Universitat de Girona

VLSI ARCHITECTURE FOR MOTION ESTIMATION IN UNDERWATER IMAGING

Viorela Simona ILA

ISBN: 84-689-8422-I
Dipòsit legal: GI-508-2006

Department of Electronics, Computer Science and Automatic Control

PhD Thesis

VLSI Architecture for Motion Estimation in Underwater Imaging

Dissertation submitted by:

Viorela Simona Ila

Supervisors:
Prof. Joan Batlle and Dr. Rafael Garcia

June 2005

To my family and to all those
who have offered me their support.

Acknowledgements

The writing of a dissertation can be a lonely and isolating experience, yet it is obviously not possible without the personal and professional support of many people. I must start expressing my gratitude to both supervisors Prof. Joan Batlle and Dr. Rafael Garcia. None of this would have been possible if Joan hadn't said "Does someone from Cluj wants to come to Girona to make part of our research team?" and I've been one of the lucky students having this chance. I would like to thank him for offering me professional and moral support, enthusiasm and valuable advices from the first day I arrived in Girona till the end. I also would like to thank to Rafa, who provided me with the guidance I needed to avoid going too far astray and with the freedom I needed to find my own solutions. I also appreciate his effort reviewing all my work and giving me his appreciable advises.

I would like to especially thank Dr.François Charot from the Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) France, for welcoming me in his research group in Rennes. I thank him for his immensely helpful comments and suggestions and also for encouraging me to work in the field of parallel architectures.

In particular, I would like to thank to Xavier "Mangui", for never letting any of my questions go unanswered, for teaching me many interesting things about computer science and for offering his unceasing support and friendship.

My very special gratitude goes to all the people from Computer Vision and Robotics research group, starting with the adorable "quartette" XCuf, PepF, Pere and JordiF; my dear companions JPages, XMuno, XLadow, Marc, Radu, Carles, Arnau, Bet, Anna, Tudor, Andres, DavidR1, DavidR2, Pinsach and Marta for making my working time more pleasant than I ever thought it could be; without forgetting those who are now helping Joan at the rectorate QuimS and JoanM. I specially would like to thank to Lluis Magi for introducing me in the complicate world of VHDL language and hardware design and for offering his advice every time I needed it. I also thank to Toni, Tomas and Ingrid for their help concerning technical aspects and to Mar, Sonia, Isela, Montse, Marta and Anna for administrative affairs. Thank all of them for their good company and funny moments which I will never forget.

Thanks go out as well to the people I met in France. Thank all the colleagues

from IRISA and all friends from Rennes for their help and specially for the nice moments I spent there.

None of this would have been possible without the constant support of all my friends here, they have been like family for me. Very special thanks to Juani for his love and support, to Mar and Silvanita, for their unconditional friendship.

Last, but certainly far from least, I would like to thank family for understanding my choice. Even if they were far away, they have always been at my side. I specially dedicate this dissertation to them.

VLSI Architecture for Motion Estimation in Underwater Imaging

Resum

El treball desenvolupat em aquesta tesi aprofundeix i aporta solucions innovadores en el camp orientat a tractar el problema de la correspondència en imatges subaquàtiques. En aquests entorns, el que realment complica les tasques de processat és la falta de contorns ben definits per culpa d'imatges esborronades; un fet aquest que es deu fonamentalment a il·luminació deficient o a la manca d'uniformitat dels sistemes d'il·luminació artificials. I és en aquest context on els treballs de la comunitat científica avancen en adaptar, a aquestes condicions extremes, els algoritmes i mètodes més àmpliament utilitzats.

La solució proposada en aquesta tesi permet detectar, en temps real, correspondències de parells de punts entre diferents imatges. Un problema fonamental en robòtica submarina és estimar el moviment d'un AUV a partir de les diferencies percebudes entre imatges successives captades per una càmera muntada en el propi vehicle. Aquestes diferencies es poden estimar analitzant correlacions entre paràmetres característics, i fonamentalment per tècniques de *matching*. En aquest sentit, la tesi proposada aporta una solució que permet, en temps real, detectar aquesta correspondència.

Criteris basats en diferències d'intensitat tal com la suma de les diferencies absolutes SAD (*Sum of Absolute Differences*) han estat sovint els més utilitzats per resoldre el problema de la correspondència. No obstant, quan es tracta d'analitzar seqüències reals d'imatges subaquàtiques, mètodes més complexes tal com el criteri de correlació normalitzada MNCC (*Mean Normalized Cross Correlation*) s'han mostrat com a més eficaços. Anant més a fons, quan les condicions esdevenen encara més complexes pel fet de trobar-nos amb rotacions o objectes en moviment (peixos, algues, etc.), la presència de punts erròniament correlacionats augmenta considerablement. En aquests entorns cal treure profit de la textura, la qual és una font especialment rica per a la caracterització de paràmetres invariants.

I és en aquesta línia en la que la present tesi complementa la informació de nivells de gris amb l'anàlisi de textura, mitjançant un procediment innovador que

permet eliminar els punts erròniament correlacionats per etapes de *matching* prèvies. Aquest procediment s'ha mostrat especialment robust i fàcilment aplicable per rebutjar les falses correspondències conegudes com a *outliers*. En comparació amb mètodes probabilístics, aquesta aportació no requereix de cap procediment a priori d'estimació de moviment.

Trobar correspondències de punts entre dues imatges requereix tasques de processat previ de baix nivell que portin a la detecció de característiques singulars de les imatges i facilitin l'aplicació posterior d'algorismes de correlació. Aquestes tasques solen ser repetitives i afecten a un nombre molt elevat de dades, per la qual cosa, quan es tracta d'obtenir resultats en aplicacions reals es requereixen sistemes potents de càlcul, i molt especialment quan es desitja obtenir resultats en temps real. Aquests condicionants són els que fan especialment recomanables els dispositius reconfigurables, els quals aporten una nova dimensió als processament d'imatges en temps real, i sobre els quals es presenta en aquesta tesi un estat de l'art dels sistemes existents. Les característiques dels nous dispositius programables tal com la seva reconfigurabilitat, l'elevat nombre d'entrades i sortides, la integració de blocs funcionals complexes i un número elevat de blocs de memòria han obert perspectives de processament d'una flexibilitat mol elevada.

Els esforços per implementar algoritmes de correlació en un sol xip i que operessin en temps real han estat considerables. En aquest àmbit, aquest treball també presenta un anàlisi comparatiu d'arquitectures sistòliques per detecció de moviment que han estat sovint utilitzades en aplicacions de codificació de video. En aquest camp, el present treball proposa una nova arquitectura VLSI dissenyada per a realitzar, amb un alt grau d'efectivitat, operacions de maching en aquests entorns complexes d'imatges subaquàtiques. L'algorisme proposat està dividit en dues parts fonamentals: el detector de punts d'interès i el procediment de detecció de correspondències. La primera etapa de l'arquitectura detecta N punts de la imatge adquirida per la càmera. Una segona etapa del sistema correlaciona els punts detectats en la etapa anterior entre dues imatges adquirides en instants de temps diferents.

L'algoritme de *matching* està dividit en diferents parts executades en un conjunt complex processadors concurrents. La disposició o estructura dels processadors està escollida de forma que es redueixi el numero d'accessos a la memòria i el temps d'execució, alhora que faciliti el posterior processament de les dades. Els assaigs per demostrar la viabilitat i fiabilitat de l'algorisme, així com la seva optimització,

s'han realitzat en seqüències d'imatges submarines reals.

L'arquitectura desenvolupada ha estat implementada mitjançant plataformes basades en dispositius reconfigurable. Les noves eines de disseny digital han permès analitzar en detall el comportament de l'arquitectura i optimitzar els recursos en funció dels resultats obtinguts en cada moment. L'algorisme ha estat parametritzat per permetre tenir flexibilitat en el tractament d'imatges de mida variable, així com en la selecció del nombre de punts a tractar i de paràmetres de correlació diferents.

En resum, els objectius aconseguits en aquesta tesi es poden remarcar en dues grans direccions:

- Millorar l'algorisme d'estimació de moviment proposant un nou mètode que introdueix paràmetres de textura per rebutjar falses correspondències entre parells d'imatges. Un seguit d'assaigs efectuats en imatges submarines reals han estat portats a terme per seleccionar les estratègies més adients.

- L'ús de paral·lelisme per accelerar algunes parts de l'algorisme d'estimació de moviment amb alt cost computacional. Amb la finalitat d'aconseguir resultats en temps real, es proposa una innovadora arquitectura VLSI, la qual ha estat prototipada mitjançant plataformes reconfigurables basades en FPGAs.

Contents

List of Figures

List of Tables

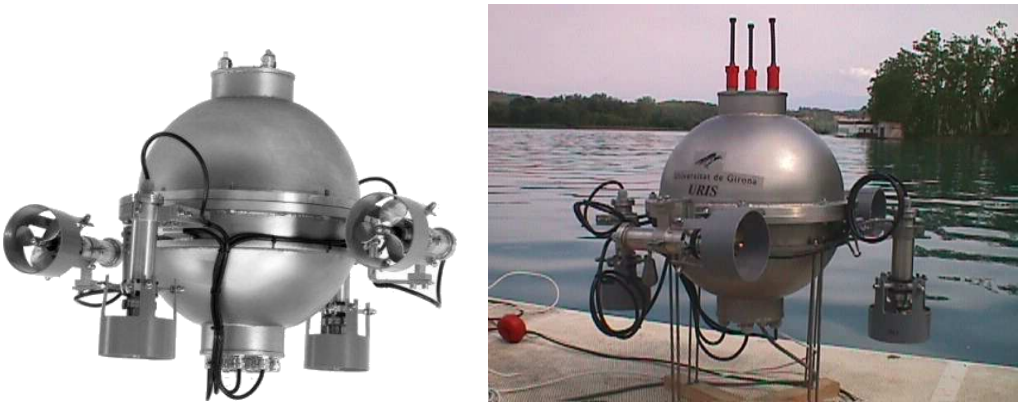
Chapter 1

Introduction

1.1 Systems for underwater environment exploration

Underwater exploration opens new perspectives to an amazing world different from our own. Water environments such as rivers, lakes, dums, seas or oceans cover about 78% of the surface of the Earth. Exploring these zones is still a great challenge for the human being. Understanding underwater environments is essential for ocean science, observation of oceanographic and geothermal events, exploitation of offshore life and mineral resources, undersea agriculture, ocean mining, oil industry or military purposes.

Whatever the environment is, land, air or sea, the aim of *robotics* is to develop tools that can be used to facilitate the work in real-world domains. Nowadays, *underwater robotics* receives a considerable amount of attention as it allows us to explore areas which are extremely difficult for humans to rich. Scuba divers and advanced diving techniques, remotely operated and autonomous robots, manned submersibles or underwater observatories are used to explore the underwater environment. Since first manned underwater vehicle called "The Turtle" built over 30 years ago, the research community has provided more and more sophisticated diving technologies, including human occupied submersibles, Remotely Operated Vehicles (ROV) and Autonomous Underwater Vehicles (AUV). There is no doubt that ROV and AUV are very important for exploring underwater environments. They reduce



(a) (b)

Figure 1.1: URIS underwater vehicle: a) Photograph of the URIS robot; b) URIS robot on the dock at Banyoles lake.

risks of offshore exploration and, due to their smaller size, they can perform missions that other craft cannot. Among them, small-sized underwater robots are very appropriate for these tasks due to their low-cost, safety and convenience (they are easy to manipulate). URIS (Underwater Robot Intelligent System) (see figure 1.1) is an example of a small-size, low-cost underwater vehicle. This robot is a research prototype platform designed and built at the Computer Vision and Robotic Group of the University of Girona. URIS has been successfully used in many experiments dealing with either intelligent control system applications or underwater image processing.

Autonomous Underwater Vehicles (AUVs) are unmanned, untethered submersible robots that are capable of carrying out missions autonomously. The robot *localisation* is a key problem when a robot must perform a mission autonomously. During a mission, a robot must know "where it is" so it can decide "what to do next". Relative and absolute measurements give feedback about the robot driving actions and the state of the environment around it. There are many sensing technologies available to provide the vehicle with knowledge about its position. Among them, vision based systems represent a good option due to their low cost, high-rate and high resolution information. When images acquired by a camera mounted on the vehicle(see figure 1.3), are processed adequately they can provide rich information about the environment and also about the position of the robot. Nevertheless, in most cases information provided by the camera is complemented by information from other sensors, such as acoustic sensors, gyroscopes and accelerometers, etc., in

Level	**Characteristics**	**Examples**
Low	Small neighbourhood, data access, simple operations, large amount of data.	Edge detection, filtering convolution.
Intermediate	Small neighbourhood, more complex operations.	Point matching, relaxation, segmentation and labelling.
High	Nonlocal data access, complex operations.	Object recognition, 3D reconstruction, localisation.

Table 1.1: Computer Vision

order to provide a more robust and reliable sensing system.

1.2 Computer vision

The goal of computer vision is to automatically extract information about a given scene by analysing images taken of the scene. The advances in computer vision algorithmic have resulted in many robotic applications integrating a vision based system into their structure. Visual sensors are potentially the most powerful source of information among all the sensors used on robots to date. The advantage of video cameras is that they have a of wide range of perception and a passive nature, which means that they do not transfer energy to the environment.

In general, computer vision tasks can be grouped into three levels low-level, intermediate-level and high-level. *Low-level* tasks consist in pixel operations such as filtering, edge detection or convolution algorithms. Tasks at this level are characterised by a large amount of data-pixel, small neighbourhood operators and relatively simple operations. Pixel-grouping operations like segmentation, region labelling or matching algorithms belong to the *intermediate-level*. These kind of tasks are also characterised by local data accesses and more complex algorithms. *High-level* tasks are more decision-oriented and are able to interpret the scene. A more schematic description of this classification is presented in table 1.1.

Image processing can reach a high-level of complexity, due either to the large amount of computation it requires or due to the fact that the problems are often ill-defined. In case of using image processing facilities in an on-line application, for example localisation of a robot performing a mission, powerful systems are needed to perform these data-intensive operations.

1.3 Motivation

The framework of this thesis is underwater robotics, in particular, a vision system for motion estimation and localisation of an underwater robot. Motion estimation involves operations from low-level computer vision such as feature detection and matching algorithms. When an underwater vehicle undertakes a mission alongside the ocean floor, images taken by a down-looking camera attached to the robot can provide rich information for the robot's navigation system. In order to recover the motion from a sequence of underwater images, point-features should be detected in the current image. A matching algorithm can be applied to find their correspondences in the reference image. Although this correlation technique provides good results in standard images, it may lead to detecting incorrect correspondences in underwater sequences. This aspect has led us to investigate the possibility of applying textural characterisation methods to eliminate bad-correlated points.

Motion estimation is often referred to as one of the most demanding operation in computer vision. The high computational load of computer vision algorithms can lead to a slow response time of such systems. One of the main objectives of this work is to obtain frame-rate performance for the execution of computer vision tasks. As the technology has evolved, the scientific community has come up with new ideas for implementing computer vision tasks with the aim of reducing the response time of the system.

Many image processing algorithms can achieve high performances by processing the tasks with high computational burden in parallel. Parallel processing can involve either hardware or software. This thesis investigates the possibility of accelerating parts of these algorithms by means of a hardware implementation of parallel architectures for motion estimation algorithms. This framework is used to give a new derivation of classic parallel Very Large Scale Integration (VLSI) architectures for

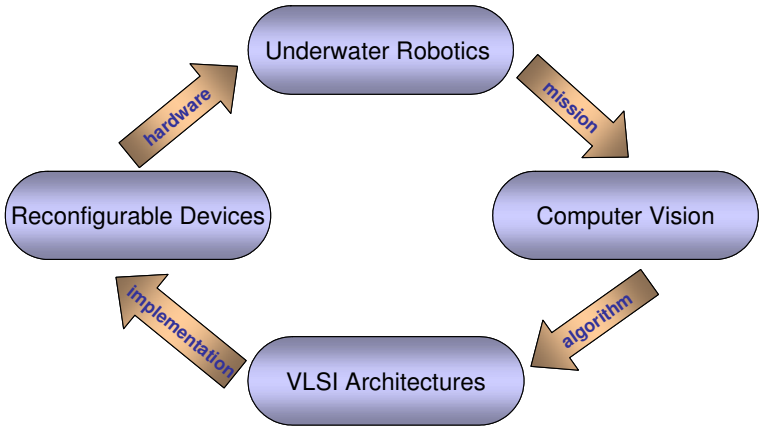


Figure 1.2: Research fields involved in this thesis.

(a) (b)

Figure 1.3: Samples of underwater images.

motion estimation, into a method which has been proved to work better in underwater imaging. Recently, reconfigurable devices such as Field Programmable Gate Arrays (FPGA) have become a viable option for implementing image processing algorithms. There are many reasons why reconfigurable devices are suitable to be used in image processing application and we will discuss them later on this thesis. Most important fact to be mention here is that these devices can achieve a much higher performance than software while maintaining a high level of flexibility.

1.4 The objectives of this thesis

The goal of this thesis is to continue the work carried out by the VICOROB[1] research team from the University of Girona in the field of underwater imaging. The objectives are distributed into two main directions:

a) To improve the motion estimation algorithm by proposing a new method which introduces texture measurements to reject "bad correspondences" between image pairs. Tests using underwater images were performed to select the most suitable strategy in this direction.

b) To speed-up the parts of the motion estimation algorithm that have a computationally high load, by using parallel implementation. A new VLSI architecture is proposed with the aim of achieving real-time processing. Reconfigurable platforms such as FPGAs have been used to prototype and test the architecture.

In order to achieve these goals several studies have been carried out during the research process:

- An exhaustive analysis of the vision based motion estimation algorithm applied to underwater images has been performed.
- The state of the art in reconfigurable systems for image processing purposes has been studied and presented in this thesis.
- A survey of VLSI architectures for motion estimation has been completed focusing on the structures which are suitable for our application.

1.5 Organisation of the thesis

This thesis integrates knowledge from several fields such as underwater robotics, computer vision, VLSI architectures and reconfigurable devices as shown in figure 1.2. Underwater robotics is the motivation of this work, even though computer vision and parallel VLSI architectures play the most important role. Algorithms

[1]From catalan: "VIsió per COmputador i ROBótica"

from computer vision are mapped into a VLSI architecture which furthermore targets reconfigurable devices. The thesis is organised into seven chapters including this introduction. Chapter 2 provides a technical background in field of reconfigurable devices and parallel approaches for image processing. The third chapter is a review of related works in the field of image processing algorithms targeting reconfigurable platforms. Solutions for increasing the performance of hardware implementation of image processing algorithms and new trends in this field are analysed in this chapter. Chapter 4 makes an overview of the algorithm for motion estimation and localisation of an underwater robot, detailing and analysing the parts to be accelerated by means of hardware implementation. A new method for rejecting bad correspondences based on texture analysis is proposed.

Mapping the algorithm into parallel architecture has been inspired by some similar VLSI approaches applied to motion compensation for multimedia standards. Chapter 5 highlights some of the most important works in this field. Our proposal for a new hardware architecture to estimate motion in underwater imaging is presented in chapter 6. An extensive analysis of the implementation of the system is also performed in this chapter. Finally, conclusions, future work and related publications are presented in chapter 7.

Chapter 2

Parallel Image Processing: Concepts and Solutions

2.1 Introduction

This chapter introduces some important concepts and taxonomies which will further be used in the following chapters of this thesis. We start by introducing in the field of parallel image processing in section 2.2. There are several different choices for designers when implementing image or signal processing applications. Real-time imaging systems like the one analysed in this dissertation require high speed video processing. Treating images is a difficult task considering the size of the image and the complexity of the algorithms. We can have a clue about the computational cost of an image processing algorithm by considering a simple 3×3 convolution kernel applied to an image size of 512×512 pixels. This involves 2.5 million basic operations such as accessing data, multiplications and additions as well as storing the result into memory. Powerful systems are necessary to perform these data-intensive operations in real-time.

Section 2.3 of this chapter is an overview of reconfigurable systems in the context of being one of greatest challenge to achieve high-performances in image processing. This field involves various hardware structures. We discuss one of the most important of these structures in section 2.4 by introducing and examining Field Programmable Gate Array (FPGA) devices. We also look at the basic internal structure, characteristics of the new FPGA devices and design's phases and tools.

2.2 Parallel image processing

The history of parallel image processing began in the early $60's$ with the idea of emulating the main functionalities of the human visual system. It was obvious that this challenge would require more powerful processing systems. We would start this journey through the field of parallel image processing referring to a what Michael Duff stated in [18]: *"Many hands make light work is a well known saying, but then so is too many cooks spoil the broth"*. This means that increasing the number of processors does not necessarily reduce the cost or the time to execute a task.

As we mentioned in the introduction of this thesis, computer vision tasks can be grouped in three levels: low-level, intermediate-level and high-level [74, 73]. Tasks from low-level image processing deal with simple operations applied to a large amount of data. In the intermediate-level, tasks are characterised by more complex algorithms applied to one or more images input from the low level. High-level tasks are more decision oriented and are able to interpret the scene.

In order to understand the parallel processing concept, we will introduce some of the main taxonomies for computing machines. The most popular computer architecture classification was defined by Flynn [22] in 1966. The classification is based on the idea of a stream of information. Two types of information flow into a processor: *instructions* and *data*. Conceptually these can be separated into two independent streams. The four combinations are presented below.

Single Instruction stream, Single Data stream (SISD) :

This is the typical Von Neumann serial computer architecture. Only one instruction is executed each time. Nowadays, even in commercial computers, a small level of parallelism is used to achieve greater efficiency, pure SISD architecture is seldom met.

Single Instruction stream, Multiple Data streams (SIMD) :

Several processors execute the same instruction on different data. In a typical SIMD architecture a control unit is used to provide a single instruction to an array of Processing Elements (PEs). They execute the instruction using the data that is stored in the memory. This structure is suitable for low-level image processing where simple operations are applied to multiple data.

Vector-processors are a subclass of the SIMD systems. In vector-processors

instruction are applied to arrays of similar data rather than to single data items using specially structured CPUs. Vector processors execute the instruction using the data almost in a parallel way the so called "vector mode". Thus, they are several times faster than when executing in conventional scalar mode.

Multiple Instruction streams, Single Data stream (MISD) :

This type of architecture involves several processors which execute different instructions for the same datum. Taking into account that distinct PEs receive distinct instructions applied to the same datum, this category can be considered hypothetical and impractical. However, when the datum flows through a series of processing units *pipeline architectures* can be defined. The pipelining principle consists of partitioning the computation into subcomputations which can be executed independently in distinct modules.

Systolic arrays can be considered as highly pipelined MISD machines. In systolic architectures the data is rhythmically pipelined from the memory through a network of the PEs. Data is synchronised with a global clock, which controls all PEs. Systolic arrays can be organised into linear or two-dimensional structures. Many applications from intermediate-level image processing can be mapped onto systolic arrays to reduce the computational burden (eg. block matching algorithms for video compression).

Multiple Instruction streams, Multiple Data streams (MIMD) :

Several processors execute different instructions on different data. Each processor can access either its own or a shared memory. They can run the same or different instructions and process different data streams asynchronously. Considering the memory allocation these architectures can address shared memory or distributed memory.

