** - A bajo nivel (regiones V1 y V2 del cerebro humano) procesamos la imagen entera extrayendo las características: mayor esfuerzo computacional. A nivel medio (V4, corteza temporal inferior) asociamos un determinado objeto. Finalmente en el nivel alto (binding problem) reconocemos un objeto específico accediendo a informaciones que no dependen de la escena. Esta figura es una adaptación de [1].

## 1.2. Plataformas de procesamiento para visión: estado del arte

La tecnología actual ofrece una gran variedad de soluciones para la implementación de sistemas de visión; en el campo de la investigación y el desarrollo de algoritmos, la forma de validación de métodos más práctica y rápida es la programación de alto nivel en procesadores de propósito general, el poder computacional de estas máquinas ha crecido según la ley de Moore y la presencia de diversos lenguajes de programación ofrece al investigador un amplio abanico de posibilidades para una rápida implementación. Por ejemplo una herramienta muy utilizada para validar modelos en ingeniería es la herramienta de software Matlab. Esta herramienta incluye varias utilidades para el procesamiento de imágenes y tiene una gran eficiencia en operaciones matriciales que se traduce en tiempos de validación cortos. Debido a esta rapidez y a la facilidad de implementación de modelos, se puede considerar el uso de Matlab como la forma más fácil de implementar cualquier modelo.

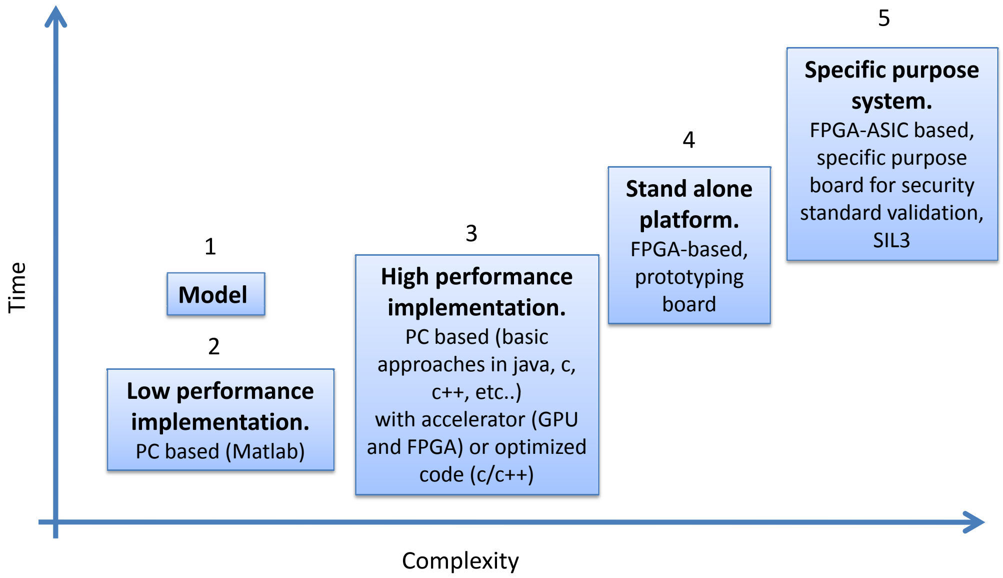

Time
Complexity
1
Model
2
Low performance implementation.
PC based (Matlab)
3
High performance implementation.
PC based (basic approaches in java, c, c++, etc..) with accelerator (GPU and FPGA) or optimized code (c/c++)
4
Stand alone platform.
FPGA-based, prototyping board
5
Specific purpose system.
FPGA-ASIC based, specific purpose board for security standard validation, SIL3

**Figura 1.2: Pasos para el desarrollo de un sistema de visión -** El proceso empieza con un simple lenguaje de programación de alto nivel para llegar hasta una síntesis en productos industriales (VLSI). Los pasos se sitúan en una gráfica tiempo/complejidad.