- In the shared memory architecture, processors access a common memory. There are two ways of interfacing the processing elements to the memory. The PEs and the memory can communicate through a bus. A more sophisticated technique comprises a multi-layer interface network, which is responsible to interconnect the PEs with the pieces of memory. These kind of machines are used more in high-level than in low-level image processing. They are applicable to parallel image processing, where each PE must process a certain part of the image. They are not as efficient

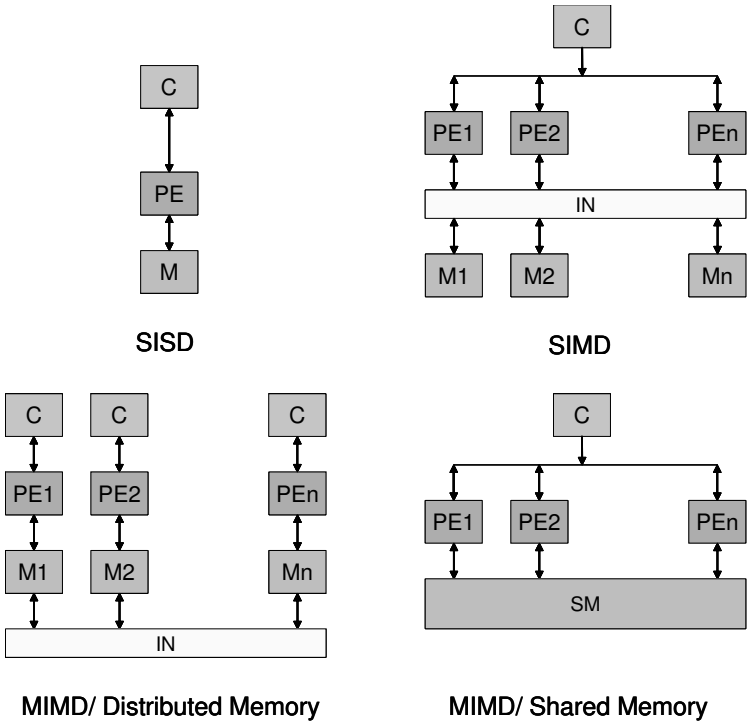


Figure 2.1: Flynn's taxonomy, computer architectures. (Control (C); Processing Element (PE); Memory (M); Interconnection Network (IN)); Shared Memory (SM)

as SIMD and the system can become very time-consuming due to the synchronisation requirements of the PEs.

- Distributed memory architectures consist of several processors interconnected in various ways. Each processor communicates with its local memory as well as with the adjacent processors. This type of machine can be applied to image processing algorithms which engage matrices.

Due to the nature of many image processing algorithms it is quite straight forward that an effective method for reducing the execution time can be based on parallel processing. Many image processing routines can achieve high performances with the addition of some processing elements working in parallel. A SIMD architecture for image processing can be composed of multiple PEs running the same instructions applied to a small part of the image. A pipeline processor is useful for processing a sequence of images. In this case one PE treats an image and passes the result to the next processor which can execute a different operation. More complicated image processing systems can be based on MIMD structure but this type of machines are

difficult to programme. Michael J.B. Duff [18] affirmed that Flynn's taxonomy is *"so crude as to be virtually useless"*, as many parallel architectures can fall into more than one of the forth classes presented above. However this is not the only taxonomy, other classifications can be found in the literature [87, 19, 52]. But the importance of Flynn's taxonomy is beyond doubt, as it is the starting point for all other classifications of processor's architecture.

Most of the time, the *parallel processing* term refers to how the system process the information. It can involve *hardware*, algorithms or *software*. Pipelining, multiprocessor architectures, interconnection between processing elements are topics of hardware parallel processing. Efficiency and concurrency techniques are algorithmic issues. Software tools are compilers and libraries for parallel computing. Parallel hardware architecture can effectively speed up a computation system to reach a performance level that is higher than that of a single processor. Nevertheless, mapping algorithms in hardware is in general a difficult task. Tools and compilers for automatising the parallelization have been developed to generate efficient code to be executed in a given hardware platform. When parallelism is introduced in the execution of an application it must fit the target architecture, which sometimes is constrained by the available technology. However, when we refer to parallel processing four main levels of parallelization can distinguished: job-level, task-level, instruction-level, gate-level.

Parallelism at job-level. A parallel computer is a computer having more than a one processor executing a single application simultaneously. *Supercomputers* are the most expensive and most powerful category of parallel computers [38]. They are typically used for scientific and engineering applications that must handle very large databases or do a great amount of computation like in image processing and computer graphics applications [71]. One of the leaders in visualisation supercomputer manufactures is Silicon Graphics. They develop powerful visualisation systems for many technical fields such as: modern science with data-intensive requirements, security and defence, modern design processes and advanced media applications. These powerful parallel systems are extremely expensive ($\sim$ 30.000$), but on the other hand supercomputers are the only systems in the world capable of solving great scientific and engineering challenges.

Parallelism at task-level. Parallel processing does not refer only to many processors working in parallel, software can do this, as well. In parallel program-

ming we have many programs processing the data in different ways at the same time. Thus, multiple programs can be executed in parallel, if there are no data dependencies. In the case of data dependency, several parallel programming models are commonly used: Shared Memory, Threads, Message Passing, Data Parallel or Hybrid [82]. Many software libraries for parallel image processing are available (MasPar's MPIPL). They are a set of routines which allow the user to perform operations on images in a parallel manner. Many implementations of well-known programming language compilers, such as C and C++, are used today for compiling high-performance computing applications based on parallel paradigms.

Parallelism at instruction-level. Parallelism at instruction-level can be also defined. Superscalar architecture refers to the use of multiple execution units, so that more than one instruction can be processed at a time [50]. These multiple parallel processing units are inside one processor. Most of the modern processors are superscalar. When the instructions are pipelined into the processor, there is a mechanism which selects the instructions which will be executed in parallel. Some compilers can make easier this process by sending the instructions in a proper way to the processor. Another way to achieve parallelism at instruction level can be seen in the case of vector processors. They provide high-level operations that work on vectors (linear or array) instead of a single data. A vector instruction is equivalent to an entire loop instruction. This type of processing avoids data hazards, reduces instruction bandwidth requirements and are obviously faster than scalar operations. In general Digital Signal Processor (DSP) devices have hardware which support the execution of vector operations. Advanced DSP processors integrates instruction parallelism where several RISC-equivalent (reduced instruction set computer) operations can be executed in parallel. These architectures allow the individual machine operations to overlap (addition, multiplication, load, store). Many of these fully programmable processors are used in multimedia applications as well as image coding and decoding (TMS 320 C6xx series) [85, 86].

Parallelism at gate-level. Hardware devices based on array architecture are able to execute a large amount of logic operations at the same time. In parallelism at gate-level, bit-level operations are executed in parallel. Moreover, the new devices in this category provide the necessary computing resources to meet the high-performance required in digital signal processing applications. An important advantage of gate-level parallelism is that the designer can implement as many parallel resources inside

the device as necessary to achieve the performance required by the system. The resources are not fixed like in general-purpose processors where each processor contains a finite number of basic computing functions. This thesis exploits the characteristics of gate-level implementation to test parallel architectures for image processing with the aim of obtaining frame-rate performance.

In spite of the large potential performance of parallel architectures, the image processing community does not benefit a hundred percent from high-performance computing. Essentially, this is due to the lack of optimised programming tools that can effectively help non-expert parallel programmers to develop multimedia applications for high-performance parallel architectures. The main objective of parallel processing is to wipe out the physical limits of serial processors by employing several processors working in parallel in order to reduce the execution time. Thus, a huge amount of research has been carried out in the field of parallel processing in the past decade, either concerning parallel hardware architectures or algorithms and programming languages.

2.3 Reconfigurable computing

For a given application, we can decide between implementing it using custom hardware or software design. The best choice, of course, would be to combine the advantages of both hardware and software:

Hardware:

- customised to the problem: no extra overhead for interpretation;
- relatively fast, as it uses high parallelism.

Software:

- flexible, tasks can be modified by changing the instruction stream in rewritable memory;
- general purpose computing.

Taking all of these characteristics into consideration the scientific community introduced the new concept of *reconfigurable computing* [12]. These systems combine the effectiveness hardware implementation with programmable processors to enhance the performance. Often these systems must also address the difficulties of both hardware and software, because they mix both technologies. Reconfigurable computing has existed as a concept since early sixties and it refers to systems incorporating some form of hardware programmability, customising how the hardware is used with a number of physical control points [10].

A reconfigurable system can integrate microprocessors, memory, I/O interface and *Reconfigurable Devices* (RD). Field Programmable Gate Arrays (FPGA) are reconfigurable devices which can bring a new dimension to digital system development. Advances in Very Large Scale Integration (VLSI) technology and design tools have had a great impact on the rapid evolution of reconfigurable devices in the past decade, increasing their performance and flexibility. Taking a look at the literature in this field, some important surveys such as [10] [12] [88] [91] place these devices in the category of digital signal processing (DSP) technologies between microprocessors and specific integrated circuits. While the Application Specific Integrated Circuits (ASICs), so called hardwired devices, and Programable Digital Signal Processors (PDSP) still play an important role in implementing mechanisms for DSP applications, we should be aware that RDs fill the gap between hardware and software, achieving better performance than software while maintaining a high level of flexibility.

As shown in figure 2.2, reconfigurable devices offer a compromise between the performance advantages of ASICs and the flexibility of software programable devices. Recent tendencies are clearly defined by an increase in flexibility and integrability and they are oriented towards reducing the design time. FPGA devices can be generically defined as functional platforms which offer the functional efficiency of hardware and the programmability of software.

Reconfigurable systems can use different combinations of microprocessors and reconfigurable logic (RL). Microprocessors can perform operations that can not be efficiently carried out by FPGAs. Compton et al. [10] defines four levels of coupling the reconfigurable systems: the RL and the processor are tightly coupled; RL used as a coprocessor able to perform operations without a constant supervision of the processor; attached RL which behaves as an additional processor and the loosest

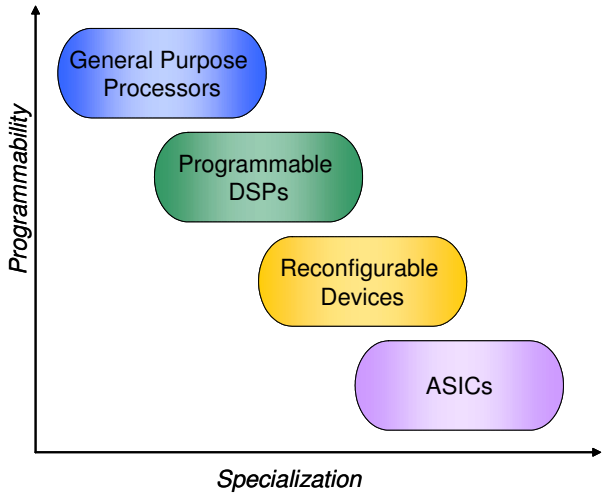


Figure 2.2: Digital devices on the scale of Programmability and Specialisation

coupling where the RL is an external stand alone processing unit. Each of these categories has pro and cons, choosing the most adequate for your design depends on the performance required and system constraints.

In reconfigurable computing the size of the computational basic blocks can vary from logic blocks like in FPGA devices to more complex structures such as arithmetic logic units (ALUs) or even CPUs. This aspect defines the granularity of the system.

Gate-level configuration is performed in *fine-grained* reconfiguration, thus the operations are performed at bit-level. In this case, individual tasks are relatively small in terms of computation size and execution time. These architectures are used to implement glue logic and irregular structures like state machines. Reconfigurable devices such as FPGA's are a typical hardware taking part of this category. These devices are configured at the logic-level which leads to complex hardware specification. However, existing compilers and Computer Aided Design (CAD) tools make this process easier and enable very high-level design specification. Hartenstein [35] claimed that this reconfiguration level appears as a methodology of logic design for hardwired logic but applied "on a strange platform" which is not really hardwired. While this type of devices was mainly used in prototyping hardwired systems, nowadays, their novel technological characteristics mean that they play an important role in reconfigurable systems. Data often does not use the whole word width, so, in the case of fine-grained solutions we are not forced to implement constant bit size, and

this can reduce the silicon area. On the other hand, bit-level implementation of arithmetic operators requires using of a big number of interconnection resources. Compared with full customised solutions, performance of synthesised operators is reduced. For this purpose new architectures integrate dedicated circuitry for multiplication and accumulation. Fine-grained solutions are developed and continuously improved by many hardware devices manufacturers [48, 42]. Their increasing performance, high-level of parallelism and high input/output throughput set them over other reconfigurable solution used in image processing applications.

Reconfiguration at the functional level is performed in *coarse-grained* architectures. Typically coarse-grained architecture consists of an array of configurable functional blocks called Data Path Unit (DPU) with a wide path width (8, 16 or 32 bits). As the functional blocks are optimised for some specific computations, they will perform these operations much more quickly. The number of basic blocks are several order of magnitude lower than in fine-grained computing solutions. This results in a major reduction of configuration data and memory and also reduces routing requirements, increasing the silicon area usability and energy efficiency. They can also provide wider data-path and more complex operations. On the other hand, due to their fixed functional block configuration, they are unable to optimise the size of the operands. Even though a large amount of research effort has been carried out in this field (RAPID and Chameleon projects) [20, 37], there are only a few commercially available coarse-grained reconfigurable systems [45]. In some research projects such as REMARC architecture [64] or RAW project [65], the functional units are in fact small processors grouped into a mesh structure and integrated into a single chip or multiple chips. A large variety of reconfigurable architectures are present in the literature from fine-grained to very coarse-grained or even heterogeneous structures. We will see later in this chapter that new reconfigurable systems provide multiplier functional blocks and even embedded microprocessors within the reconfigurable device [42, 48].

The following section of this chapter introduces the internal structure of FPGA devices, new technologies in this field and the main steps of the design flow.

2.4 Field Programmable Gate Arrays

When describing the basic architecture of a Field Programmable Gate Array (FPGA) we must take into consideration that the enhancements in their technology are quickly and continually being developed, this implies important changes in the structure of the devices every time a new family comes out on the market. Nowadays devices are for the most part derived from mid-80th FPGA technology. A basic FPGA device consists of a large array of configurable cells generally contained on a single chip. Each of these cells contains a computation unit capable of implementing one of a set of logic level functions and/or perform routing to allow inter-cell communication to take place. All of these operations can take place simultaneously across the whole array of cells. The first commercially available FPGA which appeared on the market in 1986 was the $XC2000$ family introduced by Xilinx. It had a very low capacity, less than 2000 gates and was very expensive.

A typical FPGA contains an array of individual cells called Logic Blocks (LB) interconnected by a matrix of wires and programmable switches. The user's design is implemented by specifying a simple logic function for each cell and selectively closing the switches in the interconnection matrix. The array of LB and interconnections between them form basic building blocks for logic circuits. Complex designs are created by combining these basic blocks to build the desired circuit. Apart from this, an FPGA can also have internal memory, I/O blocks assigned to each pin and recently, clock management blocks, digital signal processing dedicated circuits and hard-embedded microprocessors.

2.4.1 Basic inner structure

Each logic block has one Look Up Table (LUT), optional D flip-flops (DFF) and some fast carry logic. An N-input (usually $N = 4$) LUT is basically a memory that can be programmed to compute a function up to N-inputs. The DFF can be used for pipelining, registers, state-holding or any other situation where the clock is required. Set/reset lines and clock signals are normally global signals routed in special resources. The fast carry logic is a special resource to speed up carry-based computation like addition, parity, wide AND operations, etc. A basic structure of the LB with a 4-input LUT, DFF and a carry logic is shown in figure 2.3. The inter-

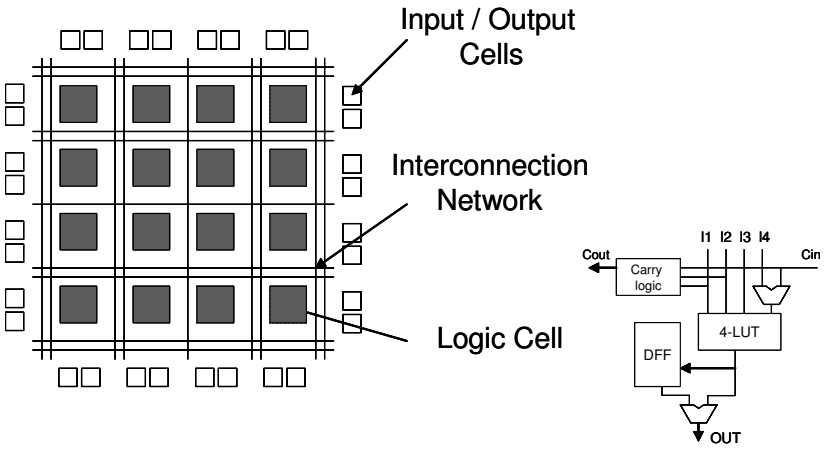


Figure 2.3: Generic FPGA architecture.

cell communications take place through interconnection resources. The outer edge of the array consists of special blocks capable of performing certain I/O operations to and from the chip. The architecture of a typical FPGA is illustrated in figure 2.3.

2.4.2 Characteristics of new FPGA devices

Altera [42] and Xilinx [48] are currently the leading FPGA vendors. This subsection does not pretend to compare the two current top FPGA families at the moment: Altera Stratix (see figure 2.4(a)) and Xilinx Virtex II Pro (see figure 2.4(b)). Its purpose is to highlight some important new characteristics of these devices, which must be taken into consideration in our design process. As we mentioned before, nowadays FPGA architectures can have additional internal memory, dedicated circuitry for digital signal processing, clock management blocks and hard-embedded microprocessors. A short introduction of these embedded components corresponding to both families is presented below. Some confusion can arise from there not being a consensus on the terminology used to describe device architecture. This is due to the different technologies being used by the two vendors. For instance, while in Stratix devices logic is organised in Logic Arrays Blocks (LABs) based on 10 Logic Elements (LE), Virtex II Pro architecture is based on Configurable Logic Blocks (CLBs) integrating four Logic Cells (LC). Benchmarks and comparison be-

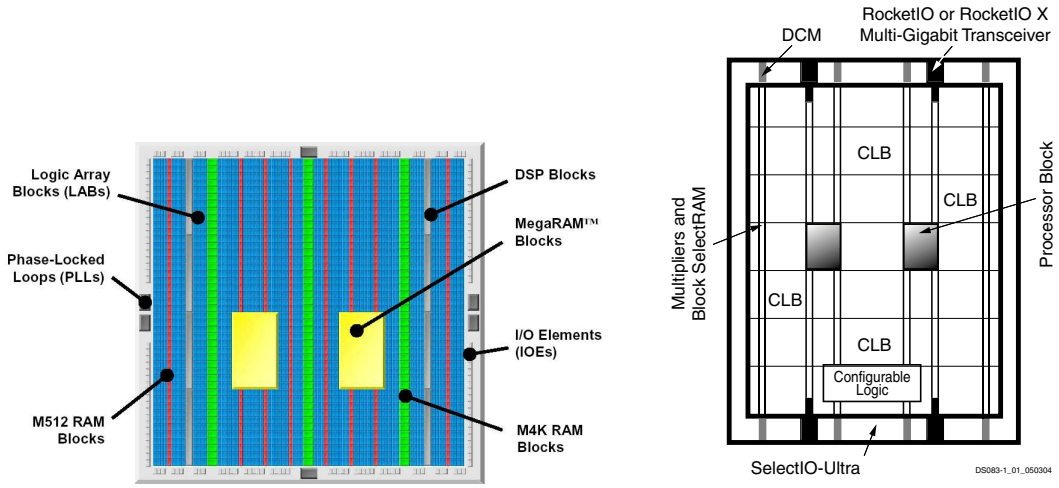


Figure 2.4: a) Altera Stratix device. b) Xilinx Virtex II Pro device.

tween these two families are provided by both the vendors and users. Controversial points of view arise due to the fact that making benchmarks is a difficult and expensive task and depends on many factors such as tools, user constrains, interpretation of the result, etc.

The Stratix family of FPGAs has densities up to 114,140 logic elements (LEs) and up to 10Mb of RAM. This device family offers up to 28 DSP blocks based on 9×9-bit embedded multipliers. Stratix devices support various I/O standards and also offer a complete clock management solution with its hierarchical clock structure through up to 12 phase-locked loops (PLLs).

The *internal memory* is organised in three types of RAM blocks called TriMatrix RAM: $M512$, $M4K$, and $M-RAM$ blocks. Although these memory blocks are different, they can all implement various types of memory with or without parity, including true dual-port, simple dual-port, and single-port RAM, ROM, and FIFO buffers. *DSP blocks* consist of hardware multipliers, adders, subtractors, accumulators, and pipeline registers and can be used for implementing high-performance digital signal processing applications. One DSP block can implement up to either eight full-precision 9×9-bit multipliers, four full-precision 18×18-bit multipliers, or one full-precision 36×36-bit multiplier with add or subtract features. These blocks also contain 18-bit input shift registers for digital signal processing applica-

tions. *Phase-locked loops* (PLLs) use several divide counters and delay elements to perform frequency synthesis and phase shifts. The PLL reconfiguration feature is useful in applications that might operate at different frequencies.

Similar characteristics appear in the Xilinx Virtex-II Pro family from Xilinx. The devices can have up to 125,136 Logic Cells, 8Mb RAM, up to 444 18×18 dedicated multipliers. One important new characteristic in this family of devices is that the devices can incorporate multi-gigabit transceivers and up to two PowerPC CPU blocks.

Block SelectRAM+ memory modules provide large 18 Kb storage elements of True Dual-Port RAM. *Digital Clock Manager* (DCM) blocks provide self-calibrating, fully digital solutions for clock distribution delay compensation, clock multiplication and division. Embedded IBM *PowerPC* 405 RISC processor blocks achieve performance up to 400 MHz. Embedded high-speed *serial transceivers* enable data bit rate up to 3.125 Gb/s per channel.

It should be noted that powerful new families like Altera Stratix II and Xilinx Virtex 4 are already available on the market. Transistors are becoming smaller and faster. 90nm technology has become a reality and opens the way to new solutions in FPGA design. In the last 40 years the feature size of silicon technology has been shrinking by a factor 1.25 each year, this is known as Moore's low. The implication of this low for FPGA devices were analysed by Vuillemin et al. [93]. They predicted that, if the device contained 400 logic blocks operating at 25MHz in 1992, in 2001 would contain 24k de logic blocks operating at 200MHz. This prediction has been reached and even passed by the last generation of FPGAs, which have achieved the incredible size of 180K logic blocks.

2.4.3 Advance in FPGA-based solutions

Programmable Digital Signal Processor (PDSP) devices have long been the best way to handle signal processing in many applications. But nowadays, the engineers often use these devices with a supporting role and even replace them by microprocessors or field programmable gate arrays (FPGA). Switching to FPGA devices can dramatically improve the performance. Computation in digital signal processing needs a lot of multipliers. In the latest FPGA devices like Altera's Stratix II or Xilinx's Virtex-4 several hundreds (400) of 18×18 multipliers are available and a lot of logic

gates ($9000K$). Computation that must be done sequentially using microprocessors can be done in a highly parallel way using FPGAs. It could be considered that FPGAs are more expensive and difficult to program than DSP or general purpose microprocessors. However, advances in technology have made it possible to decrease the FPGA price considerably (e.g. Cyclone II family from Altera and Virtex-4 by Xilinx). From designer's point of view, general purpose processors are still preferred, since they can be programmed with the widely used languages C and $C++$.

This drawback is overcome by integrating embedded "soft" processors such as Nios II processor from Altera and MicroBlaze & PicoBlaze processors from Xilinx. These processors are easy to be configured according to the system requirements and then mapped onto FPGA devices. New software tools for programming, building, debugging and running C applications on these embedded processors are also available. Design cost, complexity and power consumption are significantly reduced by integrating such processors together with embedded memory, peripherals and I/O interfaces in a single FPGA. In this context, the concept of System on Programmable Chip (SOPC) can be defined. Furthermore, FPGA family device such as Spartan from Xilinx integrates up to two embedded PowerPC microprocessor cores. New solutions in digital design are based on microprocessors tightly coupled with reconfigurable logic, embedded memories and wide throughput I/O interfaces. On the other hand, a lot of research is being carried out in the development of very high-level programming environments for FPGA. These environments should completely hide hardware details from the user point of view. They must also produce efficient FPGA configuration directly from a very high-level description language. Several structural design environments such as: Java HDL (JHDL), Handle-C, System-C, etc already exist. Although they are inherently sequential like C language, by instructing the compilers on how to build hardware to execute statements in parallel, these high-level languages target low-level hardware rather then microprocessors.

We can conclude by saying that nowadays, FPGA-based solutions offers a very high flexibility for design as well as a low-cost with very attractive performance capabilities. These devices are now built up with a combination of reconfigurable logic and programmable general-purpose microprocessors. Compilation tools for reconfigurable devices range from tools to assist a programmer in performing a hard mapping of the circuit to the hardware, tools to complete automated systems that take a circuit description in a high-level language and also tools to map it to

reconfigurable devices (e.g. DSP Builder from Altera).