Para aumentar la rapidez de la ejecución del código y llegar hasta un producto industrial se pueden seguir diferentes pasos (Figure 1.2). Dependiendo del tipo de aplicación es posible parar el proceso en cada uno de estos pasos. Si no tenemos ninguna restricción de tamaño o de consumo de potencia, podemos quedarnos en un código software para procesadores de propósito general e incrementar las prestaciones con técnicas de optimización de códigos a bajo nivel como se explica en [6] o incluso utilizar aceleradores gráficos (GPU) como alternativa. En el primer caso la optimización adoptada se pueden seguir distintas estrategias: una simple optimización del compilador o una implementación con código de bajo nivel como el ensamblador. La segunda solución es una implementación con lenguajes similares al C en entornos específicos de programación que cambian según la plataforma que se utilice: para las tarjetas Nvidia por ejemplo se utiliza el CUDA [7]. Hoy en día gracias a la importancia del mercado de los juegos, la tecnología de las GPUs ha mejorado enormemente con un alto nivel de paralelismo y rápidos accesos a memoria. Si tenemos restricciones de tamaño o de consumo de potencia la solución para nuestra implementación se mueve hacia el campo de las plataformas de procesamiento como por ejemplo la que se basan en FPGAs. Este último método se beneficia de un alto nivel de paralelismo en las operaciones con una frecuencia de reloj muy reducida: unos dos ordenes de magnitud menos que un procesador normal o una GPU. Esto va a beneficiar el consumo de potencia y su uso en aplicaciones empotradas. El último paso y el más complejo es el diseño en sistemas de propósito específico. A partir de una plataforma de procesamiento y su previa validación es posible producir un chip VLSI para aplicaciones industriales que, debido a su gran escala de integración, puede disminuir los costes de producción a través de la gran cantidad de unidades que pueden ser fabricadas. Según la aplicación final es posible pararse en una determinada etapa de diseño o seguir hasta el producto más óptimo. Por ejemplo si estamos interesados en plataformas robóticas para aplicaciones de investigación, una plataforma con hardware específico equipada con DSPs o FPGAs es la solución más adecuada; no será necesario pasar por una optimización del código o una implementación en GPU que serían inútiles para el resultado final, de hecho es suficiente una implementación básica con simples lenguajes de programación como Matlab, Octave o equivalentes para definir si el modelo es válido. Finalmente y sólo para fines comerciales se necesita de un ulterior esfuerzo en términos de tiempo de diseño y validación que incluyen la certificación de seguridad con el respectivo nivel SIL (Security Integration Level). Estas certificaciones son indispensables para el mundo industrial y habitualmente no se otorgan en el campo de la investigación. Podemos encontrar en la literatura diferentes soluciones para cada uno de los pasos presentados, pero en general los más comunes son el primero y el segundo: modelos software y optimización de códigos. Ya hemos alcanzado un alto rendimiento en la precisión para muchas de las aplicaciones de bajo nivel. Algunas contribuciones como [8] y [9] exponen en detalle diferentes algoritmos para el cálculo de visión estéreo y movimiento. Los investigadores contribuyen a la literatura con la generación de diversos conjuntos de imágenes de referencia que se utilizan como bancos de prueba común [10]. Este trabajo valioso permite una importante validación de algoritmos, pero aunque hemos llegado a un alto nivel de precisión para el procesamiento de estas secuencias sintéticas, las nuevas implementaciones intentan obtener un nivel muy elevado de precisión sobre las secuencias sintéticas y no abordan muchos de los aspectos prácticos del mundo real (el tiempo real, los cambios de iluminación, ambientes no controlados, etc.) Este problema se debe a la dificultad evidente de validación cuantitativa de las secuencias reales. Las actuales técnicas de correspondencia entre imágenes se dividen principalmente en dos categorías: enfoques locales y los enfoques globales. Los algoritmos locales (window-based), donde el cálculo en un punto dado sólo depende de los valores de píxeles en una ventana espacial local, por lo general operan suavizaciones dentro de las regiones para un mejor procesamiento. Con el fin de aumentar la precisión de las estimaciones, en particular a lo largo de las fronteras, los algoritmos de última generación utilizan tamaños de ventanas variables para calcular las correspondencias en lugar de utilizar, como en los enfoques tradicionales, una ventana cuadrada fija. Por el contrario, la mayoría de los métodos globales intentan minimizar una función de energía calculada en el área de toda la imagen mediante el empleo de estrategias de minimización, como las técnicas variacionales, el modelo de Markov (Random Field), métodos Graph Cuts (GC), Belief Propagation (BP), etc. [11, 12**?** ]. Ya que esta tarea resulta ser un problema NP-hard, se aproxima una estimación por medio de estrategias eficaces [8, 11, 12]. Si nos movemos hacia una implementación optimizada de una arquitectura con hardware específico (parte derecha del gráfico de la figura 1.2) encontramos menos contribuciones en la literatura. Esto significa que la complejidad de la aplicación crece significativamente. Podemos encontrar diversas aplicaciones en GPU de los enfoques vistos anteriormente, pero sólo algunos algoritmos sencillos se han implementado en FPGA o en procesadores DSP. La razón de esta tendencia actual es el entorno de programación más fácil y el "time to market"más corto para las aplicaciones en procesadores de propósito general en comparación con soluciones basadas en DSP o FPGA. El problema evidente de la utilización de soluciones basadas en PC estándar es el consumo de energía y el tamaño considerable del producto final que es inviable para aplicaciones portátiles. Por otra parte una implementación en FPGA o DSP puede facilitar una aplicación portátil y convertirse en un producto industrial. La comparación entre DSPs y FPGAs muestra que los DSPs son más adecuados para aplicaciones de baja potencia, mientras las FPGAs constituyen una mejor opción si los requisitos de rendimiento son muy altos [13]. Para aplicaciones aún más eficientes necesitaremos el desarrollo de soluciones en ASIC. Trabajos como [14] y [15] representan una contribución muy importante para la visión en chip basados en FPGAs, pero al mismo tiempo carecen de aplicabilidad genérica. Los principales problemas para los actuales enfoques en FPGA son la velocidad de procesamiento y la adaptación a los grandes cambios de la escena: como por ejemplo los movimientos o las variaciones de la cámara. Un enfoque multi-escala puede resolver algunos de estos problemas pero presenta un alto coste computacional y no se puede considerar un método de facil implementación hardware. Por esta razón estas metodologías se adoptan pocas veces y no se encuentran en implementaciones hardware. En lo que respecta a la última etapa de la figura 1.2 es aún mas raro encontrar aplicaciones, algunos trabajos como [16] representan una contribución muy apreciable para los sistemas específicos de visión en chip.