2.4.4 FPGA design flow

The first step in a designing process is partitioning the application into parts which will be implemented in the software and computation which needs some hardware acceleration. The part that is implemented in hardware can be programmed at the register transfer level (RTL) or gate level circuit. Many Hardware Description Languages(HDL) exist, e.g. VHDL and Verilog. These HDLs allow designers to design at various levels of abstraction. VHDL is an acronym which stands for VHSIC Hardware Description Language. VHSIC is yet another acronym which stands for Very High Speed Integrated Circuit. VHDL is a standard (VHDL-1076) developed by the Institute of Electrical and Electronics Engineers (IEEE). The language has gone through a few revisions. Currently, the most widely used version is the VHDL'93 version. However, there is an old revision of the language referred to as VHDL'87. Verilog was created as a simulation language. Using Verilog for synthesis was a complete afterthought. Verilog became IEEE Standard *Std.*1364 in 1995. The final draft of Verilog-2000 was completed on March 2000, once IEEE approved Verilog HDL to be a standard called IEEE $Std.1364 - 2000$. As we mentioned in section 2.4.3, research is being carried out into developing sequential high-level description languages and compilers for hardware description.

Due to the increasing complexity of the applications it is no longer practical to manage million gate systems at the gate-level. "Design by reuse" is a new technique which includes using Intellectual Property (IP) components or so called Virtual Components (VC). IP cores help the designer by providing pre-tested, reusable functions that can be easily plugged into the design. They eliminate the need to "reinvent the wheel" reducing the design time and optimising it for the silicon device. However, this process requires complex methodologies to be developed, either by IP providers or IP integrators.

One of the most essential steps in the design methodology is to synthesise the HDL code. This process takes the conceptual HDL design definition and generates the logical or physical representation for the targeted silicon device. There are synthesis tools either for VHDL or for Verilog or other HDLs. There are also tools for both and even for combination of VHDL and Verilog. Synthesising a hardware

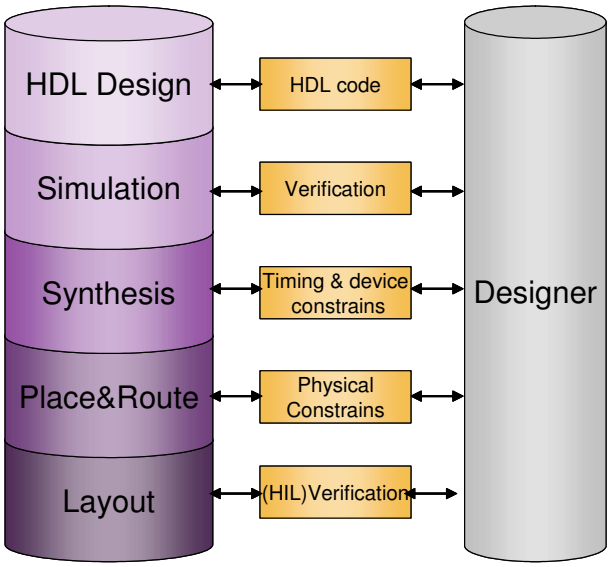


Figure 2.5: FPGA design steps.

description means mapping the circuit described using HDL into logic blocks within the reconfigurable devices. The resulting architecture must be optimised and must respect user-defined constraints regarding the targeted device and timing requirements. Verification must be performed to make sure that the design is correct before placing it into hardware. This is the most time-consuming phase of the design cycle. HDL code simulation is as important to the digital design as a debugger is to a software developer. When developing a complex design, each module can be separately simulated and validated before it is integrated. Functional and timing analysis must be performed during the simulation. Functional simulation determines if the logic in your design is correct before it is implemented in a device. Timing simulation shows how fast the circuit can be operated. It verifies that your design runs at the desired speed for your device under worst-case conditions. This process is performed after your design is mapped, placed, and routed for a given FPGA. The simulation can be done functional-first timing-after or viceversa. It is important to separate the complexity of functional correctness from the timing accuracy. It is usually faster and easier to correct design errors if there are performed functional simulation early in the design flow. The critical parts can be determined using human insight or by means of high-level simulations which can identify potential problems in the design.

The possibility of reconfiguration is one enormous advantage of FPGA design

compared with ASIC design which makes it possible to use Hardware-In-the-Loop (HIL) testing approach. In the case of ASIC design one mistake can costs hundreds of thousands of dollars and weeks of schedule time. For FPGA design, there is no such penalty, as the device can be reconfigured whenever necessary. In fact, FPGA devices are widely used nowadays for prototyping and verifying ASIC designs.

After synthesis, a Gate-level netlist is generated. Using the place and route (P&R) process, all the gates and flip-flops are placed and the clock tree synthesis and reset signal are routed for the target device. Output of the P&R tool is a file which contains the hardware bitstream configuration.

All these steps, when performed using an automatic compilation system, require minimum effort from the designer. After design specification using HDL code, synthesis and P&R are fairly straight-forward. The designer must define some device and time constrains and verify the design through simulation processes.

Chapter 3

Image Processing Algorithms on Reconfigurable Devices

3.1 Introduction

A variety of applications, including robotics, multimedia, quality control, assembly lines and security systems, require high-speed image processing. Considering that in most of these applications, one important goal of image processing is to scan objects or environments and make judgements at rates as fast as a skilled human observer can do, or even faster, powerful systems based on parallel processing must be developed. Due to the size of an image (e.g. 512×512 pixels), in order to process this amount of data at video rate (25 frames/sec) the system must be really fast. In the case of general purpose computers, achieving this performance rate is a big challenge.

As seen in the previous chapter, one solution is to use many general purpose processors working in parallel to execute computer vision algorithms. The size, cost and control requirements of these systems are considerable. The best performance, concerning execution time, can be achieved by programming the application at gate level, using Application Specific Integrated Circuits (ASICs). Typically this has always been a far more expensive alternative. On the other hand, tasks from the intermediate-level and high-level, algorithms are quite complicated and require a lot of flexibility in implementation, characteristic which is not found in ASIC devices. Reconfigurable devices combine advantages of both approaches, implementation at

a very low-level from ASIC devices, and the flexibility and rapid prototyping of software. Using reconfigurable devices such as FPGAs, the overheads associated with instruction fetch, decode and operand location are eliminated. This chapter analyses some important reconfigurable systems for image processing applications in relation to using FPGAs to accelerate performance. We do not pretend to cover all the applications, the aim of this chapter is to highlight the main FPGA-based concepts applied in image processing applications.

3.2 Review of FPGA-based solutions for image processing

For any imaging system of average complexity the biggest bottleneck in performance is the time taken to process each frame. Typically, image processing involves applying the same repetitive function to each pixel in the image, creating a new output image. A DSP device that aims to process one data-frame, fetches the data, performs the required mathematical operation, and then stores the result back in the memory. Usually, the whole frame is large to be stored in an on-chip memory and therefore, the data will need to be stored into an external memory. This adds several cycles to those required for mathematical operations.

A low-level image processing algorithm can be broken down into highly repetitive tasks which can be processed in parallel. When using an FPGA device, logic parts of the device can be mapped to execute these tasks in parallel. The important thing when using an FPGA is that the data rate through the FPGA is better than through a DSP. When the complexity overcome the device's capacity, several FPGAs can be linked together to increase the performance. New advances in the technology of these devices have made on-the-fly reconfiguration possible. When the reconfiguration time is short enough, the image processing algorithm can be split into several tasks which can be consecutively mapped into the device. The result of one task can be stored in the memory and then used as input for the next task. When computation complexity increases or decisions must be taken, many gates into the FPGA device are used. In these cases DSP devices still prove to be a highly effective tool. A solution is to mix both FPGAs and DSPs in one system. The latest FPGA device generation makes it possible to integrate these systems into one chip

(SoC). The advantage is huge considering the direct link between the devices and small integration surface. Several universities and research centres have explored the possibility of developing FPGA based systems to implement image processing algorithms. The solutions proposed by the scientific community are based on multi FPGAs boards, closely coupled DSP and FPGA devices, dynamically reconfigurable platforms and recently microprocessors, memory and logic all together in the same device (System on Chip). This section also introduces some new trends in the field, such as coarse-grained systems and evolvable hardware.

3.2.1 Multiple FPGA prototyping boards

SPLASH-2 computing machine

Splash-2 was one of the leading FPGA-based custom computing machine of all time designed and developed by Maryland SuperComputer Research Center [73, 74]. It uses a one-dimensional array of FPGA elements with a crossbar on a VersaModule Eurocard (VME) bus. A Splash-2 based system consists of 1 to 15 Splash-2 array boards, an interface board and a SPARC Station II host. Each array board contains 16 programmable elements (PEs), arranged linearly and fully connected through a 16x16 crossbar switch. Each PE consists of one Xilinx XC4010 FPGA and a 256Kx16 static RAM. A seventeenth control element regulates the crossbar network. An image processing system called VTSpals, has been developed based on Splash-2 general purpose platform. The system is made up of a video camera and a custom built frame-grabber card based on Splash-2 architecture. A SPARC Station host configures the Splash-2 processor and sends runtime commands intermixed with the video stream. Algorithms from different levels of computer vision have been mapped onto the system.

A simple one-dimensional convolution algorithm was implemented. A number of k PEs were chosen, which corresponds with the size of the mask ($k \times k$) . Each PE receives the pixel value and the partial result available from its left neighbour. The PE multiplies the pixel value with the mask assigned to it and adds the partial sum to it. This result and the pixel value are passed to the next PE. At the end of the systolic path, the result of the convolution algorithm is available. The Splash-2 implementation of this algorithm reduces the processing time by almost 300 times

compared to a C code execution on a SPARC Station 20. A page segmentation algorithm was implemented in Splash-2 based architecture to test intermediate-level image processing tasks. The goal of this application is to separate the text and the images from the background in a page. The sequential algorithm implemented in C in a SPARC Station 20, takes 90 seconds of CPU time to segment a 1024×1024 image. In a Splash-2 system it only takes 0.2 seconds for the same application. Fingerprint matching is another application tested using Splash-2 based system. A fingerprint is characterised by ridges and valleys. Both exhibit anomalies of various types, such as ridge bifurcations, ridge endings, short ridges, and ridge crossovers. All together, these features are called "minutiae". A lookup table (LUT) stores all possible points within a tolerance box that a feature can be mapped to. The LUT is computed for the query fingerprint minutiae on the host. For a 1 MHz clock rates, the Splash-2 system could perform around $6,300$ matches per second. Increasing the clock speed, the performance archived approximately $110,000$ matches per second.

The Splash system is one of the first reconfigurable architecture, having a high capacity spread over several FPGAs. The system's memory access is improved due to its distribution. In addition, the possibility of partial reconfiguration makes the system adequate for image processing implementation.

PAM computer

The PAM (Programmable Active Memory) computer utilise a fixed size of 16 FPGAs with global memory on a "turbo" channel adapter [7, 93]. PAM is a virtual machine, controlled by a standard microprocessor, which can be dynamically reconfigured into a large number of application-specific circuits. As its name suggests, Programmable Active Memory, the array is a kind of memory attached to the high-speed bus of the host computer, like any RAM memory, but it can also process the data between write and read instructions. The board is comprised of 23 Xilinx logic cell arrays (LCAs). The centre of the board is composed by a 4×4 matrix of LCAs. Each of them is linked to its four neighbours by a 16-bit-wide bus. Four on-board 256 words static RAM synchronous interface are used for storing purpose at a speed of 25MHz. These data buses are connected to four LCAs called switchers. Another two LCAs are used to provide the control and the address bus signals. The connection with the host uses a 32-bits/25MHz "turbo" channel bus.

Applications from stereo vision were implemented for testing the PAM computer. The stereo system was developed by the Institute de Recherche en Informatique et Automatique (INRIA), France to be used for correction the inertial and odometric navigation errors of a cart. The proposed implementation of the matching algorithm avoids any redundancies in the criterion computation and makes the processing time independent of the window size. For comparison purposes, the algorithm was implemented and tested using three different platforms: a SPARC-Station II, a PDSP device and the PAM architecture [93]. The software implementation computes the correlation between a pair of images in 59 seconds in SPARC-Station II. A dedicated hardware implementation using four DSPs performs the same tasks 6 time faster: 9.6 seconds. A PAM implementation of the same algorithm is 30 times faster: 0.28 seconds [21].

PARTS engine

Another powerful scalable reconfigurable computer is the PARTS engine built by Interval Research Corporation from Palo Alto, California. It consists in 4×4 array of FPGAs connected in a partial torus. Each array is associated with one-megabyte SRAMs, so that all SRAM can be accessed concurrently. PARTS engine fits perfectly on a standard PCI card in a personal computer or work station [94, 99]. If necessary, the multiple PARTS boards can be linked together in 4×8 or 4×16 FPGA array. One of the applications implemented in the PARTS engine was a depth from stereo vision algorithm that computes 24 stereo disparities in 320×240 pixel images at 42 frames per second.

Reconfigurable systems that are composed of multiple FPGAs require efficient connection schemes between the devices and also communication with the host and memory. Mesh and crossbar interconnection between FPGAs were tested [36]. In a mesh type connection, the nearest neighbours are connected. Sometimes it is necessary that for some signals to pass through one FPGA just to connect with non-neighbouring devices. The crossbar connection tries to overcome this drawback using special routing resources. While Splash-2 reconfigurable system was designed to support crossbar connection, the PAM and PARTS architectures are examples of mesh connections.

3.2.2 Combining FPGAs and general purpose processors

RETINA card

Complementing the FPGAs devices with DSP devices or microprocessors can be helpful in applications with more complex computations. Some examples are present in the literature. The heterogeneous image processing system called RETINA was developed at the University of Mining and Metallurgy from Krakow, Poland [30, 31]. The 32-bits RETINA card is used for image acquisition, processing and analysis. The system is composed of high-speed video A/D and D/A converters, a Virtex FPGA device working together with floating point Motorola 96002 DSP and a 32 bits PCI Master interface. The FPGA implements all the control logic and also performs image processing operations. The platform was initially designed for implementation of Log-Polar [1] remapping algorithm, but due to the flexibility of its components, it has become a universal platform for image processing and analysis.

Hierarchical LINDA

LINDA system was developed at the Université de Technologie de Compiègne, France [39, 63] and permits user friendly parallel programming and real time execution of applications. It consists of four parts. An image acquisition board with a 40 MHz TMS340C20 graphic processor is used to generate the video control signal for image acquisition and display. The low-level image processing board contains an array of 6×6 Altera FLEX 8000 FPGAs and interfaces directly with the image acquisition board and processing element board. Real-time edge detection and edge tracking algorithms were mapped onto this board. The output of this module can be stored back in the image acquisition board or in the shared memory of the high-level processor. The high-level image processing board is a parallel processing development system with a Texas TMS320C40 DSP processing element. The interconnection network permits the system to be reconfigurated and adapt to different algorithms in order to get the optimal performance. A recursive and multi-stage interconnection network was developed by using FPGA technology. A 3D scene reconstruction for mobile robot perception, was implemented to test the LINDA architecture.

Automated Image Recognition (AIR) system

Institute of Software Integrated Systems from Vanderbilt University of Nashville developed a real time embedded Automated Image Recognition (AIR) system based on FPGA and DSP devices [66]. The computational complexity of a distance classifier correlation filter (DCCF) algorithm needs a DSP processor to be integrated. The prototype system uses $10-14$ DSPs and $2-6$ FPGA modules. The two-dimensional Fast Fourier Transform (2DFFT) required by the algorithm was implemented using FPGA devices. One FPGA module (slave) implements a 32/128 point 1DFFT, while the other (master) breaks down the 2DFFT into two 1DFFT operations, supporting operands and receiving results from the slave. The 1DFFT was implemented using Altera's FFT megacore block. This is a scalable AHDL (Altera Hardware Descriptive Language) component, that contains multiplier and adder sub-units, which, together with scheduler logic that addresses on chip memory fulfills a butterfly operation for storing temporal results. The master module also controls the data flow of the 2D operations. The performance achieved by this prototype, at a maximum clock frequency of 12.84 MHz, is 0.9024 ms for a 32×32 2DFFT.

3.2.3 The codesign approach and experimental platforms

Partitioning an algorithm in hardware and software can be done either manually or automatically. In the systems presented above the algorithms were manually distributed over FPGAs and microprocessors. Moreover, the concept of hardware/software codesign can be introduced at a higher abstraction level. Codesign changed the way researchers approached reconfigurable hardware and also the way of designing high-performance applications. In codesign, the hardware mainly serves as an accelerator to reduce high computational burden or a real-time interface with the environment. Co-developers bring the power of software programming languages to programmable hardware targets. There is now a need for software tools to assist the co-design, as well as the cosimulation of hardware and software and the proper interface between them. When talking about codesign we do not only refer to implementing applications on a hardware platform based on FPGAs and general purpose processors, we are also referring to partitioning the design between the host processor and the hardware accelerator [81].

Riley-2 experimental platform for codesign

Riley-2 [62] is a flexible experimental platform for codesign and dynamic reconfiguration developed jointly by Imperial College, London and Hewlett-Packard. A collection of tools to facilitate rapid design, evaluation and validation was also developed. The platform is based on a general purpose processor, several FPGA devices and flexible interfaces such as PCI interface with the host or extensible external I/O. A model of the system which makes it possible to examine the interaction between the software and the dynamically changing hardware was developed in VHDL language as well as a commercial simulator for codesign. A number of real time image processing applications were developed using codesign in order to evaluate this platform. The designs could fit properly into the proposed system but the performance was poor, mainly because the camera used was a limitation(QuickCam).

3.2.4 SoC concept applied in image processing applications

Adopting new System-on-Chip (SoC) design technologies, traditional hardware-software codesign techniques need to be adaptated and revised to satisfy the growing needs of complex system designs. Characteristics of new FPGA devices make it possible to integrate soft and/or even embedded hard processors into an FPGA device. Software and hardware can work together in a platform which consists of a single chip. In this case, part of the application can be programmed and executed in a sequential manner (software approach) and some other parts can be accelerated by means of hardware implementation. As we mentioned above, this can be useful when application requires a decision-making process. New FPGA devices can be used as prototyping platforms for SoC systems in different kind of applications of image processing. On the other hand, hardwired SoC solutions for image processing have been proposed by the scientific community, some of them even bringing in improvements in timing and area in comparison with FPGA architectures.

Embedded reconfigurable array

Khawam et. al [54] proposed a heterogeneous platform with multiple processors and computational elements. A vision-based motion estimation algorithm was tested using this platform. The SoC system targets low-power multimedia applications

for future mobile devices. It integrates specific programmable arrays for motion estimation, a DSP processor and microcontrollers which communicate through an AMBA bus. This new approach consumes less power and occupy less chip area than FPGA solutions. Reconfigurable array are based on hardware multiplexers, adders, accumulators and comparator which can be used to implement a motion estimation algorithm in parallel [98]. The implementation was compared with ASIC, FPGA and ARM processor based solutions. The proposed architecture reduced power consumptions compared with FPGA and ARM processor, it had a 45% smaller area than FPGA and four times higher frequency than the ASIC implementation. It is important to point out that by replacing a fine-grained approach by a coarse-grained approach power consumption and silicon area can be reduced.

3.2.5 Run-time reconfigurable solutions

Computing solution which takes advantage of the dynamic nature of FPGAs, are called Run-Time Reconfigurable (RTR) solutions. They solve the scalability problems of the rapid prototyping techniques by adopting a divide-and-conquer approach. This means that, large problems are temporally broken up into stages, each fitting into the system. After one stage is executed, the output is stored in a memory and that part of the device is reconfigured for the next stage that processes the stored data. This process is repeated until all the required stages are executed and the final result is available. In this case the efficiency depends on the amount of time the system spends on reconfiguration. These kinds of systems are efficient when more time is spent in computation than on reconfiguration. Some examples are presented below.

Image interpolation algorithm

An example is the implementation of an image interpolation algorithm in a Xilinx XC6264 device [2, 49]. The interpolation problem is broken down into two parts: the Inverse Filter IF, and the Fast Spline Transform FST. Since the Xilinx XC6264 does not have enough resources to implement the entire computation, the algorithm has to be divided up into different stages to be executed sequentially in the platform. This is an efficient approach if the reconfiguration time is much smaller than the time required for each step to be executed. Partial reconfiguration is one of the

methods used by RTR applications designers in order to reduce the reconfiguration time. In this case, if parts of the algorithm from two consecutive stages are similar, after being loaded into hardware they can be reused in the next stage. Therefore, only the part that differ from the previous stage are reconfigured.

ARDOISE platform

Architecture Reconfigurable Dynamically Oriented Image and Signal Embedded (ARDOISE) [9, 53] is a prototyping platform based on an Atmel FPGA device used for evaluating the performance of an image processing dedicated architecture using dynamic reconfiguration (DR). The Atmel AT40k family has been chosen due to its reduced reconfiguration time, while it needs less than 1 ms for total device reconfiguration. The ARDOISE project is supported by ten French research teams and its main goal is to help in the research into the DR paradigm applied to image processing applications. The ARDOISE architecture is composed of three identical modules, each including one FPGA connected with two memories. A fourth FPGA is in charge of the configuration storage and scheduling, clock generations and communication with the host processor. In one of the tests, a part of the JPEG2000 standard for image compression was implemented in the ARDOISE board [9]. The algorithm was partitioned in order to exploit the dynamic reconfiguration of the FPGA device. Another test consisted in implementing an image segmentation algorithm. The architecture was reconfigured for each stage of the algorithm: noise filtering, edge detection, contour closing and region labeling. It achieved a performance of 31 ms for the whole algorithm applied to an image of 512×512 pixels.

The idea of run-time reconfiguration is to exploit temporal parallelism instead of spatial parallelism. The main objective is not to reduce the execution speed but to optimise the hardware usability. As image processing tasks require a large amount of hardware resources, this solution can be considered when the system has limited space.

3.2.6 Evolvable hardware

Another interesting idea to be considered is based on the theory of evolution from biological systems. The advances in technology in the field of reconfigurable de-

vices permits increasingly complex algorithms to be implemented in hardware. Applying evolutionary techniques to hardware design is called *Evolvable Hardware* (EHW) [6, 17, 90]. The main goal is to replace traditional design methods with evolutionary techniques for given applications which cannot be achieved using traditional methods. EHW applies the concept of genetic algorithms (GA) to reconfigurable devices like FPGAs in order to properly reconfigure the device through an evolutionary process. In the past, evolutionary computation methods have mainly been applied to software applications, but recently these methods have been successfully applied to the design of hardware circuits.

GAs provide the robustness necessary for efficiently searching for possible solutions to a given problem in a complex spaces [29]. Each "individual" can be evaluated using hardware simulation or hardware itself. The GA can be implemented either in a host computer or directly in the hardware. Taking in consideration these aspects we can distinguish three types of evolvable systems:

- *Extrinsic evolution* uses the software simulation of the hardware to evaluate the fitness value of each individual. This may be an advantage if you do not wish to be too technology specific and in this case the hardware model could be quite abstract. On the other hand if technology is the goal, then more accurate fitness values may be obtained. Since fitness is an important element in the evolution, abstracting the model can lead to a less optimal solution.

- *Intrinsic evolution* fitness evaluation, is based on a hardware implementation in which every individual is implemented and evaluated on the target technology. This approach can be used to explore properties of the technology which cannot be exploited using traditional design methods. The evolution process runs on a host computer responsible for selection and the performance of genetic operators. Each individual is downloaded to the chip as configuration data. The fitness of a given individual is evaluated by applying test vectors to the implemented individual and then calculating the fitness value from the response.

- In *Complete Hardware Evolution* the genetic algorithm (GA) is implemented in the same chip as the evolving design. The individuals from the GA are placed in a RAM and the fitness is evaluated in the same way as in the previous

approach. Since the GA and the evolving design are implemented in the same chip, the evolution process can observe the evolving design continuously.

Using complete hardware evolution, in which the evolution process is in the same chip, means that adaptation does not need to be decided on during the design process but rather the changes required in the circuit are free to evolve with the changing external factors. Choosing one of the three approaches presented of depends on the requirements of the application. The question that appear is "why do we need to design hardware circuits using alternative methods?". One of the main reasons is to look for better circuits, i.e. smaller, faster or less power consuming. A classical designer is limited to a set of mathematical models, rules, techniques, etc. In the case of EHW they can explore freely all possible circuits, and new unconventional designs can appear.