## 1.3. Nuestra aportación

Este trabajo parte de modelos de visión existentes. En particular proponemos un estudio y una validación de algoritmos y su adaptación a plataformas hardware. Al mismo tiempo ofrecemos un análisis detallado del error debido a la utilización de una aritmética en punto fijo (errores de cuantización). Por primera vez se introducen arquitecturas de visión con una complejidad muy alta y metodologías de diseño innovadoras. El problema de precisión en amplios rangos de movimientos presentados por los enfoques existentes se resuelven con una metodología de trabajo multi-escala. Operaciones no totalmente apropiadas para sistemas hardware como la compensación iterativa de las correspondencias (warping), están incluidos en las nuevas arquitecturas que para tal propósito utilizan hasta cuatro dominios de reloj. Se introducen y optimizan varias estrategias de compartición de recursos y se estudian los compromisos entre el consumo de recursos y la precisión. La idea general es obtener una reducción de rendimiento aceptable frente a un ahorro de unidades de procesamiento paralelas como se muestra en la figura 1.3. Se han explorado diversas implementaciones en chip para la validación de la modularidad y de la escalabilidad de nuestra arquitectura.

Al mismo tiempo se estudia la adaptación del diseño al tamaño del chip y a los requisitos del sistema. Nuestro requisito mínimo es un procesamiento en tiempo real