Khepera robot

In the Micro-engineering department of the Lausanne Polytechnic (Switzerland), an X6216 FPGA device set on a basic Khepera robot was evolved to perform navigation in a labyrinth, including an obstacle avoidance behaviour [69]. The objective was to use EHW as a motor controller for the same Khepera mobile robot. The robot can be able to move towards a pattern sensed by a camera placed on top of the robot. The EHW was implemented in an FPGA. The robot performs in a box with the pattern placed on one of the walls. Completely unconstrained evolution was performed on the FPGA using a classic genetic algorithm. The FPGA receives visual information coming from the camera and outputs the commands to be applied to the motors. The controller's performance is evaluated by monitoring the behaviour of the robot with an onboard program.

CAM brain machine

We should also mention one of the greatest projects in this field. Genobyte Inc. (Boulder, Colorado, USA) in collaboration with STARLAB from Brussels, Belgium, built the CAM-brain machine (CBM). This is an experimental machine composed of reconfigurable hardware, Xilinx XC6264 FPGA, capable of training and evaluating cellular automata based neural network modules directly in silicon. The main ob-

Nr.	Architecture	First Author	Characteristics
1	Splash-2 computation system	Ratha N. K. et al., 2000	**Multiple FPGA**
2	PAM computer	Vuillemin J. E. et al., Bertin, 1993	
3	PARTS engine	Woodfill J. et al., 1997	
4	RETINA card	Gorgon M. et al.,1997	**FPGA and General Purpose Processors**
5	LINDA architecture	Hou K.M. et al., 1995	
6	AIR system	Neema S. et al., 1997	
7	Relay-2 codesign platform	Mackinlay et al.,1997	
8	Image Interpolation Algorithm	Athanas P. M. et al., 1998	**Run-Rime Reconfigurable**
9	ARDOISE platform	Kessal L. et al., 2000	
10	Embedded reconfigurable array	Khawam s.et al., 1998	**SoC Concept**
11	Khepera robot	Roggen D.et al., 2000	**Evolvable Hardware**
12	CAM brain machine	De Garis H.et al., 2000	

Table 3.1: Image processing algorithms on reconfigurable devices.

jective of developing the CBM was to design a computationally powerful evolvable hardware research tool for the evolution of complex digital circuits [14, 15, 16].

3.2.7 Summary

Table 3.1 summarises the systems and architectures presented above, each category represents a solution for increasing the performance of hardware implementation of image processing algorithms or, in case of EH, new trends in this field.

Big breakthroughs in semiconductor technology have made developing high-speed and high-density FPGA devices possible. A few years ago, tasks were mapped onto multiple FPGA-based system, but now these tasks can be implemented using only a

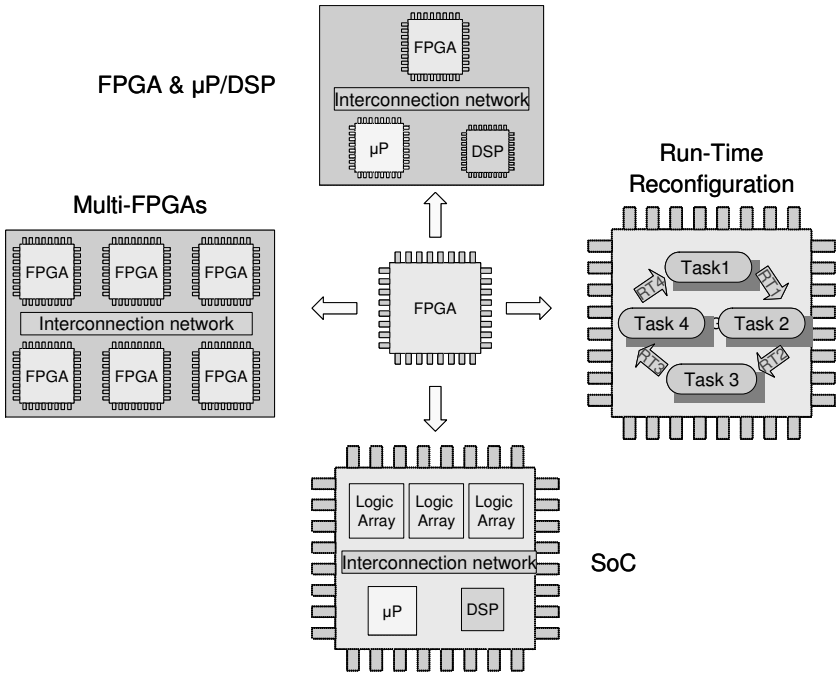


Figure 3.1: FPGA-based platforms and concepts.

small part of one powerful device. The features of new FPGA devices make it possible to integrate on-chip DSP and FPGA mixed architectures. Nowadays, higher reconfiguration speed can be achieved and the devices can be partially reconfigurated. These aspects can improve the performance and efficiency considerably. One of the larger contributions of the systems presented above (e.g. PAM, SPLASH2, etc.) consists in the way that image processing algorithms are mapped over several parallel processing elements and memory blocks in order to reduce redundances and to achieve high computation speed. Some of the systems presented here were designed to implement specific algorithms. Since the programming of an FPGA can be entirely or even partially changed by downloading a new configuration, these systems became general purpose platforms for image or any other signal processing. Comparing the algorithms execution in a hardware platform and in a microprocessor (or other general purpose processors) demonstrates that they provide orders of magnitude better performance. It has also been proved that some of them, such as the SPLASH-2 architecture, reached world-record performance for many applications [36]. As we can see, there are different forms in which this performance can be archived: using multiple-FPGA system, co-design, RTR or SOC solutions (see figure 3.1). On the other hand, EHW is a new concept for developing hardware

applications. In EHW the design is not particularly oriented towards optimising the performance, but rather exploring new hardware configurations which can be used in adaptive systems.

3.3 Commercial FPGA based platforms for image processing

Many commercially powerful FPGA-based platforms are currently available on the market and are mainly dedicated to rapid prototyping of hardware systems or development kits for educational purpose. The recent advances in parallel implementation of image processing algorithms has resulted in the rapid development of reconfigurable systems based on FPGA devices. Several companies, who were confident about the potential of FPGA devices in the field of signal and image processing, embedded this technology into their products such as frame grabbers or specific boards. One of the first commercial such system was the Spectrum $G800$ from the GigaOps corporation. This custom computing was a commercially available configurable video rate system capable of supporting a wide range of graphics, image processing and computer vision applications [28]. It can hold up to 16 $G210$ modules, each $G210$ module containing 2 Xilinx $XC4010$ FPGAs. FPGA configurations can be loaded within 20 ms. The video module of the $G800$ custom computer board contains a frame grabber and an encoder which enables S-video and Composite video I/O. Giga-Ops Spectrum $G800$ custom computer can compute the video data up to a rate of 320 MB/sec.

Nowadays, many powerful commercial image processing FPGA based systems are available on the market. For example, MaxRevolution [44] is Datacube's latest addition to the MaxRevolution family of frame grabbers based on FPGA processing capabilities. Pro-Design's VisioSpeedster2 [46] is the second generation of a product family of high-speed FPGA-based image processing cards for PCs. The system is based on a VIRTEX-II FPGA from XILINX and it has excellent memory resources and possibilities for expansion. Titan Corporation commercialises former VisiCom products such as VigraVision and VigraWATCH [47] product families. These are high-performance visualisation products for graphic and imaging based on FPGA technology from Xilinx. PCI Reconfigurable Image Advanced Processor (PRIAP) is

another real-time image processing board designed and manufactured by the SECAD company [41], specialised in Xilinx FPGA design. Annapolis Micro Systems [43] introduced Wildfire reconfigurable platform based on the SPLASH-2 image processing architecture presented above. FPGA devices from the board can be independently configured by the host processor.

Many other FPGA-based frame grabbers or imaging systems exist on the market. The main drawback of these powerful commercial systems is the incredibly high cost which make them incompatible for research application such as robotic prototyping systems. Sometimes, space and shape limitation are also important aspects of these systems. The idea we wanted to emphasise above is that FPGA devices play an important role in imaging systems and can be considered as a huge breakthrough in this field.

The latest generation of FPGAs platforms for hardware acceleration offers the possibility of implementation of computationally-intensive algorithms. Design teams taking advantage of these platforms intensify their research in the field of parallel algorithms for signal processing in general as well as for image processing. This is a reverse process, evolution in hardware determines evolution in algorithms and tools and viceversa. Recently, there have been advances in design tools supporting software-oriented design techniques for programmable hardware platforms. This makes life easier for software designers who can now take advantage of the power of FPGA-based platforms. In image processing, where the algorithms are more and more complex, helpful intellectual property (IP) available solutions considerably reduce the design time. However, this field is still underdeveloped. Only specific IPs are available for image processing and research continues to optimise image processing oriented high-level HDL compilers. What is beyond a doubt is that FPGA devices are low-cost, low-risk platforms oriented towards application prototyping and they can also be used in high-performance end-products.

Chapter 4

Vision-Based Motion Estimation as a Tool to Localise an Underwater Vehicle

4.1 Introduction

This chapter provides an overview of an algorithm for motion estimation and localisation of a submersible, detailing and analysing those parts which need to be accelerated by means of hardware implementation. Once the algorithm has been tested using general purpose processors, the necessity to speed up some parts with a high computational load emerged. A high-level implementation of the algorithm using MATLAB® has been carried out prior to its parallelization. The algorithm has many parameters. These have been tested a priori using tools oriented to image processing such as MATLAB® "Image Processing Toolbox", or C++ libraries such as Matrox Image Processing Library (MIL). As well as describing every step of the algorithm, this chapter also introduces a new method to improve the result of the correspondence problem in motion estimation. This method takes into consideration textural characterisation for rejection of bad correlated points.

4.1.1 Systems and sensors for localisation in underwater robotics

Localisation is an important problem when a robot needs to complete a mission autonomously. It is the process by which a robot or any other system determines where it is. Apart from image-based localisation systems many other technologies have been developed to provide information about vehicle position. These include inertial systems, doppler-based systems, acoustic transponder networks, global positioning systems and more.

Inertial navigation systems (INS) are composed of gyros and accelerometers. They are normally used in fields like civil and military aviation, cruise missiles, submarines and space technology. Because of these areas of operation, the systems and all their components have to be very precise and reliable (bias error$\simeq 0.015^{o}/h$). As a consequence, the costs for such a system are still very high (> 100.000\$) and the systems are not yet small enough to be used for small robotic prototypes.

Sound navigation and ranging (sonars) are based on the propagation of sound waves. They emit ultrasonic pulses and "listens" to reflect pulses from a potential obstacle. In underwater robotics they can serve to determine the distance from the sea-floor and also survey the area beneath the vehicle. The speed of the sound can be affected by different factors from the underwater environment: temperature, salinity and depth. Since it is an active sensor it might be not adequate for some applications. A sonar-based system can be used to build a map of the seabed. Sonar images can be used to build up a map of the environment. This map can be further used for the localisation of the robot. *Doppler effect based sonars* are based on the Doppler principle, which states that the frequency of the received signal differs from the frequency of the emitted one when the source and the reference point are in motion relative to each other. Such devices are called Doppler Velocity Log (DVL) and their precision is affected by various factors such as altitude from the seabed, signal power, frequency, pulse length, etc. The use of Dopplers as well as other sonars for slow moving vehicles is mainly affected by the variation in sound's speed depending on water conditions.

Acoustic Transponders Network (ATN) can be placed in the area where the mission is going to be carried out. The vehicle can interrogate all transponders simultaneously and each of them replies using its own frequency. The robot

can then calculates its position by means of triangulation. This method has several drawbacks, such as the fact that navigation is restricted to the area where the transponders are placed, the transponders must be carefully calibrated and, of course, the cost of installing and recovering the entire system in the area is high.

Global Positioning System (GPS) is a radio navigation system based on an array of 24 satellites orbiting the Earth and some antennas for ground support. They are outdoor localisation systems for small devices but GPS signals are not available to vehicles operating at depths where the GPS signal berely propagates through solid or aquatic media.

Vision-based localisation systems. From relative observations of the seafloor, a vision system can estimate the vehicle's motion. While moving, the robot can build a map of the seafloor and uses it to provide continuous estimates of the vehicle's location. The construction of visual maps of the ocean floor can be used in motion estimation and localisation of an underwater vehicle. These visual maps are known as "mosaics". As well as for localisation of a submersible vehicle, underwater mosaics can be used in undersea exploration like sea-floor mapping, the study of oceanographic and geothermal events, geological sampling, the inspection of submersed structures such as pipes, dams and harbours, in ocean mining and oil industry or military purposes. Figure 4.1 synthesises the main systems and sensors for localisation of un underwater robot.

4.1.2 Underwater platform

URIS (Underwater Robot Intelligent System) (see figure 4.2) has been designed and build up at the Computer Vision and Robotic Group of the University of Girona. The robot is a small-size, low-cost Autonomous Underwater Vehicle (AUV) used as a research prototype. It was build with the aim to perform missions either in a controlled environment such as water tanks, or in natural environments such as lakes or sea. The vehicle can be powered by an external source using an umbilical cable for long term experiments but it can also carry its own batteries which provides more autonomy to the robot. The hull has been design as a sphere and it offers equal hydrodynamic coefficient in any directions. It is build up by two hemispheres joined by nuts and bolts. The vehicle is passively stable in roll and pitch. Therefore it has four Degrees Of Freedom (DOF): *x, y, z,* and *Yaw*. The submersible incorporates

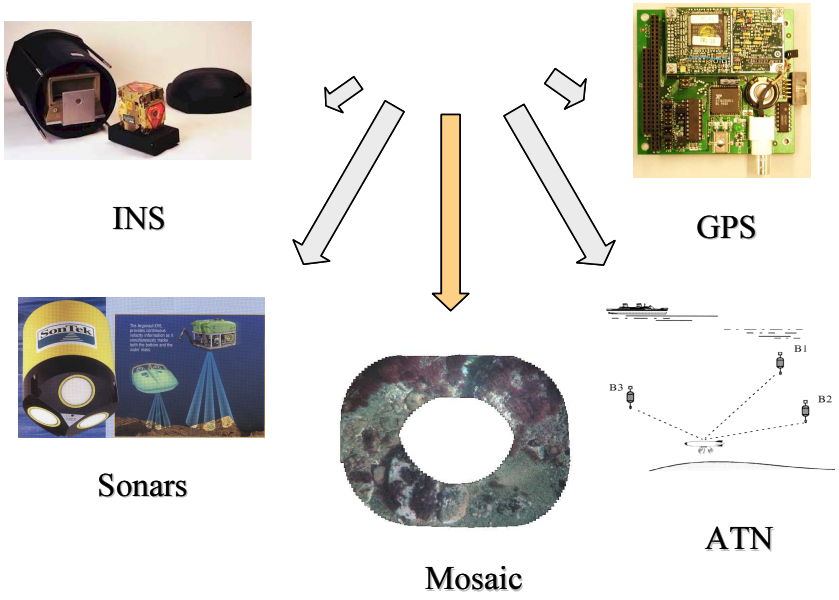
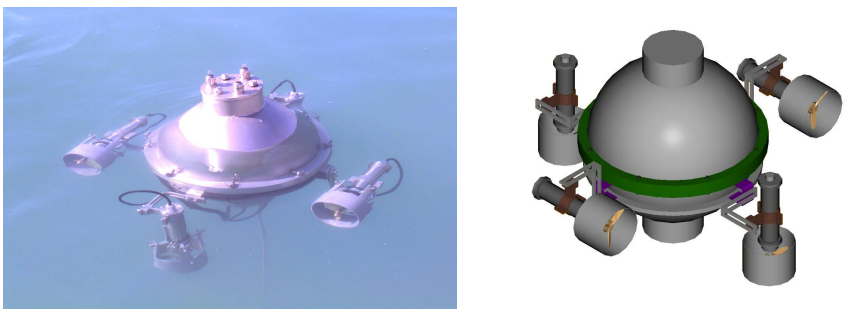


Figure 4.1: Systems and sensors for localisation of an underwater vehicle.

a magnetic compass, pressure sensor, water speed sensors, water leakage sensors and computer vision systems. The control system runs on a Pentium processor from an embedded PC104 plus architecture. A 80C552 microcontroller-based card is used to reduce the computing load of the main processor and is in charge of handling the peripherals. Table 4.1 summarises the main characteristics of this robot. URIS underwater platform, is a low-cost, small-scale robot. This means that making some of the available localisation systems are inadequate for our vehicle. Small accelerations are present in the motion of the AUV and these require large,



(a) (b)

Figure 4.2: URIS underwater vehicle: a) Photograph of the URIS robot in natural environment; b) URIS robot, synthetic image.

Type	Autonomous Underwater Vehicle (AUV)
Propulsion	4 thrusters
D.O.F.	4 : x, y, z, Yaw
Energy	4 packages NiCd batteries ($50W \times 12V$)
Umbilical cable	Functions: power supply, differential video and Ethernet
Computing System	80C552 μcontroller + Pentium processor
Max. depth	30 metres
Sensors	Magnetic Compass Pressure sensor Vision System Speed sensors Differential GPS Water and battery charge detection

Table 4.1: Characteristics of URIS underwater vehicle.

expensive inertial systems. GPS systems do not work underwater, since radio waves are attenuated in such environments. In the case of acoustic transponder networks the system is also very expensive. For every mission a complex system consisting of a boat and the acoustic transponders network must be deployed than recovered after the mission is completed. DVL sensors are also too large to be integrated in the URIS architecture. In addition to the sensors presented above, compasses and inclinometers can also be used to measure the orientation of the robot with respect to Earth's magnetic field and the deviation from the gravity frame, respectively.

4.2 Overview of the mosaicking algorithm

A sequence of images acquired by the camera mounted on the robot can be used to construct a map of the zone surveyed by the submersible. This map is a composite image constructed by aligning a set of smaller consecutive images. In most cases the process involves recovering the motion of the vehicle by mean of gray level correlation [32] or using optical flow [96]. Although these techniques provide good results in standard images [27], they may lead to detection of incorrect correspondences in underwater sequences.

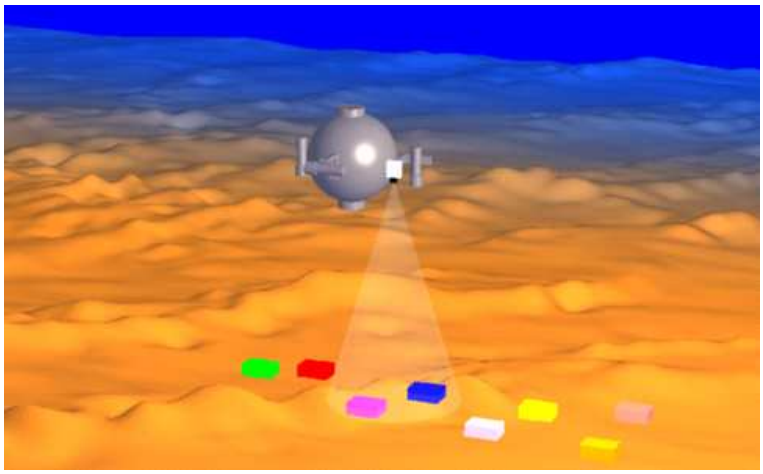


Figure 4.3: Synthetic image of underwater robot navigating alongside seafloor.

4.2.1 Properties of underwater images

The special properties of the medium makes underwater images difficult to process [51]: the elements of the image become blurred and some regions of interest present high clutter and lack of distinct features. Although most of the techniques neglect the use of textural information, considering only image intensity, texture provides a rich source of information to solve image alignment [24]. In the mosaicking algorithm, every image of the sequence is registered to a reference image. Therefore, when an inaccuracy is introduced in the transformation between two images, this error affects not only the current registration, but all the following ones.

One solution can be to install a down-looking camera on the robot's hull. When the robot perform a mission along the sea-floor, the relative motion of the vehicle can be recovered from the two consecutive image. Furthermore, when a sequence of image is acquired by the camera, a map of the sea-bed can be build using mosaicking techniques. In order this information to be used by the robot's control system, some factors such as the distance from the bottom of the sea, the robot's velocity and the camera angle must be considered.

Estimation of the motion between the current frame and the mosaic image is necessary in order to extract the information which to help us to align the frame to the whole mosaic. Motion is considered as a main operation in many computer vision application. A great research effort is therefore dedicated in this field. When dealing with underwater images transmission proprieties of the medium makes the extrac-

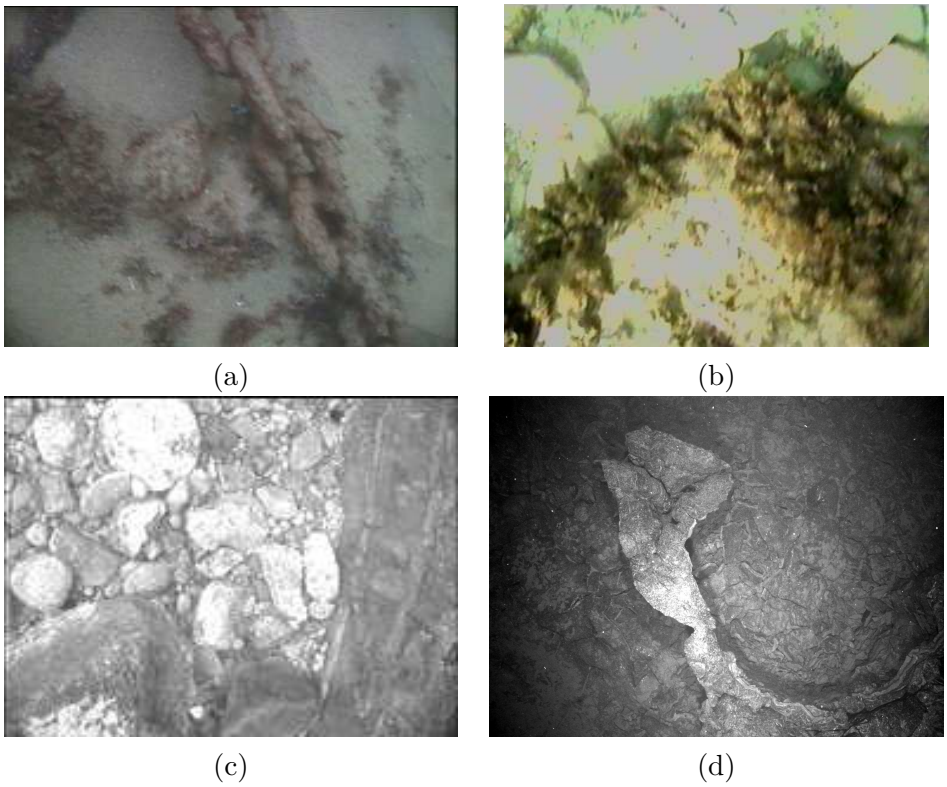


Figure 4.4: Samples of underwater images: (a) scattering produces blurring in the image; (b)strong shading effects produced by the waves; (c) blurring; (d)non-uniform illumination .

tion of the information even more difficult [23]. Light incidents in an underwater environment basically produces two processes: absorption and scattering [24].

Absorption is produced due to the fact that the light energy is transformed in a different form (e.g. heat) and it disappears from the image-forming process. Artificial light is therefore needed to explore this environment. On the other hand, artificial light also introduces many problems such as *non-uniform illumination*. This can play an important role in motion estimation algorithms. Normally the source of light is connected with the camera so that when the camera is moving the light source does so. One feature extracted from the image will appear differently illuminated in the next frame and this can affect the matching process. *Scattering* is another process present in an aquatic environment. It is produced by the changing of direction of individual photons which causes blurring of the elements of the image. This phenomena can affect the feature extraction process because of the lack of well-defined contours.