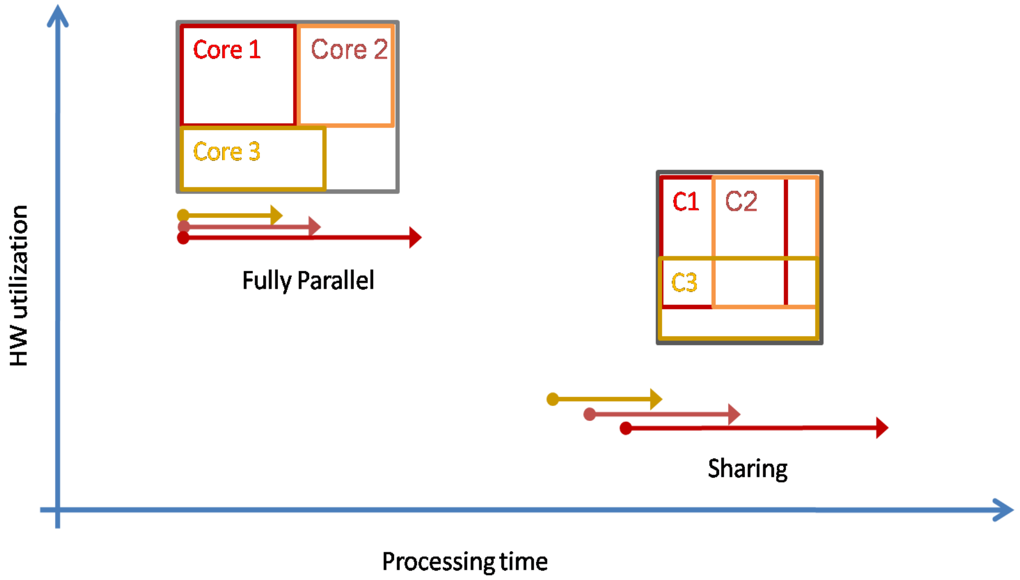


**Figura 1.3: Compartición de recursos** - Una implementación completamente paralela (izquierda) comparada con una implementación en la que se comparten recursos (derecha). Al compartir recursos se pierde en rendimiento pero se gana en consumo de recursos.

(25 cuadros por segundo) para una resolución de imagen de 512x512 píxeles. Este trabajo ofrece una buena contribución al avance de la visión por ordenador en sistemas portátiles ya que al día de hoy no existen aportaciones de tal complejidad y poder de computación. Intentamos comparar nuestra propuesta con el estado del arte en términos cuantitativos. Secuencias de referencia muy conocidas como la de [10] se utilizan como validación cuantitativa, por otro lado la estabilidad de nuestro sistema ha sido validada también en secuencias reales de forma particular en entornos de posibles aplicaciones industriales como el campo de la automoción.

## 1.4. Marco del proyecto

Nuestro sistema de visión a bajo nivel ha sido desarrollado en el marco del proyecto europeo DRIVSCO [17] (Learning to emulate Perception-Action Cycles in a driving school scenario) en colaboración con seis universidades europeas. El objetivo del proyecto es desarrollar, testar e implementar una estrategia que combine nuevos métodos de aprendizaje adaptativos con métodos de control clásico, empezando por un sistema de control con interfaz hombre-máquina y acabando por sistemas autónomos que actúen de forma independiente después de haber aprendido del comportamiento humano. La Universidad de Granada fue participe del proyecto en la parte de implementación del motor de procesamiento en chip. En particular en nuestro grupo han colaborado cinco distintas personas. El producto final incluye la realización de una placa de co-procesamiento con una plataforma basada en FPGA. As indicated in Fig. 1.4 the complete work is composed by:

- Interfaz software con el PC
- Controlador de memoria (Memory Controller Unit)
- **Cores de procesamiento**
- **Arquitectura multi-escala**
- Módulos de condensación de la información
- Retroalimentación con informaciones de alto nivel: signal to simbol loop.

El trabajo presentado en esta memoria se enfoca principalmente en los cores de procesamiento y en la arquitectura multi-escala. Obviamente el trabajo en grupo supone una colaboración cruzada entre los distintos miembros de forma especial en la parte de test y validación.

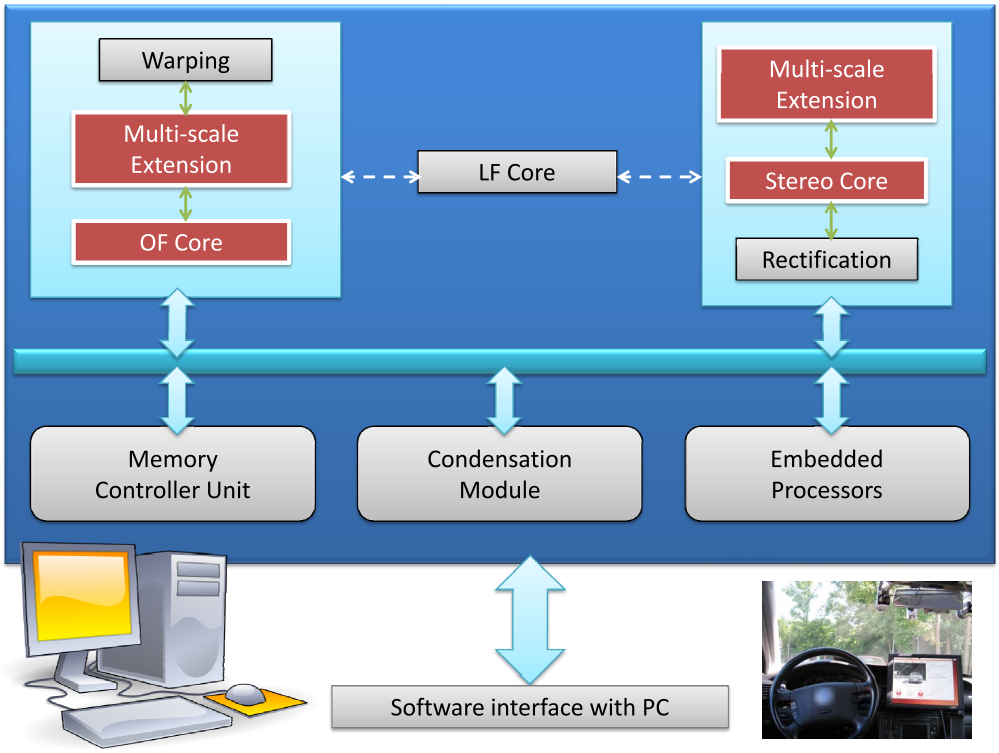


**Figura 1.4: Marco del proyecto.** - Diagrama de bloques del sistema DRIVSCO desarrollado por el grupo de la Universidad de Granada. Los bloques en rojo representan nuestra aportación.

# 2

# Discusión

## 2.1. Motivación general del trabajo

El objetivo principal de este trabajo es el estudio y la aplicación de un sistema de visión de bajo nivel en un mismo chip. Los estudios preliminares y los experimentos demuestran que un enfoque basado sobre la fase representa un buen candidato para un motor de visión multimodal complejo y preciso. A pesar de que el algoritmo no es la mejor opción en términos de precisión entre todos los trabajos presentes en la literatura, nuestra elección se justifica no sólo por la robustez de la fase en escenarios sin restricciones, sino también por la validez de la detección de la fase para la extracción de diferentes características como energía, orientación y fase locales, estéreo y computación de flujo óptico. Así, este algoritmo multi-orientación permite abordar todas estas características de visión con la posibilidad de compartir una gran cantidad de circuitos para el procesamiento de la visión temprana. Trabajos anteriores abordan este problema por separado (movimiento o estéreo) con velocidades de fotogramas muy alta o buena precisión, pero la mayoría de ellos no llegan a estudiar conjuntamente los beneficios de un cálculo que sea a la vez preciso, de alta velocidad y capaz de extraer múltiples modalidades en el mismo chip. Para ello la metodología de trabajo se basa en un cauce de grano fino y en una nueva arquitectura multi-escala. El primero es raramente utilizado en la literatura (ya que requiere un diseño muy estructurado) y suele ser sustituido por enfoques multi-núcleo, más caros en términos de consumo de energía. El segundo, por lo que sabemos, nunca ha sido implementado en dispositivos reconfigurables. Esto representa una contribución novedosa y la definición de esta arquitectura se puede reutilizar para diferentes algoritmos o modelos de procesamiento de imágenes. Después de un exhaustivo estudio y la implementación de diferentes modalidades por separado como estéreo, flujo óptico y descriptores locales, el trabajo se centra en el estudio de un motor global de visión de bajo nivel que reúne a todas las arquitecturas anteriores. Este ambicioso objetivo nunca ha sido ofrecido en la literatura y representa una aportación novedosa para la investigación y el desarrollo de arquitecturas basadas en dispositivos FPGA. El diseño del sistema incluye el estudio y la definición de muchos parámetros y opciones para la arquitectura que pueden generar un amplio conjunto de soluciones. Nosotros demostramos que una arquitectura se puede definir según los diferentes requisitos del usuario y su implementación final será muy diferente en términos de consumo de recursos y de potencia. Nuestro objetivo inicial de diseñar un motor de procesamiento para visión de bajo nivel en un sólo chip ha sido completamente resuelto y un buen compromiso entre precisión y coste viene ofrecido e implementado en una plataforma de co-procesamiento. El sistema calcula el flujo óptico y el estéreo de múltiples escalas, los descriptores de contraste local (la energía, la orientación y la fase) con una velocidad de 28,6 cuadros por segundo (con una resolución de imagen de 512x512 píxeles). La implementación final de alto rendimiento consta de 2221 unidades de procesamiento en paralelo en el mismo chip, un rendimiento de 92,3 GigaOPS (con un reloj de sistema a 42 MHz) y un alto numero de operaciones ejecutadas por vatio de potencia consumida (aproximadamente 12,9 GigaOPS / W). La solución puede ser de interés en muchos campos de aplicación como en imágenes médicas, en robótica, en la industria y la automoción.