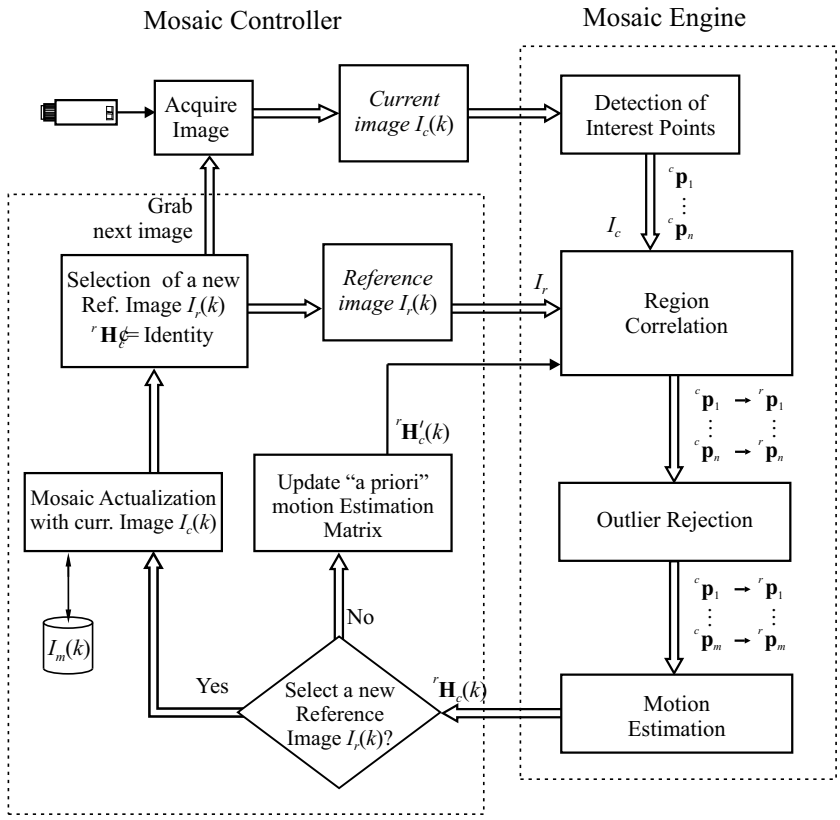


Figure 4.5: Mosaicking algorithm: mosaic controller and mosaic engine.

4.2.2 Underwater mosaicking system

An underwater mosaicking system has been proposed as a means for localisation of the URIS submersible while performing a mission [26]. The proposed mosaicking system is divided into two main parts, namely the *mosaic controller* and the *mosaic engine*. The mosaic controller keeps the state of the mosaicking system and takes decisions according to this state. It is in charge of the mosaic data structure, i.e., updating the mosaic image I_m according to the estimated motion. Meanwhile, the motion is estimated by the *mosaic engine*. It considers the current image I_c acquired by the camera and a reference image I_r and computes a planar "homography" which describes the motion between the two. The homography is a matrix which relates the 2D coordinates of any point from the current image I_c with the coordinates of the same point in the reference image I_r. Selection of the reference image is performed by the mosaic controller. Figure 4.5 shows the relationship between the two modules.

Mosaic controller

The mosaic controller module aims to analyse how the vehicle is moving and generates the pertinent commands to control the mosaic engine. The mosaic controller provides the engine module with the reference image which will be used to estimate the motion of the vehicle. The reference image I_r can be the previous image or an image extracted from the mosaic image I_m by the controller. Every iteration of the algorithm starts when the current image I_c is acquired.

The current image and the reference image at time instant k are denoted $I_c(k)$ and $I_r(k)$, respectively. Let us consider a 3×3 matrix ${}^{r}H_c(k)$ as the homography which transforms the coordinates of a point in image $I_c(k)$ into its corresponding coordinates in the reference image $I_r(k)$. The motion estimated at the previous time instant, ${}^{r}H_c(k-1)$, is assumed to be valid as an "a priori" motion estimation for instant k, since motion between two consecutive images is quite small due to the high frame-rate of the sequence. Then, images $I_r(k)$ and $I_c(k)$, together with "a priori" motion estimation matrix ${}^{r}H_c(k-1)$, are passed to the mosaic engine and is told to execute. The output of the mosaic engine is the homography matrix ${}^{r}H_c(k)$, which estimates the motion between $I_c(k)$ and $I_r(k)$ at time instant k.

Once the engine has finished its execution, the controller decides whether or not I_m should be updated . The controller uses equation (4.1) to update the mosaic image $I_m(k)$ with the current image $I_c(k)$. I_m is only updated in those areas which have not been updated before by the previous images. Thus, the first available information for every pixel is used to actualise the mosaic image. This strategy of using the less recent information to construct the mosaic is known in the literature as "use first" [2].

$$ {}^{m}H_c(k) = {}^{m}H_r(k) \cdot {}^{m}H_c(k) \tag{4.1} $$

The next step consists of deciding whether a new reference image I_r has to be selected for the next iteration. The controller uses matrix ${}^{r}H_r(k)$ to check if the overlapping between the reference image $I_r(k)$ and current image $I_c(k)$ is below a given threshold (e.g. 40% of the size of the image). In this case, it has to select a new reference image $I_r(k+1)$ for the next iteration of the algorithm. The new reference image will be extracted from the mosaic image $I_m(k)$ at the same position and orientation as that of the last image added to the mosaic. Following this methodology, drift in the estimation of the trajectory of the vehicle increases more slowly than

by registering every pair of consecutive images. On the other hand, if the overlap between images $I_c(k)$ and $I_r(k)$ is greater than the threshold, the reference image will not change, i.e. $I_r(k+1) = I_r(k)$.

Mosaic engine

The mosaic engine begins its execution by detecting interest points in image I_c. The goal of the interest point detector is to find scene features which can be reliably detected and matched with points in the reference image. These features should be stable when lighting conditions of the scene change. Corner detectors proposed by Harris [34] and Tomasi-Kanade [89] were tested for underwater images and both will be detailed in the following section. Once the relevant features of image I_c have been detected, the next step consists of finding their correspondences in the reference image I_r. Before searching for correspondences, both images are smoothed with a 3×3 Gaussian mask. Given an interest point ${}^{c}p$ in image I_c, instead of considering the point as an individual feature, we select an $n \times n$ region $R({}^{c}p)$ centred at this point. The system then aims to find a point ${}^{r}p$ in the reference image I_r, surrounded by an $n \times n$ area which presents a high degree of similarity to ${}^{c}p$. This "similarity" is computed as a correlation function (4.2):

$$corr\{R({}^{c}p), R({}^{r}p)\} = \frac{cov\{R({}^{c}p), R({}^{r}p)\}}{\sigma\{R({}^{c}p)\} \cdot \sigma\{R({}^{r}p)\}} \tag{4.2}$$

From equation (4.2) we can observe that the correlation between two points is defined as the covariance between the grey levels of region $R({}^{c}p)$ in the current image and region $R({}^{r}p)$ defined in I_r, normalised by the product of the standard deviation of these regions. Equation (4.2) is computed for all points of the reference image which fall inside a small *search window*. The search window is centered either in the same coordinate of the interest point from current image or, in a point ${}^{r}\widetilde{c}$, as shown in the equation (4.3). The point ${}^{r}\widetilde{c}$ can take into account the estimation of the motion at the previous stage and this is needed when rotation or scaling motions appear in the sequence of images.

$${}^{r}\widetilde{c} = {}^{r}H_c(k-1) \cdot {}^{c}\widetilde{m} \tag{4.3}$$

being ${}^{r}\widetilde{c}$ the projection of interest point ${}^{c}\widetilde{m}$ into the reference image. The coordinates provided by ${}^{r}\widetilde{c}$ are uniquely used to open the search window for the matching process.

Equation (4.2) is normalised by subtracting the mean and dividing by a factor which takes into account the dispersion of the gray levels in the regions under consideration. For this reason, this measurement of correlation is very suitable for underwater imaging, where lighting inhomogeneities are frequent.

According to equation (4.2), given an interest point ${}^{c}p$ in the current image I_c, its correspondence ${}^{r}p$ in I_r should be the point which has obtained the highest correlation score. Those pairs (point-matching) which have obtained a correlation score lower than a given threshold are deleted. However, experimental work developed in our group with underwater images has proved that in some cases the true correspondence is not the one with the highest correlation score [24]. Although these techniques provide good results in standard images [27], they may lead to the detection of incorrect correspondences in underwater sequences.

In order to characterise incorrect correspondences (known as "outliers") through textural analysis, the textural properties of the neighbourhood of both the interest point ${}^{c}p$ and its estimated correspondence ${}^{r}p$ are computed. In this way, the regions $R({}^{c}p)$ and $R({}^{r}p)$ are characterised by two feature vectors (${}^{c}v$ and ${}^{r}v$), which encode their textural properties. Some of the Energy Filters defined by Laws (e.g. L5S5, E3E3, etc.) are used to perform the textural analysis. This textural characterisation may consist, for instance, of measuring the texture at neighbouring locations. If the similarity error between both vectors is smaller than a selected threshold, the pair of point-matching is considered to be an outlier. In fact, this approach is based on the assumption that interest points and their correspondences are located at the border between at least two regions with different textural properties. It is a reasonable assumption since interest points are detected by finding areas of high variation of the image gradient through a corner detector, i.e. located in the border of different image textures.

After a pair of correctly matched points is obtained, the motion estimation ${}^{r}H_c(k)$ between current and reference images can be computed from the remaining pairs of

points applying equation (4.4).

$$ {}^{r}\widetilde{\mathbf{p}} = {}^{r}\mathbf{H}_{c} \cdot {}^{c}\widetilde{\mathbf{p}} \quad \text{or} \quad \begin{bmatrix} \lambda \cdot {}^{r}x \\ \lambda \cdot {}^{r}y \\ \lambda \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^{c}x \\ {}^{c}y \\ 1 \end{bmatrix} \tag{4.4} $$

where λ is an arbitrary non-zero constant. Therefore, solving the homography of (4.4) involves the estimation of eight unknowns. By using inhomogeneous coordinates instead of the homogeneous coordinates of the points, and operating the terms, the projective transformation of (4.4) can be written as:

$$ \begin{bmatrix} {}^{c}x_1 & {}^{c}y_1 & 1 & 0 & 0 & 0 & -{}^{r}x_1 \cdot {}^{c}x_1 & -{}^{r}x_1 \cdot {}^{c}y_1 \\ 0 & 0 & 0 & {}^{c}x_1 & {}^{c}y_1 & 1 & -{}^{r}y_1 \cdot {}^{c}x_1 & -{}^{r}y_1 \cdot {}^{c}y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ {}^{c}x_n & {}^{c}y_n & 1 & 0 & 0 & 0 & -{}^{r}x_n \cdot {}^{c}x_n & -{}^{r}x_n \cdot {}^{c}y_n \\ 0 & 0 & 1 & {}^{c}x_n & {}^{c}y_n & 1 & -{}^{r}y_n \cdot {}^{c}x_n & -{}^{r}y_n \cdot {}^{c}y_n \end{bmatrix} \cdot \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} {}^{r}x_1 \\ {}^{r}y_1 \\ \vdots \\ {}^{r}x_n \\ {}^{r}y_n \end{bmatrix} \tag{4.5} $$

When the engine completes its execution, it gives the control back to the mosaic controller.

This section has given an overview of the mosaicking algorithm used in the localisation system of an underwater robot. The algorithm has two main parts: the mosaic controller and the mosaic engine. The functionality of both has been explained. Sections below will focus on the mosaic engine since the work carried out in this thesis revolves around this subject. Several feature detection methods are detailed in section 4.3.1. Section 4.3.2 of this chapter shows the efficiency of normalised correlation in underwater imaging. In addition, a method for outliers rejection through textural characterisation is proposed in section 4.4 of this chapter.

4.3 Solving the correspondence problem

In order to build up the mosaic image, the apparent motion of the camera mounted on the underwater vehicle must be estimated. This motion can be computed in image coordinates. Many motion estimation methods describe the differences between successive frames of an image sequence. The literature suggests that motion estimation

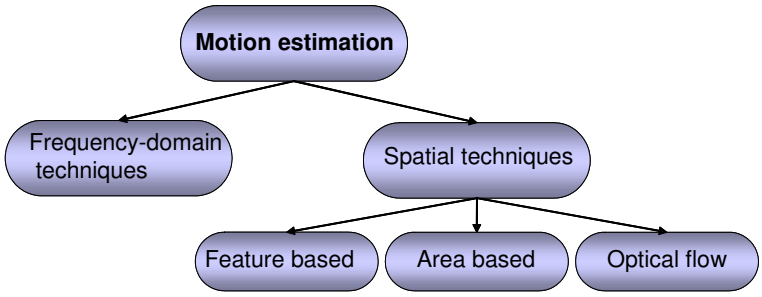


Figure 4.6: A classification of methods for motion estimation.

methods can be divided between spatial-domain motion estimation and frequency-domain motion estimation. Concerning *frequency-domain techniques*, where the relative motion between two images is determined by their Fourier spectrum, very few examples of applying this technique to construct a mosaic exist [78]. Olmos et al. asserted that spatial methods provide better results that frequency-based methods in underwater imaging [68]. *Spatial techniques* are based on a natural way to describe a motion. Algorithms in spatial techniques can be classified as *feature-based*, *area-based* algorithms or *optical flow*. Features can be extracted from an image and used to register two images. The algorithm uses some type of correlation measure to search for a corresponding match in a previous image. These scene features can be reliably found when for example the camera moves from one location to another or lighting conditions change. In area-based matching methods the image is divided in blocks. The algorithm uses the correlation measure to search for a block within an image which most closely matches a particular block in a previous image. This method yields dense depth maps, but fail within occluded areas and/or poorly textured regions [70]. Optical flow computes the apparent motion of pixels from an image taking into consideration image gradient. An important assumption in optical-flow methods is the uniform illumination. But non-uniform illumination and poor gradient are two of the major problems in underwater images. Additional image processing such as local brightness transformation [96] must be performed to counteract this inconvenience in the case of underwater images. These methods are represented in figure 4.6.

4.3.1 Feature detection

Feature detection is a fundamental problem in computer vision. Recovering the displacement of a feature in a sequence of images gives us information about the structure of the environment and the motion of the viewer. Typical features to be matched are interest points [80], straight line segments [13] or, less frequently, image contours [79]. The selection of features may depend on the application, although points are commonly used because they can be easily extracted and are quite robust to noise [34]. The scattering phenomenon, present in an underwater environment, makes the extraction of features such as line segments or contours difficult. Points are reliable features when lightning condition are changing and can be easily detected in underwater images.

Point feature extraction is usually called *corner detection* or *interest point detection*. Many algorithms have been developed in this field. A corner detector algorithms consists of computing the image gradient components: I_x and I_y by convolving the current image with the Prewitt masks [72] (see equation (4.6)).

$$H = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} V = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \tag{4.6}$$

These components are combined resulting in: $J_x = I_x I_x$; $J_y = I_y I_y$ and $J_{xy} = I_{xy} I_{xy}$ and then smoothed with a 3×3 Gaussian mask. In case of the Harris-Stephens corner detector the *"cornerness"* value c is computed according to the equation (4.7):

$$c = \frac{J_x + J_y}{J_x J_y - J_{xy}^2} \tag{4.7}$$

Small values of parameter c imply high image gradient. Then, for the neighbourhood under consideration, any values which are not the minimum of the neighbourhood are suppressed. This leads to a sparse map of interest points for each image. Finally, since c provides a measure of the quality of the corner, the corners are sorted in descending order and best N points are chosen. When considering hardware implementation of this method two problems can arise: the division has a high computational cost and floating-point numbers need to be manipulated. For this reason other strategies were studied. Benedetti et al. [4] proposed a modified

Tomasi-Kanade [89] algorithm which avoids this problem, providing similar results. The method is robust to noise and distortion while characterising each possible corner by an N_s size patch where N_s is small (i.e. 9). The algorithm starts from the assumption that all points in the patch are moving with the same velocity, which is reasonable for small inter-frame displacements. In this algorithm, matrix G of partial derivatives is computed as in equation (4.8):

$$G = \begin{pmatrix} \sum_{k=1}^{N_s} (I_x^k)^2 & \sum_{k=1}^{N_s} (I_x^k I_y^k)^2 \\ \sum_{k=1}^{N_s} (I_x^k I_y^k)^2 & \sum_{k=1}^{N_s} (I_y^k)^2 \end{pmatrix} = \begin{pmatrix} a & b \\ b & c \end{pmatrix} \tag{4.8}$$

The algorithm, first calculates $a(i,j)$, $b(i,j)$, $c(i,j)$; then

$$P_{\lambda_t}(i,j) = (a - \lambda_t)(c - \lambda_t) - b^2 \tag{4.9}$$

is found and every pixel having:

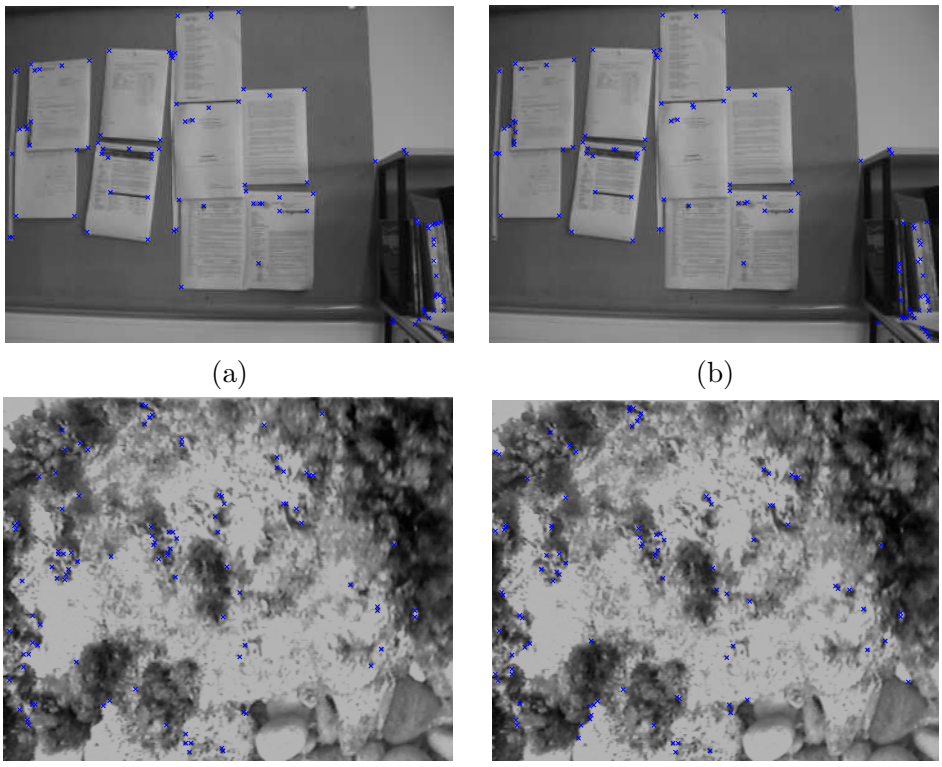
$$P_{\lambda_t}(i,j) > 0 \quad and \quad a(i,j) > \lambda_t \tag{4.10}$$

is retained, where λ_t imposed lower bound for the solutions of the equation (4.9). In the last step the algorithm discards any pixel that is not a local maximum of $P_{\lambda_t}(i,j)$. Similar to previous method, the best N interest points are selected, but considering highest values for $P_{\lambda_t}(i,j)$. Using this approach the complexity is considerably reduced and no floating point operation is required. Figure 4.7 compares the presented feature detection methods applied to different image samples either indoor scenes or underwater images. It can be observed that the results are quite similar in both cases.

4.3.2 Matching algorithm

Detecting correct correspondences in a pair of images taken at two consecutive time instants is an important issue in computer vision. Often this means detecting features in one image and matching them in the second. However, the matching of these features in the second image is normally a complex task. Quite often, local gray level correlation is applied to detect matchings in the pair of images.

Several measure functions for similarities have been proposed in the past. Some



(a) (b)

(c) (d)

Figure 4.7: Comparison between Harris-Stephens and Modified Tomasi-Kanade corner detectors. (a)(c) Harris-Stephens; (b)(d) Modified Tomasi-Kanade

authors compared and characterised them, while the selection of this function influence the performance of the motion estimation [27]. The matching criteria is a measure of the degree of similarity between the current image and a reference image. The correspondence problem can be influenced by many factors [70] such as: *ambiguity*, when the search requires some physical constraints: search window or epipolar constraints; *occlusion*, when points from one image do not appear in the previous one; *photometric distortion*, when illumination conditions changes and figural distortion due to the fact that perspective images of the same objects appear different taken from different views.

Ambiguity is solved by choosing a proper search area in the reference image. While the aim of this work is to achieve real-time performance, motion between frames is small enough to avoid occlusion. Photometric distortion is one of the most important problems which can affect the correctness of the result. Non-uniform illumination appears when artificial light has to be used in the visualisation process.

A correlation algorithm provides, for each interest point $p_c = ({}^c x, {}^c y)$ of the current image I_c, its corresponding match $p_r = ({}^r x, {}^r y)$ in the reference image I_r. Each interest point is characterised by the intensity of the pixels from an area called the *correlation window*. The algorithm looks for similar pathes in an wider area in the reference image called *search area*.

Several distance-similarity measures such as SAD (Sum of Absolute Differences) (equation (4.11)) and SSD (Sum of Squared Differences) (equation (4.12)) were widely used to solve the correspondence problem.

$$SAD = \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} \mid (I_c({}^c x + i, {}^c y + j) - I_r({}^r x + i, {}^r y + j)) \mid \tag{4.11}$$

$$SSD = \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} (I_c({}^c x + i, {}^c y + j) - (I_r({}^r x + i, {}^r y + j))^2 \tag{4.12}$$

where $\alpha = (n-1)/2$ and $n \times n$ is the size of the correlation window. The SAD or SSD measurements are computed for every candidate in the search area. Minimising these distances we obtain the best candidate provided by the method.

Distance minimisation can be replaced by the maximisation of another correlation measure called cross-correlation (CC). The value of CC is computed as in equation (4.13).

$$CC = \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} (I_c({}^c x + i, {}^c y + j) I_r({}^r x + i, {}^r y + j)) \tag{4.13}$$

The standard CC is too sensitive to noise and is normally replaced by the normalised cross-correlation (NCC) (equation (4.14)) or even mean normalised cross-correlation (MNCC)(equation (4.15)) [27] where the mean value of the intensity of pixels from the correlation window is taken into consideration.

$$NCC = \frac{\sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} (I_c({}^c x+i, {}^c y+j)(I_r({}^r x+i, {}^r y+j))}{\sqrt{\sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} I_c({}^c x+i, {}^c y+j)^2 \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} I_r({}^r x+i, {}^r y+j)^2}} \tag{4.14}$$

$$MNCC = \frac{\sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} (I_c({}^c x+i, {}^c y+j) - \overline{I_c({}^c x, {}^c y)})(I_r({}^r x+i, {}^r y+j) - \overline{I_r({}^r x, {}^r y)})}{\sqrt{\sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} (I_c({}^c x+i, {}^c y+j)^2 - \overline{I_c({}^c x, {}^c y)}) \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} (I_r({}^r x+i, {}^r y+j)^2 - \overline{I_r({}^r x, {}^r y)})}} \tag{4.15}$$

where $\overline{I_c(x_c, y_c)}$ and $\overline{I_r(x_r, y_r)}$ are the average intensity of $I_c(x_c, y_c)$ and $I_r(x_r, y_r)$,

respectively.

From equations (4.14) and (4.15) we can observe that the normalised cross-correlation between two points is defined as the covariance between the grey levels of correlation windows characterising a point in I_c and its candidate from I_r, normalised by the product of the standard deviation of these regions. The variance is written as shown in equation (4.16):

$$\sigma^2(I) = \frac{\sum_{i=-\alpha}^{\alpha}\sum_{j=-\alpha}^{\alpha}(I(x+i,y+j)^2}{(2\alpha+1)^2} - \overline{I(x,y)}^2 \tag{4.16}$$

where:

$$\overline{I(x,y)} = \frac{\sum_{i=-\alpha}^{\alpha}\sum_{j=-\alpha}^{\alpha}(I(x+i,y+j)}{(2\alpha+1)^2} \tag{4.17}$$

The correlation score (4.15) then becomes:

$$C = \frac{\sum_{i=-\alpha}^{\alpha}\sum_{j=-\alpha}^{\alpha}(I_c({}^cx+i,{}^cy+j)-\overline{I_c({}^cx,{}^cy)})(I_r({}^rx+i,{}^ry+j)-\overline{I_r({}^rx,{}^ry)})}{(2\alpha+1)^2\sqrt{\sigma^2(I_c)\cdot\sigma^2(I_r)}} \tag{4.18}$$

Four correlation criteria: SAD, SSD, CC and MNCC were applied to several pair of images selected from 7 sequences of underwater images. Average percentage of good correspondences (called "inliers") corresponding to each method is shown in figure 4.8. From this analysis we can conclude that MNCC provide much better results than the other three measurements in case of the underwater images. A comparative result of applying these measurements to underwater image samples characterised by nonuniform illumination, is shown in figure 4.9.