## 2.2. Trabajo futuro

Como sabemos las etapas de visión de bajo nivel requieren la mayor parte del coste computacional de un sistema de visión. Con este procesamiento masivo llevado a cabo en un solo chip será posible la integración de diversos algoritmos de medio o de alto nivel a un costo reducido. Estas tareas pueden ser tratados con técnicas de co-diseño HW/SW gracias a la integración en FPGAs de procesadores empotrados: por ejemplo hoy en día muchas de las FPGAs incluyen procesadores como los PowerPCs dentro de sus arquitecturas. Además, es posible construir descriptores visuales más complejos basados en procesamiento de segundo orden del movimiento. Por ejemplo, la percepción del movimiento en profundidad, el calculo del movimiento propio (ego-motion) o detección de objetos con movimiento independiente (IMO) en el que son absolutamente necesarias las primitivas de primer orden calculadas por nuestro sistema de visión. Con el fin de mejorar la portabilidad del sistema es posible el desarrollo de una placa independiente a partir de la arquitectura definida. Se puede incluir una nueva interfaz con las cámaras y la salida de vídeo en la plataforma diseñada de forma que el sistema de procesamiento pueda ser utilizado en aplicaciones como la robótica.

## 2.3. Publicación de resultados

Nuestro trabajo de investigación ha sido evaluado en el marco de conferencias internacionales y revistas científicas (con factor de impacto en el JCR).

[1] M. Vanegas, M. Tomasi, J. Díaz, E. Ros, Multiport abstraction layer for FPGA intensive memory exploitation applications, submitted to Journal of System Architecture (2009)

[2] M. Tomasi, F. Barranco, M. Vanegas, J. Diaz, E. Ros, High performance optical flow architecture based on a multiscale and multi-orientation phase-based model, submitted to IEEE Trans. on CSVT

[3] M. Tomasi, M. Vanegas, F. Barranco, J. Diaz, E. Ros, Real-time architecture for a robust multiscale stereo engine, submitted to IEEE Trans. on Image Processing

[4] M. Tomasi, F. Barranco, M. Vanegas, J. Diaz, E. Ros, Fine grain pipeline architecture for high performance phase-based optical flow computation, submitted to Journal of System Architecture

[5] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, Arquitectura multiescala de cálculo de flujo óptico basado en la fase, iX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009 (Sept. 2009)

[6] M. Tomasi, J. Díaz, E. Ros, Real time architectures for moving-objects tracking, in: ARC, 2007, pp. 365–372

[7] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, A novel architecture for a massively parallel low level vision processing engine on chip, iSIE2010. Accepted for IEEE Internetional Symposium on Industrial Electronics, Bari (Italy) (July 2010)

[8] F. Barranco, M. Tomasi, M. Vanegas, S. Granados, J. Diaz, Entorno software para visualización y configuración de procesamiento de imágenes en tiempo real con plataformas reconfigurables, iX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009. (2009)

## 2.4. Marco científico general

Este trabajo científico se ha realizado y financiado por el Proyecto Europeo DRIVSCO: "Learning to emulate perception action cycles in a driving school scenario" (IST-016276-2). Esto ha representado un excelente marco de colaboración con diversos grupos de investigación de otras universidades europeas y centros de investigación. El trabajo presentado representa la mayor contribución de la Universidad de Granada en este consorcio. Por lo tanto, una alta responsabilidad en la obtención de los resultados previstos en tiempos definidos ha sido necesaria durante el proceso de investigación. Además de los informes técnicos necesarios, presentaciones para los exámenes científicos de la UE, se ha realizado una demostración final de la aplicación. El esfuerzo invertido en esta demostración es significativo, pero permite una fácil evaluación del rendimiento del sistema y también facilita la difusión de los resultados más allá de un escenario científico puro, por ejemplo hacia futuras colaboraciones industriales además del impacto en los medios de comunicación (periódicos, TV, etc).

El trabajo ha requerido una colaboración muy estrecha con diferentes investigadores en el laboratorio. Se ha requerido una gran cantidad de esfuerzos en la fabricación de los diferentes módulos y diseños para su facilidad de integración en otras arquitecturas y también para la inclusión de módulos diseñados por estos colaboradores. Esto ha supuesto una alta coordinación del trabajo en equipo.

## 2.5. Aportaciones principales

A continuación presentamos un resumen de las principales aportaciones de nuestro trabajo:

- Entre los diferentes algoritmos de visión por computador ha sido elegido un enfoque basado en la fase. Un estudio comparativo de la robustez justifica que el planteamiento es adecuado para una correcta computación de visión de bajo nivel.

- Se utilizó una modificación apropiada del algoritmo con el fin de adaptarlo a una aplicación hardware. Además se hace un estudio exhaustivo del compromiso entre precisión de los resultados y prestaciones del sistema.

- El sistema ha sido diseñado con cauces de grano fino (con varias etapas superescalares) para maximizar el paralelismo de procesamiento. Esta estrategia de diseño es muy exigente en términos de sincronización, soporte para memoria externa y dependencias de datos.

- Se presenta una nueva arquitectura multi-escala. La comparación con los enfoques mono-escala confirma que esta extensión del algoritmo es útil para mejorar la precisión y la reducción del rango de trabajo. Este enfoque no ha sido abordado antes en sistemas hardware.

- La arquitectura definida se ha aplicado a diferentes algoritmos por separado como cálculos de estéreo, de flujo óptico y descriptores locales (energía, orientación y fase). Un estudio de evaluación comparativa con el estado del arte demuestra como nuestras implementaciones son muy competitivas con respecto a las soluciones existentes.

- Ha sido realizado un sistema conjunto de procesamiento que incluye cálculo de flujo óptico y estéreo. Una gran cantidad de parámetros son analizados con el fin de relacionar los cambios estructurales (las decisiones de diseño) con su impacto en el rendimiento del sistema final. Se ha obtenido un buen compromiso entre prestaciones y coste.

- El estudio de los parámetros del sistema concluye con una interesante comparación de tres arquitecturas diferentes: una de costo bajo, una mediana y una versión de alto rendimiento.

- La implementación de la solución de medio coste en una plataforma de coprocesamiento alcanza una potencia de cálculo muy elevada. El sistema consta de 2221 unidades de procesamiento en paralelo en el mismo chip, un rendimiento de 92,3 GigaOPS (con un reloj de sistema a 42 MHz) y un alto numero de operaciones ejecutadas por vatio de potencia consumida (aproximadamente un 12,9 GigaOPS / W). Esto representa una excelente potencia de procesamiento frente a un bajo consumo de energía.

- La solución de altas prestaciones alcanza un rendimiento aun más elevado pero con costes muy altos en términos de consumo de recursos. Esta arquitectura consta de más de 3000 unidades de procesamiento que consiguen procesar hasta 165 GigaOPS (30 GigaOPS/W).