4.4 Textural characterisation

Although the local grey level correlation technique provides good results in standard images [27], it may lead to the detection of incorrect correspondences in underwater sequences. Automatic detection of unreliable matchings can be achieved by means of outliers rejection techniques, such as LMedS [77] or RANSAC [8]. However, these probabilistic algorithms are based on random sampling and robust regression. Due to their probabilistic nature, they may produce incorrect results, although with a

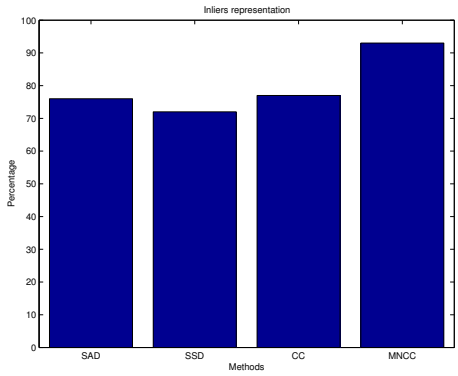
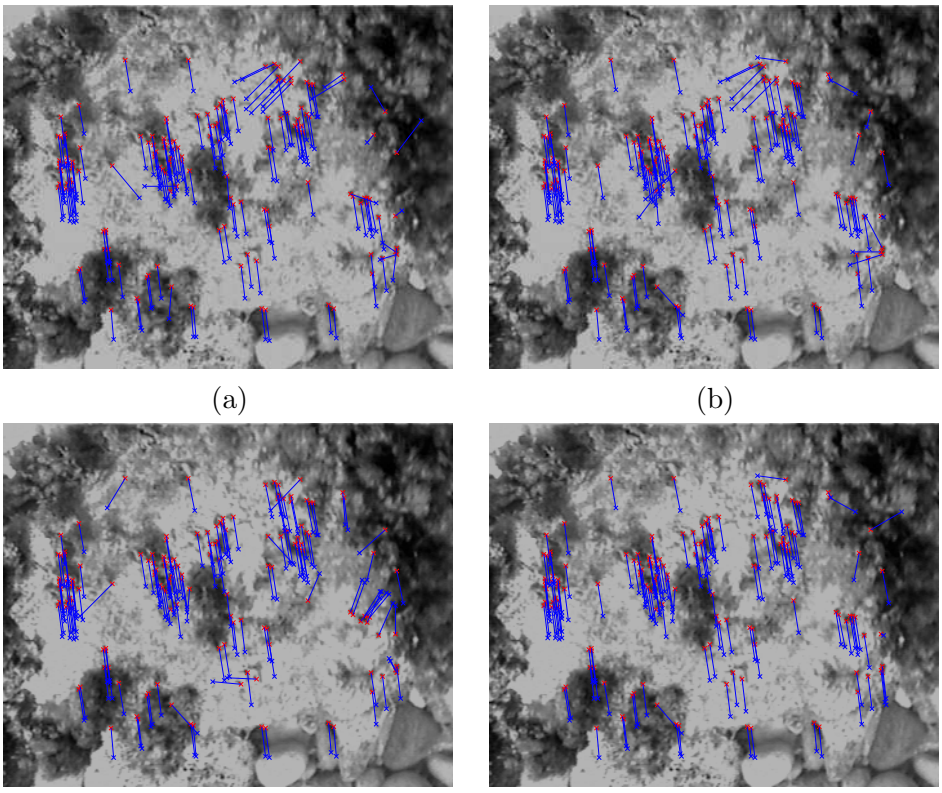


Figure 4.8: Average percentage of inliers obtained by applying four correlation measurements to several pairs of underwater images.

bounded error probability, but their main problem is the high computational cost associated with searching the area of possible estimates generated from the data. In all cases, the aim is to find a set of point pairs which minimise the square sum over the residuals. Given a set of point correspondences, a M-estimators robust technique could also be applied [101]. The M-estimators are based on replacing the residual squares with a weighted function of residuals to make the estimation less sensitive to outliers. Some authors have reported robust behaviour of M-estimators in the presence of bad correspondence localisations but not to false matchings [95]. Moreover, an M-estimator is an iterative algorithm with a considerable computational cost.

The idea of applying texture operators in matching problem was first introduced by Garcia et al. [24]. In this proposal texture operators are used to choose the best correspondence from a set of candidates. In our new approach, the outlier rejection process is independent of the matching problem. It uses intensity-based techniques to detect pairs of point-matching, and texture information to eliminate possible outliers. In this new approach we take advantage of previous results to try out a different method for outlier rejection. These techniques provide the basis for a new methodology to improve point correspondences with a reduced computational burden. Different techniques for solving the matching ambiguities can be found in the literature [100, 8].

The present approach uses texture information to characterise bad correspondences among a set of point pairs in two images. Laws's texture energy filters [59],



(a) (b)

(c) (d)

Figure 4.9: Solving correspondence problem in underwater imaging using: a) SAD criteria; b) SSD criteria; c) CC criteria and d) MNCC criteria applied to the same pair of images.

have been tested and proved to provide far more good results than other texture operators such as co-occurrence matrix [33] and Local Binary Patterns [67]. Energy filters come from a computation of different statistical measures Absolute mean, Standard Deviation, Positive Mean and Negative Mean over a pre-filtered image. Filters are based on 1×3 (see equation 4.19) or 1×5 (see equation 4.20) vectors, namely Level, Edge and Spot.

$$\begin{aligned} L3 &= [\ 1 \quad 2 \quad 1\ ] \\ E3 &= [\ -1 \quad 0 \quad 1\ ] \\ S3 &= [\ -1 \quad 2 \quad -1\ ] \end{aligned} \tag{4.19}$$

and:

$$\begin{array}{rcl} L5 & = & [\quad 1 \quad 4 \quad 6 \quad 4 \quad 1\ ] \\ E5 & = & [\ -1 \ -2 \quad 0 \quad 2 \quad 1\ ] \\ S5 & = & [\ -1 \quad 0 \quad 2 \quad 0 \ -1\ ] \end{array} \tag{4.20}$$

Consider P interest points ${}^k p_1$, and $k = 1..P$, defined in I_1. Texture-based characterisation is performed by considering an $m \times m$ window in their neighbourhood. For every k^{th} point, a texture operator is then computed in its neighbourhood. The same operation is performed for its corresponding matching ${}^k p_2$ in I_2. In this way, two characterisation vectors ${}^k\mathbf{v}_1$ and ${}^k\mathbf{v}_2$ of the k^{th} interest point and its matching are obtained:

$${}^k\mathbf{v}_1 = [{}^k v_{11}, {}^k v_{12}, ..., {}^k v_{1N}] \tag{4.21}$$

$${}^k\mathbf{v}_2 = [{}^k v_{21}, {}^k v_{22}, ..., {}^k v_{2N}] \tag{4.22}$$

where $N = (\frac{m-1}{s} + 1) \times (\frac{m-1}{s} + 1)$ is the size of the characterisation vector, and s is the value of the subsampling of the characterisation window.

The characterisation vector of every point is compared with the characterisation vector of its correspondence point in the second image. Thus, we are searching for similarities in terms of texture. Normalised correlation was used to measure similarity between these vectors, ${}^k\mathbf{v}_1$ and ${}^k\mathbf{v}_2$:

$$C({}^k\mathbf{v}_1, {}^k\mathbf{v}_2) = \frac{\sum_{i=1}^{N} ({}^k v_{1i} - \overline{{}^k\mathbf{v}_1}) \cdot ({}^k v_{2i} - \overline{{}^k\mathbf{v}_2})}{N \cdot \sqrt{\sigma^2({}^k\mathbf{v}_1) \cdot \sigma^2({}^k\mathbf{v}_2)}} \tag{4.23}$$

where $\overline{{}^k\mathbf{v}_1}$ and $\overline{{}^k\mathbf{v}_2}$ are the average value of vectors ${}^k\mathbf{v}_1$, ${}^k\mathbf{v}_2$, respectively, and $\sigma^2(\cdot)$ defines the variance.

This similarity measure can be used to detect bad correspondences. A set of experiments has been carried out in order to select the best texture operators to be applied for outlier rejection in underwater images.

4.4.1 Experimental methodology

Our experimental methodology is in two stages. The first part consists of selecting the best texture operators from a list of 20 energy filters. The second step is to

apply every operator and a combination of them to different underwater sequences and compare the results with robust methods.

Consider a pair of underwater images, a set of point-matchings was obtained through a normalised correlation algorithm, as defined in equation (4.18). As part of our experimental methodology, we marked all the visually incorrect matches, thus providing a list of bad correspondences. Human experts selected the incorrect matchings from the set of pair point-matching. Since the presence of the human factor may introduce errors, five different people performed this operation. When the pairs of points are represented like in Figure 4.12, by drawing the motion vector, incorrect matchings can be easily detected being the vectors whose orientation is different from the dominant one. In some cases, exceptions appear due to underwater conditions such as presence of moving objects (fishes, algae) or a non-uniform see floor (3D aspects), etc. The motion vector of some points can have a different orientation from the dominant one but the correspondence between points may be correct. Every pair of points and their corresponding matchings were textural-characterised. The similarity value between texture characterisation vectors for every point and its matching is analysed. We chose the highest value corresponding to a manually-detected badly correlated point as the *threshold*. This means that, when sorting all the points according to the similarity value, all the outliers lay on the right side of the threshold. On the left side we could guarantee a total absence of bad correspondences, as can be seen in figure 4.11.

The threshold should ensure the rejection of bad correspondences while maintaining a high number of good correspondences. This threshold is used to automatically reject all the outliers at the price of losing a number of correct matchings. This proposed technique looks for the texture characterisation methods which provide a higher separation between *correct* and *false* correspondences. Several experiments were carried out to find the best texture operator.

In the second stage of our experiment the pairs of point-matching from several sequences, the majority differing from those used in the first part of the experiment, were characterised using the best texture operator. In this experiment the false correspondences were unknown. We assumed that the whole sequence could have a fraction ϵ of possible outliers.

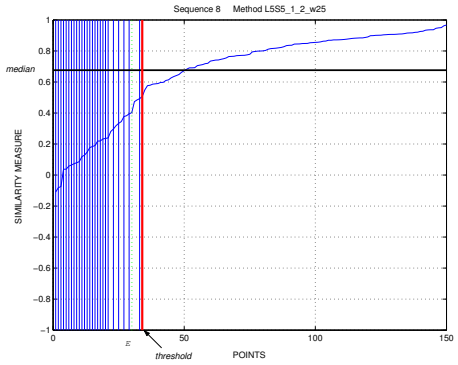


Figure 4.10: Threshold for rejecting the outliers.

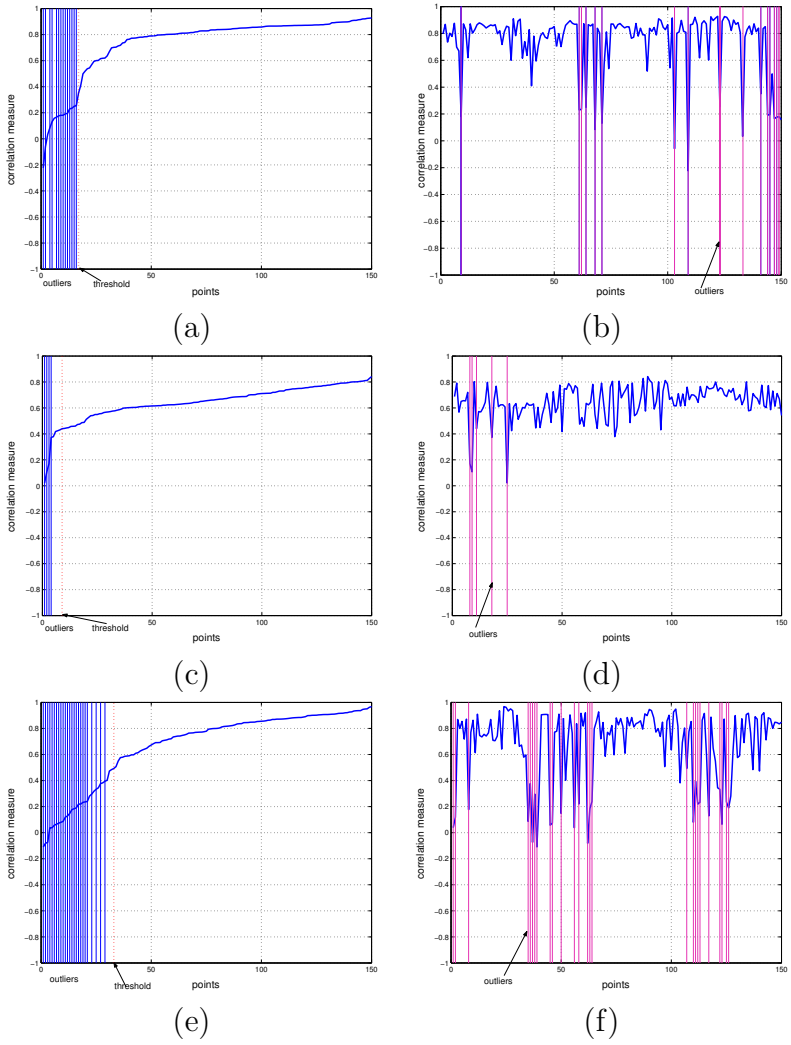


Figure 4.11: Outliers distribution when computing the similarity measure between characterisation vectors for all 150 interest points corresponding to the images from figure 4.12, left side. (a),(c),(e): sorted and (b),(d),(f): unsorted according to their correlation score.

The new threshold was computed like in the figure 4.10 and applied to the sequences under test. This threshold is a function of ϵ and the mean value of the similarity measure between texture characterisation vectors. As we can see in the figure, even if the number of outliers is greater than the estimated ϵ the new threshold eliminate them all. The new threshold was computed by finding a mean value between the average error measurements and the one corresponding to ϵ.

4.4.2 Experimental results

The proposed approach was tested with fifteen underwater image sequences. These images were selected because they provided a good representation of the possible conditions found in underwater environments, such as blurring, lack of well-defined contours, bad visibility, low contrast, scattering effects, non-uniform illumination, lighting artifacts generated by waves, etc. Moreover, different scenarios have been selected: rocky seafloor, sand and algae seafloor, moving fishes, as well as some man-made objects like an old submerged chain, etc.

Two of the sequences used in our experiments are available at the Underwater Vision Lab web site:[1]. The first sequence of images (figure 4.12 (a) and (b)) was acquired using a colour camera at Costa Brava (Spain), near Palamos. The sequence from figure 4.12 (c) and (d)) includes 997 images which were taken by the URIS underwater robot close to the place where the previous sequence was taken. The robot was navigating at a depth of two metres, and its altitude was approximately three metres above the sea floor. The last pair of images presented in this thesis is part of a sequence of images acquired at Platja d'Aro (Costa Brava). These are colour images from a sequence where the robot was following a submerged chain. Some other images used in these experiments were acquired at Nice (France)[2] in September 2000 at a depth of six metres using a PAL camera.

For every image sequence, a set of pairs of point-matching was obtained as described in section 4.3.2, considering a correlation window size of 25×25 and a search window of 61×61, according to the possible displacement between two frames. Figure 4.12 (left) shows three of the tested underwater sequences and the point-matching obtained through region-based correlation.

[1] $http://mosaic.udg.es/pgENG/indexE.html$

[2] The copyright is owned by ISR-IST.

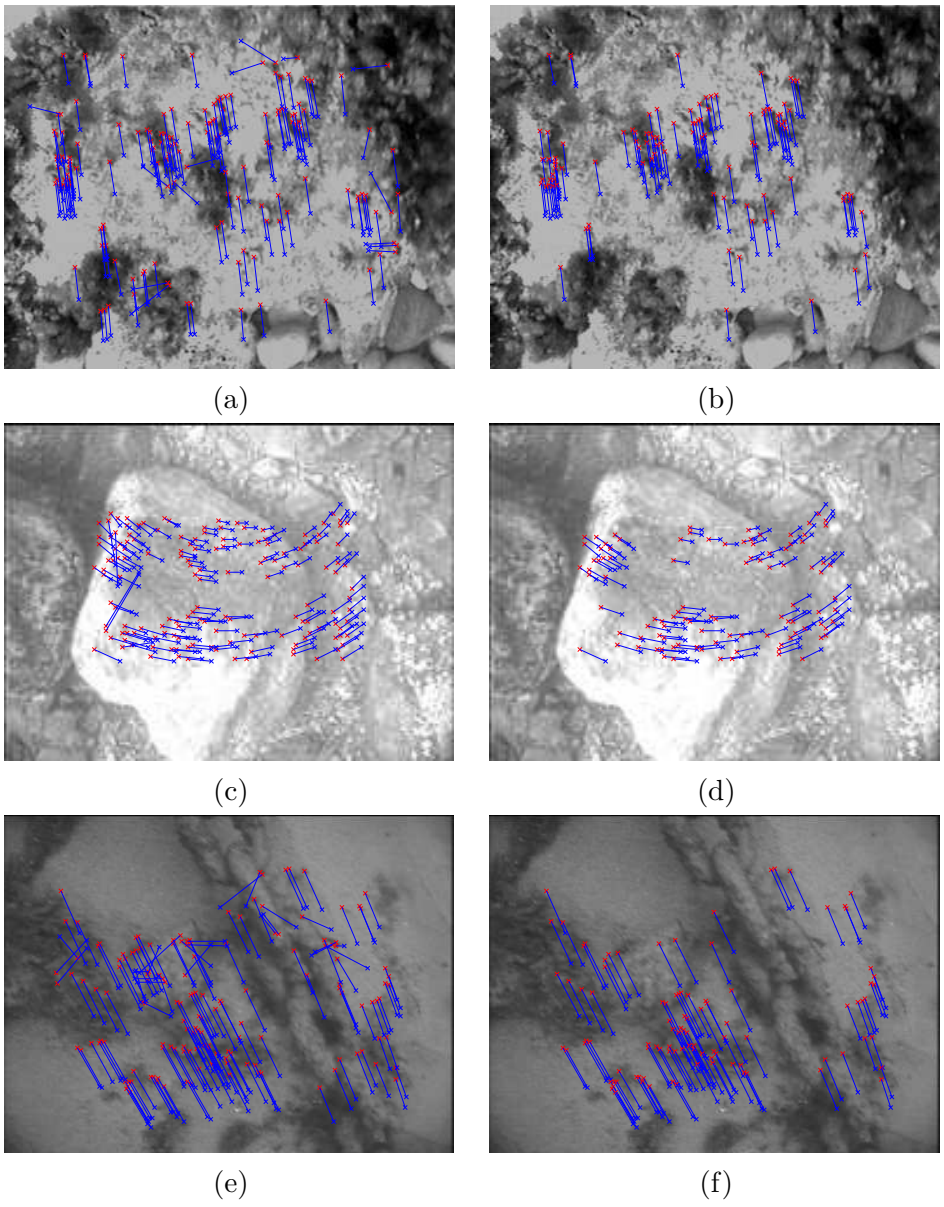


Figure 4.12: Selected image samples from the 15 tested sequences: (a) strong shading effects produced by the waves; (c) rotation; (e) scattering produces blurring in the image. Detection of correspondences in two consecutive images using intensity-based techniques. (b), (d), (f) corresponding pairs of points after outlier rejection using texture characterisation.

Since texture is a property of regions, the textural characterisation of the interest points depends on the size of the neighbourhood under consideration. For every texture operator, four different neighbourhoods of size $m \times m$ were consid-

ered: 11×11, 15×15, 19×19 and 25×25. The neighbourhoods size does not depend on the image resolution. Moreover, within a neighbourhood there is a high degree of redundancy when texture characterisation is applied to every pixel. By choosing different subsampling strategies, our approach will verify whether redundancy introduces robustness in the system. Three significative subsampling modalities were selected: considering every point: $s = 1$, every two points: $s = 2$ or only nine points of the neighbourhood: $N = 9$. Table 4.2 shows the corresponding size of the characterisation vector depending on both the selected neighbourhood and its subsampling strategy. For every pair of images, we tested 20 texture operators, every operator in four different modalities depending on the size of the characterisation window, and every window having three different subsamplings. The following Laws filters, resulting from combinations of basic vectors from equations 4.19 and 4.20, were applied to the image: $L3L3$, $E3E3$, $L5S5$, $E5S5$, $E5L5$. Figure 4.13 shows the resulting images after applying $L5S5$ to underwater image samples. These filters are detailed below (4.24).

$$
\begin{aligned}
L3L3 &= \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \end{bmatrix} \\
E3E3 &= \begin{bmatrix} 1 & 0 & -2 & 0 & 1 \end{bmatrix} \\
L5S5 &= \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \\ -4 & 0 & 8 & 0 & -4 \\ -6 & 0 & 12 & 0 & -6 \\ -4 & 0 & 8 & 0 & -4 \\ -1 & 0 & 2 & 0 & -1 \end{bmatrix} \\
E5S5 &= \begin{bmatrix} -1 & 0 & 2 & 0 & -1 \\ -2 & 0 & 4 & 0 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & -4 & 0 & 2 \\ 1 & 0 & -2 & 0 & 1 \end{bmatrix} \\
E5L5 &= \begin{bmatrix} -1 & -4 & -6 & -4 & -1 \\ -2 & -8 & -12 & -8 & -2 \\ 0 & 0 & 0 & 0 & 0 \\ 2 & 8 & 12 & 8 & 2 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}
\end{aligned}
\tag{4.24}
$$

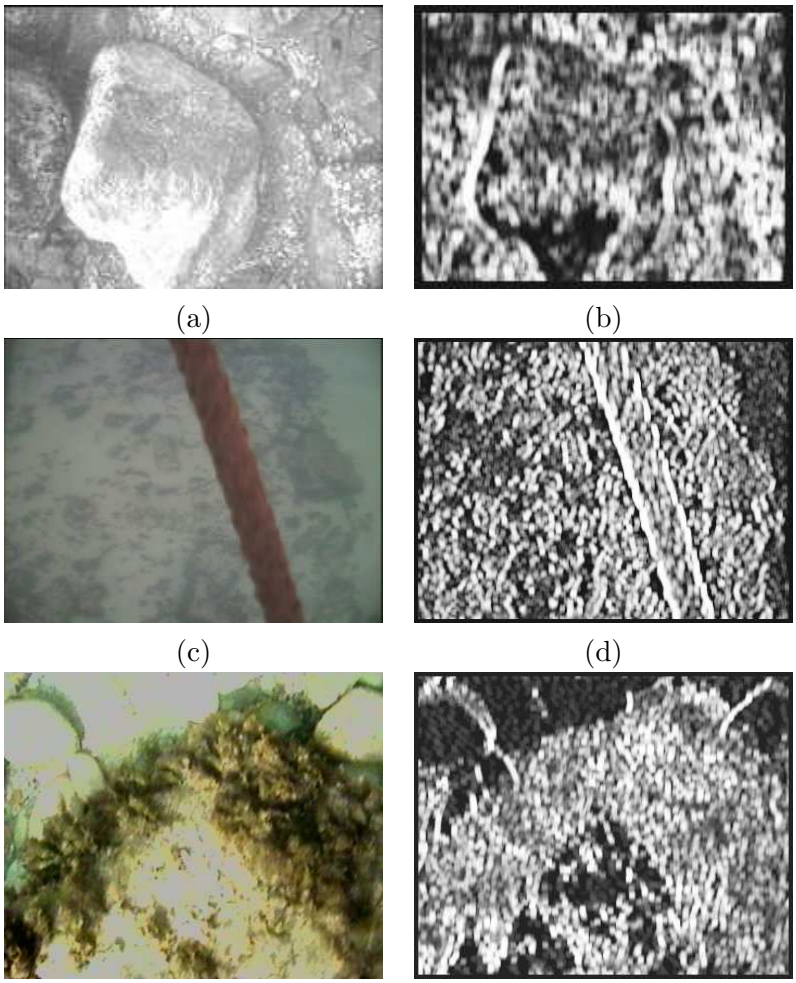


Figure 4.13: L5S5 energy operator applied to samples of underwater images. (a), (c), (e): underwater image samples. (b), (d), (f): images after applying the L5S5 energy filter

Four statistical measures were considered: Absolute Mean, Standard Deviation, Positive Mean and Negative Mean. The goal was to find the most adequate textural operators to solve the correspondence problem. Operators were compared in terms of number of *rejected outliers* and number of *surviving good correspondences*. By observing the distribution of outliers when representing the similarity measure between characterisation vectors, we could select the highest value giving rise to a bad correspondence as a *threshold*. By rejecting all the points with a similarity value lower than this threshold, we could analyse the number of remaining points in order to see the efficiency of this method, see figure 4.11: (a), (c), (e). In order to

evaluate the results, all pairs of point-matching obtained using normalised correlation were visually verified and classified into good and bad correspondences. Even if an error is present in the analysis provided by the human experts, there is only a small probability that this error coincides with the "worst case" which determines the threshold.