# Bibliografía

[1] E. R. Kandel, J. H. Schwartz, T. M. Jessell, Principles of Neural Science, 4th Edition, McGraw-Hill Medical, 2000.
URL http://www.worldcat.org/isbn/0071120009 2, 3, 4

[2] C. Weems, Architectural requirements of image understanding with respect to parallel processing, Proceedings of the IEEE 79 (4) (1991) 537 –547. doi:10.1109/5.92046. 2, 3

[3] Web page of mobile eye, Website, http://www.mobileye.com (2010). 3

[4] K. R., M. B., FPGA coprocessing architectures for military applications, Military embedded systems. 3

[5] European machine vision association web page, Website, http://emva.org (2010). 3

[6] M. Anguita, J. Diaz, E. Ros, F. Fernandez-Baldomero, Optimization strategies for high-performance computing of optical-flow in general-purpose processors, Circuits and Systems for Video Technology, IEEE Transactions on 19 (10) (2009) 1475–1488. 5

[7] Nvidia web page, Website, http://www.nvidia.com/object/cuda_home.html# (2010). 6

[8] D. Scharstein, R. Szeliski, A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, Inter. J. of Computer Vision 47 (1-3) (2002) 7–42. 6, 7

[9] J. Barron, D. Fleet, S. Beauchemin, Performance of optical flow techniques 12 (1) (1994) 43–77. 6

[10] Middlebury vision, Website, `http://vision.middlebury.edu/stereo/` (2010). 7, 9

[11] M. Z. Brown, D. Burschka, G. D. Hager, S. Member, Advances in computational stereo, PAMI 25 (2003) 993–1008. 7

[12] M. F. Tappen, W. T. Freeman, Comparison of graph cuts with belief propagation for stereo, using identical mrf parameters, in: In ICCV, 2003, pp. 900–907. 7

[13] D. Bilsby, R. Walke, R. Smith, Comparison of a programmable dsp and a FPGA for real-time multiscale convolution, in: High Performance Architectures for Real-Time Image Processing (Ref. No. 1998/197), IEE Colloquium on, 1998, pp. 4/1 –4/6. 8

[14] J. Díaz, E. Ros, R. Carrillo, A. Prieto, Real-time system for high-image resolution disparity estimation, IEEE Trans. Image Processing 16 (1) (2007) 280–285. 8

[15] J. Díaz, E. Ros, R. Agís, J. L. Bernier, Superpipelined high-performance optical-flow computation architecture, Computer Vision and Image Understanding 112 (3) (2008) 262–273. 8

[16] S.-K. Han, S. Woo, M.-H. Jeong, B.-J. You, Improved-quality real-time stereo vision processor, in: VLSID '09: Proc. of the 2009 22nd Inter. Conf. on VLSI Design, 2009, pp. 287–292. 8

[17] Drivsco project, Website, `http://www.pspc.dibe.unige.it/~drivsco/` (2007). 10

[18] M. Vanegas, M. Tomasi, J. Díaz, E. Ros, Multiport abstraction layer for FPGA intensive memory exploitation applications, submitted to Journal of System Architecture (2009).

[19] M. Tomasi, F. Barranco, M. Vanegas, J. Diaz, E. Ros, High performance optical flow architecture based on a multiscale and multi-orientation phase-based model, submitted to IEEE Trans. on CSVT.

[20] M. Tomasi, M. Vanegas, F. Barranco, J. Diaz, E. Ros, Real-time architecture for a robust multiscale stereo engine, submitted to IEEE Trans. on Image Processing.

[21] M. Tomasi, F. Barranco, M. Vanegas, J. Diaz, E. Ros, Fine grain pipeline architecture for high performance phase-based optical flow computation, submitted to Journal of System Architecture.

[22] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, Arquitectura multiescala de cálculo de flujo óptico basado en la fase, iX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009 (Sept. 2009).

[23] M. Tomasi, J. Díaz, E. Ros, Real time architectures for moving-objects tracking, in: ARC, 2007, pp. 365–372.

[24] M. Tomasi, M. Vanegas, F. Barranco, J. Díaz, E. Ros, A novel architecture for a massively parallel low level vision processing engine on chip, iSIE2010. Accepted for IEEE Internetional Symposium on Industrial Electronics, Bari (Italy) (July 2010).

[25] F. Barranco, M. Tomasi, M. Vanegas, S. Granados, J. Diaz, Entorno software para visualización y configuración de procesamiento de imágenes en tiempo real con plataformas reconfigurables, iX Jornadas de Computación Reconfigurable y Aplicaciones. JCRA2009. (2009).