Fifteen sequences were tested. For every sequence, 20 texture operators were applied using 4 window sizes and 3 subsamplings. Figure 4.14 shows, for every window size, the average percentage of good correspondences after rejecting the outliers using the texture information.

As we expected, the best percentage of correct correspondences after eliminating the outliers corresponds to the 19×19 and 25×25 neighbourhood size. It can also be observed that a moderate subsampling (considering every two points) provides similar results to processing every point. On the other hand, higher subsampling drastically decreases performance. Six texture operators providing the best performance can be selected according to figure 4.14.

1. mask=L5S5, Absolute Mean

2. mask=E3E3, Absolute Mean

3. mask=E3E3, Positive Mean

4. mask=L5S5, Negative Mean

5. mask=L5S5, Positive Mean

6. mask=E5S5, Negative Mean

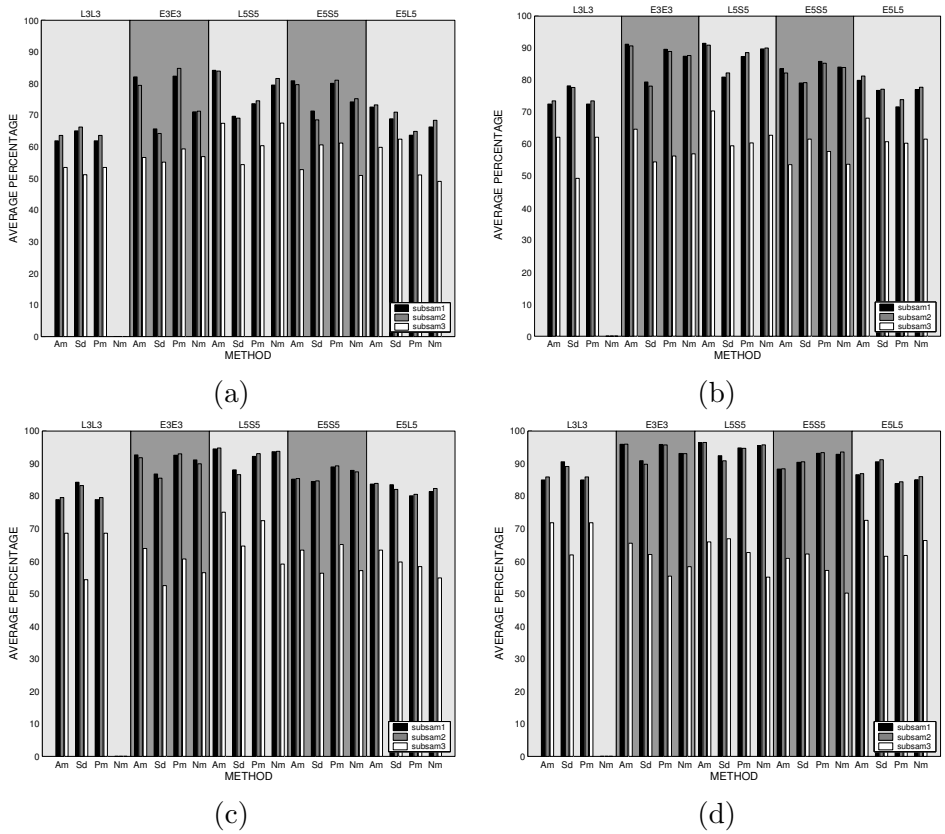


Figure 4.14: Average percentage of remaining correct correspondences after eliminating all the outliers using the proposed texture based method. Neighbourhood size: a) 11×11. b) 15×15. c) 19×19. d) 25×25.

Looking for the possibility of improving the results, we also tested combinations of these six operators and compared them with single operator performance. Different ways of combining texture characterisation were tested: summing the similarity measure obtained using two texture operators or applying their minimum value. Fifteen combinations were tested using a 25×25 window sampled every two pixels. The results are shown in table 4.3: data below the diagonal are the average percentage when the texture operators were combined by summing the similarity values; the diagonal contains the results in the case of applying a single operator and above the diagonal, the minimum similarity value to reject the outliers. It can be observed that better results were obtained following the summing approach. However, the differences between these three approaches are quite small, and so only one textural operator will be considered to reduce computation time.

The second step in our experimental methodology was to apply the best textural

Window size				
Sampl.	11×11	15×15	19×19	25×25
1.	121	225	361	625
2.	36	64	81	169
3.	9	9	9	9

Table 4.2: Size of the characterisation vector according to different neighbourhood sizes and their subsampling.

Op.	$L5S5\_Am$	$E3E3\_Am$	$E3E3\_Pm$	$L5S5\_Nm$	$L5S5\_Pm$	$E5S5\_Nm$
$L5S5\_Am$	**96.0722**	95.3538	96.1968	96.2814	95.2629	93.4897
$E3E3\_Am$	96.0989	**95.5880**	96.3751	96.0911	95.2749	93.2692
$E3E3\_Pm$	96.6912	96.3472	**95.3515**	95.3277	96.0041	94.8135
$L5S5\_Nm$	96.4366	96.5661	95.7628	**94.2817**	96.0194	94.4341
$L5S5\_Pm$	95.8195	95.6658	96.4316	96.1840	**93.4180**	92.9625
$E5S5\_Nm$	95.9136	95.5415	95.9599	96.3436	95.1490	**93.0360**

Table 4.3: Average percentage of correct correspondences after eliminating all the outliers using the six proposed texture operators and their combination. 25×25 characterisation window and every two pixel subsampling were applied. **Below diagonal**-sum of correspondence measures between characterisation vectors; **Above diagonal**-minimum of correspondence measures between characterisation vectors; **Diagonal**- apply single textural operator.

operators for outlier rejection and comparing the results with robust methods. One important issue is to find a threshold to ensure the rejection of all outliers. In this experiment we observed that by sorting the pairs of point-matching according to the value of their correlation measurement between texture characterisation vectors and applying the threshold proposed in subsection 4.4.1 we obtained satisfactory results. This strategy was applied to fifteen other sequences, different from those used in the first experiment. Figure 4.12: (b), (d), (f) shows the results in the case of applying texture based strategy. Due to space limitation only three sequences are illustrated in this thesis. Results from all the sequences are available on the web[1].

Tests on real underwater images show an adequate characterisation of the incorrect correspondences. We selected different sequences of underwater images in order to test how the algorithm can tackle underwater imaging problems. Texture provides a rich source of information for feature characterisation, this being essential when difficulties arise with the use of standard techniques. The robustness of the proposed

[1] $http://eia.udg.es/\sim viorela/image\_texture\_chr.htm$

technique is based on the exploitation of gray level information complemented by texture cues. In terms of computational cost, this new approach outperforms random sampling techniques such as LMedS or RANSAC. On the other hand, due to their probabilistic nature, random sampling methods may produce incorrect results. An important observation is that LMedS or RANSAC methods eliminate all the points with differentf displacement comparing to the majority of points, while texture-based techniques eliminate only bad correspondences. While robust these need a priori estimation of either fundamental matrix, in case of $3D$ scene, or homography in case of planar scene, the presented new texture-based approach does not require any a priori information. When considering hardware implementation of a registration algorithm, this new approach can be helpful due to its ease of parallelization.

4.5 Parameters and assumptions

In a real application, the image signal is acquired by a down-looking camera mounted on a underwater vehicle. From that signal, the computer vision algorithms can extract useful information to be sent to the control unit of the robot. Based on this information from the vision system together with information provided by other sensors, the control unit generates the commands to be sent to the actuators. These are the basic steps the robot's system might perform for an autonomous navigation. As we can see, the process is complex and requires a high amount of operations. Therefore, accelerating some parts of computer vision algorithm by mean of hardware implementation can be a useful technique. As it is shown in figur 4.5, feature detection is the first step of the algorithm. Afterwards, correspondence problem must be solve.

The feature detection algorithm proposed by Benedetti et al. [4] was chosen to be implemented in hardware. In this method, the computation is simplified and avoid floating-point operations. The parameter N_s is set to 9 in the further hardware implementation. Motion estimation algorithms need a minimum of $N = 4$ correlated points to compute the homography matrix. In order to obtain a robust estimation this number should be increased to 100 or 200 points [26]. Table 4.5 shows the parameter's values used in corner detection algorithm.

Modified Tomasi-Kanade Algorithm	
Parameter	*Value/range*
N_s	$9 \dots 25$
λ_t	1000
N	$100 \dots 200$

Table 4.4: Parameter's values for feature detection algorithm.

MNCC criteria	
Parameter	*Range*
correlation window	$7 \times 7 \dots 15 \times 15$
search window	$15 \times 15 \dots 30 \times 30$

Table 4.5: Parameter's values for correlation algorithm.

In section 4.3.2 we showed that the Mean Normalised Cross-Correlation (MNCC) works far better than other correlation criteria in underwater imaging. Key parameters for the good performance of this algorithm are the *correlation window* and *search window* size. Typical values for the correlation window lay in a range between 7×7 and 15×15. The search window size depends on the apparent motion between frames influenced by how fast the vehicle moves. When the sequence of images is processed at frame-rate, small inter-frame displacement appears. For this reason, small search windows should be considered. Taking into account the navigation altitude of the robot, i.e. the distance from the camera to the sea floor, and the vehicle maximum speed a search window of 30×30 in the worst-case scenario. Table 4.5 shows the search window and the correlation window sizes.

Outlier rejection and the mosaic controller are aimed to be implemented on the host computer while more complex operations must be performed. The hardware accelerator might provide pairs of correlated points to the host. The host processes them and takes the pertinent decisions. It actualises the mosaic image when necessary and extracts the reference image to be sent to the hardware accelerator. A threshold is used by the mosaic controller to decide whether the reference image may be actualised or not. This threshold defines the degree of overlapping between the current image and the reference image.

Chapter 5

Parallel Approaches for Motion Estimation Algorithms

5.1 Introduction

Motion estimation requires a huge amount of computation. A lot of research has been carried out to develop efficient architectures to make possible real-time execution and single chip integration of these algorithms. In motion estimation, points from current image and points from reference image need to be correlated. Correlation algorithms have important properties such as regularity and modularity. Therefore, they can be broken down into computational blocks which can be processed in parallel. As we showed in chapter 4 different correlation methods can be applied to find pairs of matched points. The Sum of Absolute Differences (SAD) from equation (5.1), is the most extensively used matching criteria in VLSI implementation due to its simplicity and the fact that it provides good results in motion estimation. In this approach, simple operations like multiplication and accumulation are applied to multiple data. SAD criteria is normally used in motion compensation for many multimedia standards, including MPEG-1 and MPEG-2 and other non-standard video coding in application areas like video conferences, digital television, etc. Block Matching Algorithms (BMA) are used for motion estimation in this type of applications. In these algorithms the current frame is divided into blocks of pixels, and for each block the algorithm searches for similar reference blocks within a search area in the previous frame. When this algorithm considers all the blocks from the

search area for the comparison, it is called Full Search Block Matching Algorithms (FSBMA). The BMAs give non-optimum solutions but are less computationally intensive than FSBMA [3]. While BMAs are suitable for software implementation, FSBMA can be easily mapped in SIMD parallel architectures in order to decrease their computation time. In a full search, the algorithm has a very regular data-flow for the search area. These are often repeatedly used in the computation for different candidates. In FSBMA the motion is estimated using SAD criteria which can be computed as shown in equation (5.1):

$$SAD(u,v) = \sum_{i=1}^{n}\sum_{j=1}^{n} \mid S(i+u, j+v) - R(i,j) \mid \tag{5.1}$$

where n is the block size, $R(i,j)$ are the pixels of the reference block, and $S(i+u, j+v)$ are the pixels of the search area corresponding to the displacement (u,v) with respect to the reference block. The best match for the reference block is associated with the smallest SAD within the search area, as in equation (5.2).

$$m = \min_{(u,v)}(\sum_{i=1}^{n}\sum_{j=1}^{n} \mid S(i+u, j+v) - R(i,j) \mid) \tag{5.2}$$

The cost of any correlation algorithm depends on the block size and the search area. For a FSBMA, in the case of an image size of $(N \times M)$ pixels where p is the maximum displacement of a pixel, the search area becomes $(n+2p) \times (n+2p)$ and the computational complexity is shown in equation 6.8:

$$(\frac{N}{n} \times \frac{M}{n} \times (2p+1)^2 \times n^2) \tag{5.3}$$

Parallel implementation of these algorithms aims to reduce this complexity.

5.2 Overview of parallel implementations for FSBMA

An extensive literature exists about array architectures for Full Search Block Matching Algorithms applied to motion estimation [3, 4, 55, 84]. In general, proposed the VLSI architectures for FSBMA are based on three parts: an array of Processing

Elements (PE), address generators and memories. The array of PEs performs the operation matching (eg. SAD), the address generator generates the addresses for reading the memory according to the data flow required by the array of PEs and the memory stores the reference and the search area.

The computation core is normally based on systolic arrays. A *systolic array* (by analogy with the regular pumping of blood by the heart) is a network of processors which rhythmically compute and pass data through a system. Kung and Leiserson [56] published the first paper describing systolic arrays in 1978. Each processor at each step takes in data from one or more neighbours (e.g. North and West), processes it and, in the next step, outputs the results in the opposite direction (South and East). Modularity, regularity, local interconnection, high degree of pipelining, synchronised processors are important characteristics of systolic arrays. Arrays of processors are connected to a small number of nearest neighbours in a mesh-like topology. These processors perform a sequence of operations on data that flow between them. Generally these operations are the same in each processor. Each processor performs an operation on a data item and then passes it on to its neighbour.

The address generator generates the addresses for the data which will be needed in the next computation step. The data can be read from the external memories and fed to an array of multiple processors. In this case a very large bandwidth is required. Based on the fact that in matching algorithms a large amount of search window data is reused in the computation of the matching criteria for nearest candidates, the number of read/write operations from the memory can be reduced by using internal buffers and/or registers. The current frame can be preloaded in the array architecture in order to reduce the external memory accesses. In this case, $n \times n$ array of Processing Elements (PE) are used. Due to the advances made in technology over the last years, large on-chip memory is now available. This permits storage of high amount of data in a so called internal "cache" memory, either in a line buffer [92] structure or macroblock buffers [3].

Baglietto et. al [3] proposed mesh based processor and a register array which permit the parallelism to be exploited both at the frame and block level reducing the computation complexity. A $(2p + 1)^2$ core of Processing Elements (PEs) is surrounded by a mesh of Memory Elements (ME). Each PE computes the SAD associated with a candidate block while the MEs store the pixels of the search area.

Each ME can exchange data with the four neighbours and the size of ME array depends on the size of the search area. Therefore, this mesh type storage array is suitable for applications where the search area is small.

Vos [92] proposed two possible storage approaches: passive elements and line buffers. Passive elements play the same role as in the memory elements, but the architecture needs less of them. The quadratic array of MEs is replaced by lines of passive elements placed above and below the PE array. Roma et al. [76], making minor changes in the data-flow control, remove the lines of passive elements placed below the array of PEs. This approach is very appropriated if the data is supplied by a memory. On the other hand, a linear buffer is used to delay the incoming pixel. Linear buffers are preferable when the data are coming directly from the camera. This type of storage minimises the on-chip address calculation, while each pixel has to be read only once and delayed for the next time it is used.

Memory bandwidth problems can be solved by carefully scheduling the input data or optimally setting up the on chip memory. These two approaches were adopted with minor changes in many VLSI architectures for FSBMA [92, 75, 97]. Depending on the application and also depending on the resources, the main objectives are to reduce the memory bandwidth requirement [97], control the dataflow through the array of processors [3] and the number of registers [75].

5.2.1 Mapping an algorithm onto an array of processors

In order to be mapped onto an array of processors, an algorithm is broken down into its basic operations and converted into a form where each result is assigned to a single variable. This defines the Single Assignment (SA) principle. The algorithm can be applied over an n-dimensional space and the SA principle can be represented by an n-D dependence graph (DG). In a DG the nodes represent the basic operations and the direction of the edges denotes the operations sequence. The time scheduling and the projection of multiple nodes over one processing element can be represented in a data flow graph (DFG). Kung [57] compared different mapping methodologies for systolic architectures. He defined three stages for deriving a VLSI architecture for a given algorithm:

- Representing the algorithm using a DG which specify the data dependencies

of the algorithm.

- Mapping the DG into a DFG by projection and scheduling.
- Transforming the DFG into a VLSI array which can be: a Single Instruction Multiple Data (SIMD), a Multiple Instruction Multiple Data (MIMD) architecture, a systolic array or a wavefront, the term Kung uses in his classification [57].

Komarek and Pirsch [55] used index projection, time scheduling and graph folding techniques to analyse four types of systolic arrays for FSBMA mapping. Two possible directions of the dependence arcs from DG comes out from the property of associativity of the operations in the algorithm. One dimensional ($1D$) and two dimensional ($2D$) projection were analysed. Some authors refer to linear and quadratic arrays for $1D$ and $2D$ structures, respectively. The $2D$ arrays are obtained by projecting the initial DG once. The $1D$ array of PEs can be seen as the projection of a column from the $2D$ array. The four resulting array structures are denominated as: $AB1$, $AS1$ for $1D$ array and $AB2$, $AS2$ for $2D$ arrays. These architectures exploit concurrency using different structures and numbers of PEs.

5.2.2 Systolic arrays for block matching algorithms

In an exhaustive search the algorithm looks for the best match between all possible candidates inside a search area. For a better understanding of the steps that are performed for mapping the algorithm into a VLSI architecture let us consider a 3×3 reference block like in figure 5.1 and a displacement $p = 2$ inside the searching area. Considering a FSBMA, the algorithm's DG can be represented like in figure 5.2 over a four dimensional index space due to its four indexes: (i, j, u, v). Every **(AD)** node performs an Absolute Difference, the **(A)** nodes accumulate the differences as in equation (5.1) and the **(M)** nodes compare the result in order to decide the best match, (see equation (5.2)). One and two dimensional projections are appropriate for a hardware implementation. In the following sections, four basic structures are analysed considering their processing time, hardware utilisation and memory accesses.

AB1 Projecting on the i-axis results in a one-dimensional DFG and an n-size architecture. Consecutive computation of $(2p + 1)^2$ candidates is completed in $n \times$

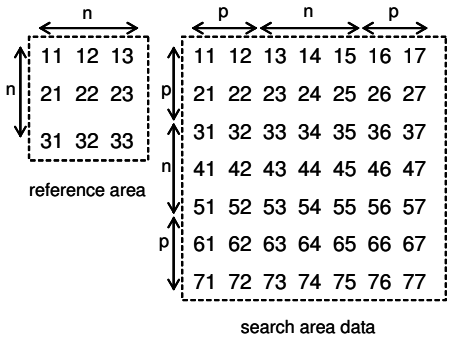


Figure 5.1: Reference Block and Search Window.

$(2p+1)^2$ time cycles. Replacing of one input data column with another introduces additional $n-1$ meaningless time cycles. The number of time cycles can be computed as shown in equation (5.4) and the architecture is illustrated in figure 5.3.

$$t_{AB1} = n \times (2p+1) \times (2p+1+n-1) = n \times (2p+1) \times (2p+n) \tag{5.4}$$

AB2 Projecting on the i, j-plane results into a two-dimensional DFG. The numbers of PE is equal to the size of the reference block: $n \times n$. In this case, the reference block data $R(i,j)$ remains fixed in the PEs and the search area $S(i+u, j+v)$ are fed into a $2D$ array of PEs. See the total time cycles in equation (5.5) and the $n \times n$ array of PEs in figure 5.4.

$$t_{AB2} = (2p+1) \times (2p+n) \tag{5.5}$$

AS1 Projecting on the j-axis results into a simple one-dimensional architecture which require only sequential input of the reference and search frame. As in the first case (AB1) some dummy data must be inserted into the data input stream to ensure regular dataflow. Equation (5.6) shows the time cycles corresponding to the array structure of figure 5.5

$$t_{AS1} = n \times (2p+1) \times (2p+n) \tag{5.6}$$

AS2 Projecting onto the i, u provides a systolic array like in figure 5.6. In this case the structure has $n \times (2p+1)$PEs which in fact is equal with one line of possible matches. A different value of the reference block term corresponds to every column

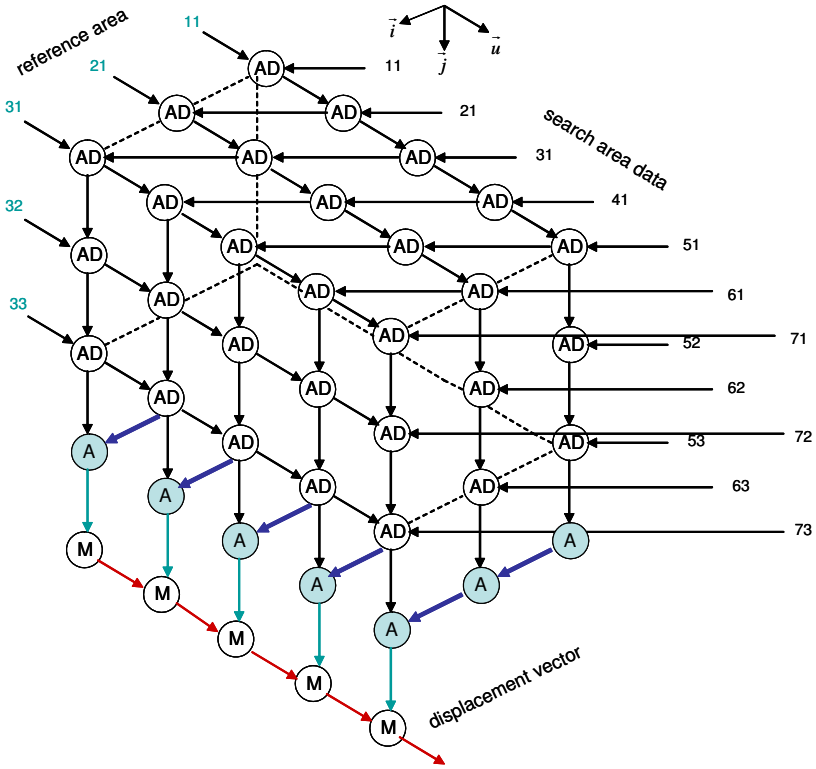


Figure 5.2: Dependence graph for FSBMA.

in this structure. The number of time-cycles needed to perform the whole search corresponds to equation (5.7):

$$t_{AS2} = n \times (2p + 1) \tag{5.7}$$

5.2.3 Proposed architectures

Different proposals for reducing memory access, execution time and increasing hardware efficiency were derived based on the four structures presented above.

Architectures proposed Vos

Vos and Stegherr [92] proposed some efficient architectures for solving the input data bandwidth problems and reducing the control complexity. Their approach included on-chip line buffers and additional registers thoroughly optimising the PE

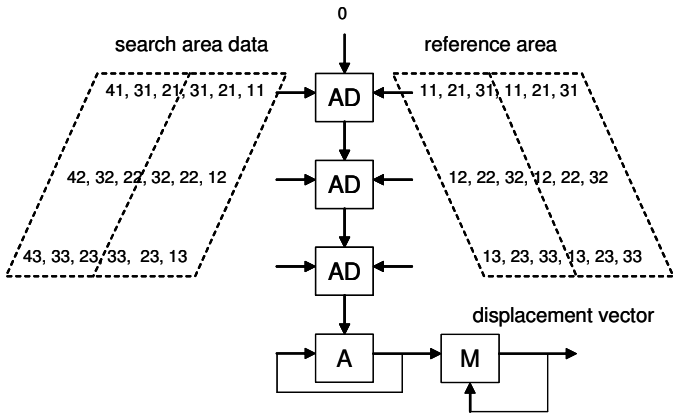


Figure 5.3: AB1 linear structure.

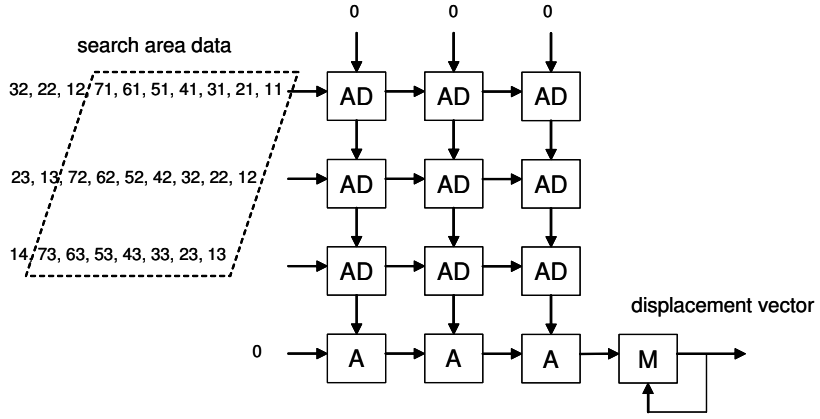


Figure 5.4: AB2 quadratic structure.

structure."Time sharing", in the case of low pixel-rate was also analysed. New classifications were introduced in relation to *Type I* vs. *Type II* architectures and *linear* vs. *block* scan. Type I structure refers to AB2 architecture. In Type II structure the execution order of the loops are inverted, so that the size of the array of PSs depends on the search area $(2p+1)\times(2p+1)$ which differs from the $n\times(2p+1)$ size of the AS2 structure. While in *linear-scan*, the search area data are fed into the PE array line after line as they come from the camera, in *block-scan* pixels within the search area are traversed column by column.

For the AB2 architecture, each PE contains an arithmetic unit and registers for previous and the current frame. Reference frame data is fed into the array of PEs though the *reference data* input and the previous frame through the *search area data1* and *search area data2* inputs as in figure 5.7. Additional $2p$ rows of registers

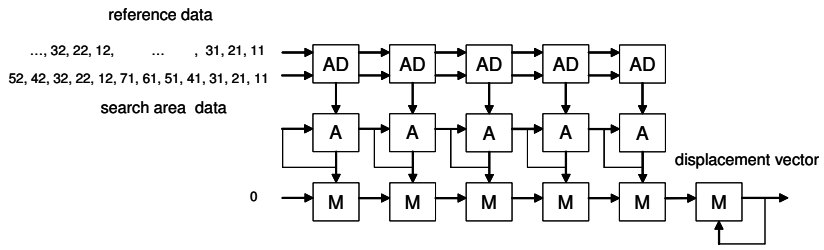


Figure 5.5: AS1 linear structure.

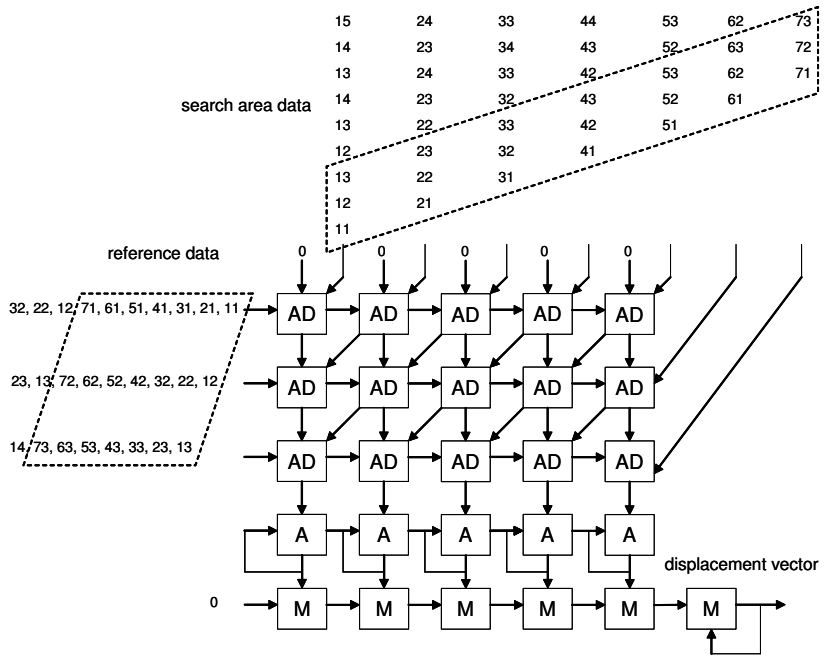


Figure 5.6: AS2 quadratic structure.

were added on the upper and lower sides of the array of PEs. This permits moving the previous frame data upwards, downwards and to the left until the whole search is performed. For a proper data flow through the array structure, "cut sets" are introduced in the structure of each passive or active processing elements. These are registers to delay the data being sent to the next processor.

For the AB1 structure, the distributed registers for data exchange between PEs are replaced by a compact memory block. The advantage of linear array structures is mainly their low hardware occupancy and reduced memory access: only about half of the total number of transistors are needed. One memory block with n rows and $(n + p)$ columns is required for storing the previous frame and two $(n \times n)$ memory blocks are needed for the reference block. Pointer control for read and

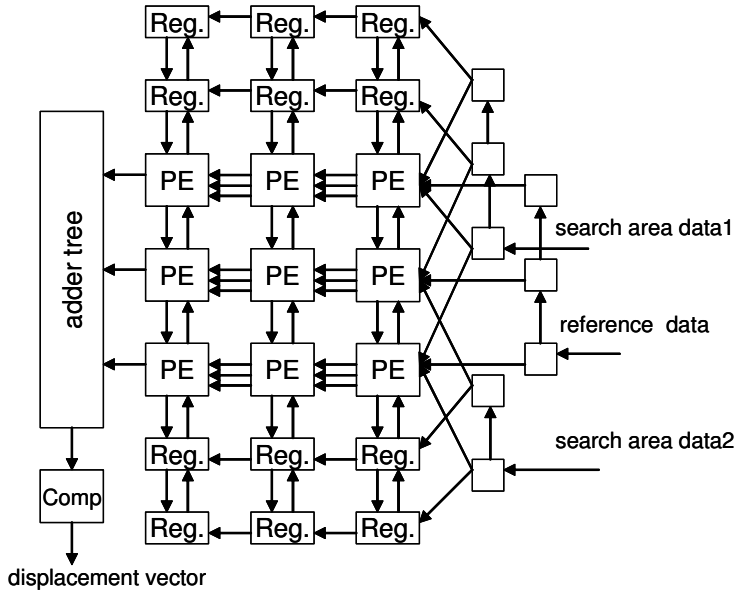


Figure 5.7: Vos's AB2 architecture.

write operations are necessary. The transport and accumulation along the rows is simplified in this case but all of these sums must be further accumulated (Acc) during n cycles as can be seen in figure 5.8.

Hsieh's flexible array

Hsieh [40] proposed an AB2 type architecture which permits a flexible adaptation to the dimensional change of the search area and reduces memory throughput. Each PE stores the reference pixel, and shifts the search area pixel to the right neighbour and the absolute difference result to the PE below. Figure 5.9 illustrates the data flow through the array. One search area pixel datum is input at each instant, into the array, thus shift registers are used to delay the data to feed the other PEs. If the search area changes, the structure simply can be adapted by changing the number of shift registers. Compared to Vos quadratic architecture, it reduces the number of registers from $(4 \times n \times p)$ to $(2p-1) \times (n-1)$.

Roma and Sousa contribution

Roma and Sousa [75, 76] improved the Vos quadratic architecture by reducing the number of additional register rows. They also performed an extensive efficiency and

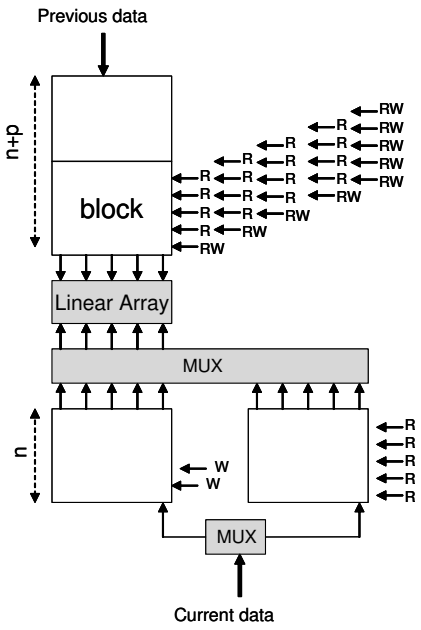


Figure 5.8: Vos's Type 1 architecture. Linear array in block-scan mode (n=5, p=4)

performance analysis of different systolic architecture. They compared eight systolic structures proposed in the literature by analysing the number of PEs and the time cycles.

The main differences between the architecture proposed by Roma and Sousa and the one proposed by Vos can be understood better by picturing the structure of the array of PEs and the rearrangement of the register rows like in figure 5.10. The planar processor that Vos proposed is disposed over a cylindrical structure, doing that,

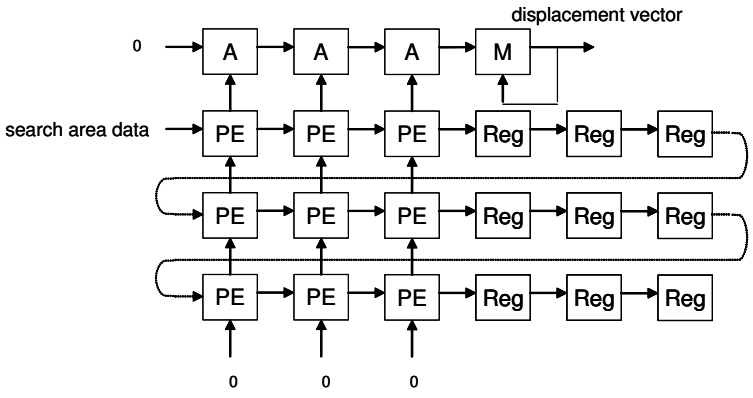


Figure 5.9: Hsieh's architecture.

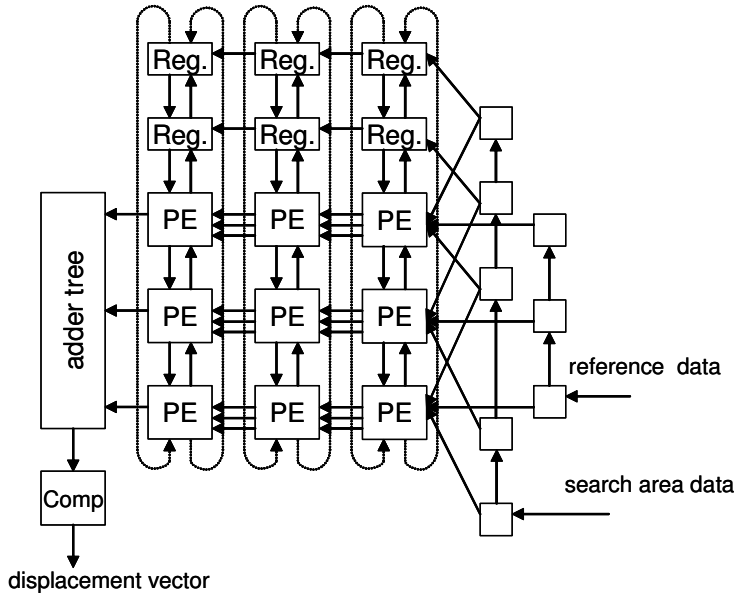


Figure 5.10: Nuno's improved AB2-type architecture.

the pair of passive registers are superimposed, so that one of two can be eliminated. This new proposal improves the search area data transfer and avoids dummy clock cycles between two adjacent rows while reducing the number of registers.

Yang's linear arrays

When mapping an algorithm into an array of processors, the problem is to access multiple data to feed all the PEs at the same time. Yang et al. [84, 98] proposed a solution which consists of local data exchange between PEs. It has two parallel memory accesses (s_1, s_2) for the search area and one for the current block (c), see figure 5.11. Once the data are read from the memory, they are broadcasted to every PE. Buffers are used to delay data and multiplexors to switch between the data. For the architecture to be used more efficiently, the size of the search window must be defined according to the size of the correlation window. Two different architectures were proposed. The difference between them consists in the way the data is broadcasted to every PE. In one of them at each step, an exchange of reference block data between PEs is performed while the search data are fed from external registers. In the second proposed architecture the search data are exchanged between PEs while the current block comes from external registers. While in the first architecture the new result of every accumulation is available at every clock

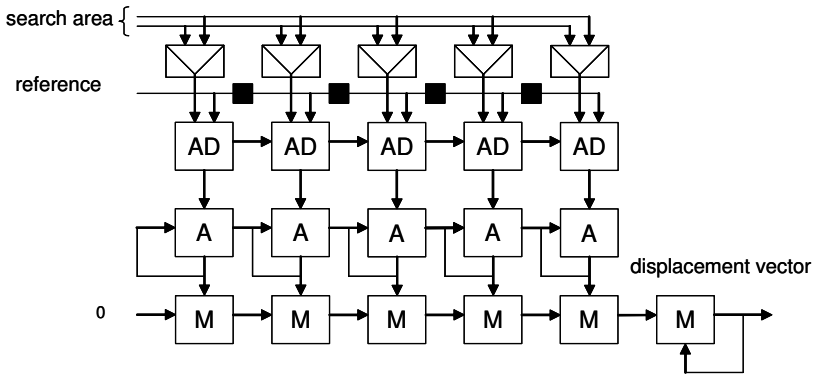


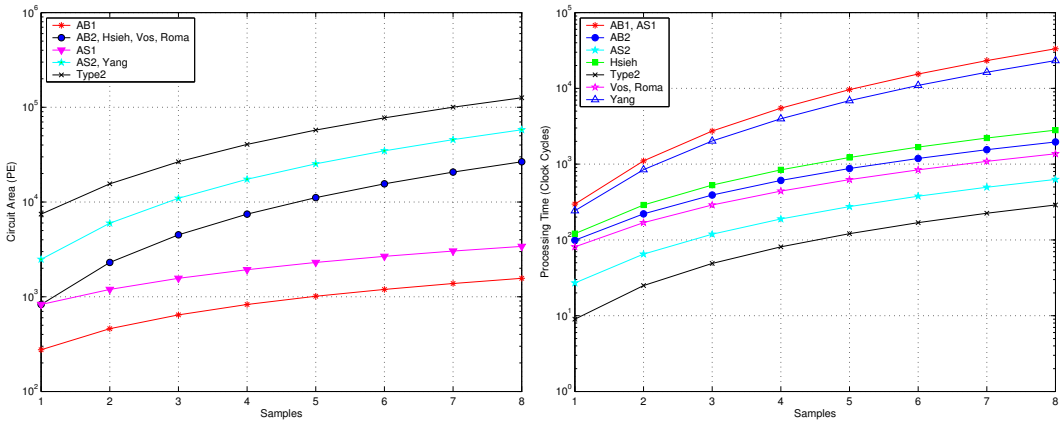
Figure 5.11: Yang's architecture.

cycle, in the second proposed architecture all the results are available at the same time. The architectures were firstly described for a fixed block size $n = 16$ resulting in a search area of 32×32 pixels. They are based on 16 PE linear array and can process one line of candidates in 255 clock cycles. 16 additional time cycles are used to feed the array. Flexibility consists in the fact that the same hardware containing 16 PEs can be configured by a very simple control signal to process different block sizes (8×8, 16×16, 32×32) as long as the tracking range is fixed to 16 searches in one coordinate.

Table 5.1 summarises all the proposed architectures. We can conclude saying that in the AB1 and AB2 structures the number of PEs depends on the search window size, making them suitable for applications with small motion between frames. On the other hand in AS1, AS2 and type 2 it depends on the block size and are advantageous when larger search areas are required. In most of the architectures the processing time increases with the search window size, sometimes depending on the block size, too. We may notice that in the type 2 structure the processing time only depends on the block size, this makes it appropriate for applications where time is critical. Figure 5.12 shows a visual comparison in terms of performance by plotting the processing time and the number of PE corresponding to every structure analysed above. The work in this thesis focuses on linear array, as these structures are more appropriate for our application.

Architecture	**No. of PEs**	**Time Cycles**
Single PE	1	$n^2 \times (2p+1)^2$
AB1	n	$n \times (2p+1) \times (2p+n)$
AB2	$n \times n$	$(2p+1) \times (2p+n)$
AS1	$(2p+1)$	$n \times (2p+1) \times (2p+n)$
AS2	$n \times (2p+1)$	$n \times (2p+1)$
Vos	$n \times n$	$(2p+1)^2$
Roma	$n \times n$	$(2p+1)^2$
Hsieh	$n \times n$	$(2p+n)^2$
TypeII	$(2p+1) \times (2p+1)$	n^2
Yang	$2p+1$	$n \times (2p+1)^2$

Table 5.1: Performance and resources corresponding to the proposed linear and quadratic arrays for FSBMA.



a) b)

Figure 5.12: Comparison of the performance of linear and quadratic architectures for motion estimation based on FSBMA. a)Circuit Area. b)Processing Time.

5.3 Choosing an appropriate architecture when the complexity of the matching criteria increases

The complexity of the motion estimation algorithm presented in chapter 4 results in some restriction for choosing the most appropriate array architecture. For instance, a quadratic array is suitable only in the case of a simple PE architecture. When a lot of computation must be done in parallel it can use up a lot of resources, which are sometimes not available. This is the reason why we restrict our analysis to linear arrays. One solution for reducing the latency introduced by both AB1 and AS1 structures is to increase the memory bandwidth. When every PE is supplied with data coming from an external memory, idle cycles can be avoided. However, when accessing external memory is a constraint, the latency can also be reduced by controlling the data-flow through the array using registers and multiplexors. Vos [92], Roma [76] and Hsieh [40] applied this strategy in order to make the AB2 quadratic architecture more efficient and Yang [97] applied it to improve the performance of linear arrays.

Our approach estimates the motion of certain point from the image, this reduces the number of operations but increases the complexity of memory addressing. The complexity of the chosen motion estimation algorithm determine the restriction of

Architecture	**Time Cycles**	**No. of PE**	**No. of Mem. Access**
AB1	$n \times (2p+1) \times (2p+n)$	n	2
AB1_reduced	$n \times (2p+1)^2$	n	$n+1$
AS1	$n \times (2p+1) \times (2p+n)$	$2p+1$	2
AS_reduced	$n^2 \times (2p+1)$	$2p+1$	$(2p+1)+1$
Yang arch.	$n \times (2p+1)^2$	$(2p+1)$	3

Table 5.2: Linear arrays: performance and resources.

the output bandwidth, while a post processing calculus must be performed. The aim is to design a motion estimation architecture that reduces the input/output bandwidth, maintains the hardware efficiency and reduces the execution time. Reducing the memory throughput can be solved by adding many registers for the search area and also for reference block. Vos proposed a solution to reduce the space occupied in the device by replacing the registers with memory blocks, this reduces half of the silicon occupancy. In the new FPGA technology the available embedded memory blocks can save a lot of logic elements. A part from low memory bandwidth, the solution proposed by Yang [97] also reduces the latency, being the most efficient out of the proposed linear arrays. But as any other solution, it also has some small inconveniences. One is that, for intensive hardware utilisation the restriction $(2p+1) = n$ must be fulfilled. Appendix A shows the data-flow of a linear array based on Yang's proposal.

Table 5.2 analyses the proposed structures for performance vs. resources point of view. AB1_reduced and AS1_reduced correspond to structures similar to AB1 and AS1 but in this case each PE is fed from a separate memory source, therefore reducing the number of time cycles but increasing the memory throughput. In chapter 6 the motion estimation algorithm based on normalised correlation will be mapped into a linear array of PEs. Three linear structured presented above will be analysed in the context of our algorithm.

Chapter 6

Proposal of a VLSI Architecture for Motion Estimation Based on MNCC Correlation Criteria

6.1 Introduction

Our aim in the present work is to develop efficient hardware implementations, starting from a high-level description and analysis of the matching algorithm. In order to enable hardware implementation, a number of transformations must be performed in the algorithms. The matching algorithm involves applying region operators over a whole image. These types of algorithm display data-parallelism which are well suited to implementation in a regular architecture. The implementation aims to detect pairs of point-matchings at video rate. Parallel execution of tasks can be used in order to achieve such *performance*.

Computational complexity affects the level of parallelism, while several multiplications and accumulations must be performed at the same time. When considering the MNCC correlation criteria from chapter 4, square root and division operations may also be performed. This is also an influential factor when choosing the adequate architecture.

Silicon area must be used optimally to implement the computation so that data storage and control parts are minimised. For data storage internal or external mem-

ory can be used. In FPGA devices, embedded memory blocks can be used for the storage of structures such as FIFOs. An image consists of a large amount of data, therefore it is adequate to store it in an external memory. The frequency used for interfacing with the external memory has an important impact on the total execution time of the algorithm. Therefore architecture must be carefully chosen to reduce the memory throughput. Savings of circuit area can also be achieved by using external controllers to communicate with the host computer.

Flexibility needs to be an important characteristic of our implementation, when some parameters of the algorithm are changed, the newly generated hardware must be valid. The architecture must be designed in such a way to permit changing of these parameters. Flexibility can refer either to the architecture or to the implementation. The architecture must be able to support variation of the algorithm's parameters such as image size, number of corners, correlation and search window size, etc. Indeed, it implies parametrisation of the implementation. Choosing an adequate description language allows migration of the design to different hardware platforms.

Complexity reduction refers to avoiding floating-point units which are area expensive. Both corner detection and matching algorithms make use of the division operation. Transformation of these algorithms must be performed in such a way that the computation involves only fixed-point arithmetic.

As we see in chapter 4, motion estimation is just an initial step in building an underwater photo-mosaic which will be further used in a robot localisation system. By partitioning the motion estimation algorithm, computational-intensive parts can be accelerated by means of hardware implementation, while decision tasks can be executed on the host processor.

6.2 Proposal of hardware implementation of the matching algorithm

The goal of the algorithm is to detect point correspondences between the current image acquired by the camera and a previous reference image. Underwater images are difficult to process due to the medium transmission properties and non-uniform illumination [25]. These aspects can provoke undesirable bad correspondences (*out-*

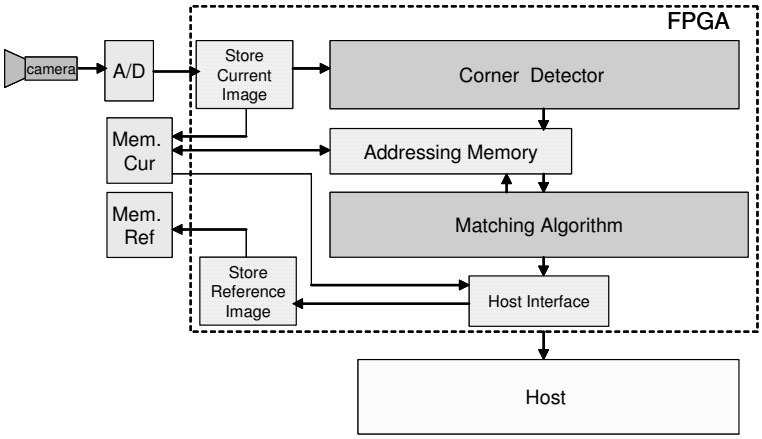


Figure 6.1: Matching algorithm: block diagram.

liers) that can introduce errors in the motion estimation process. For this reason MNCC correlation proved to be a very suitable method reducing the influence of non-uniform illumination [100, 11]. Priori to a full hardware realisation, the algorithm was analysed and discussed in chapter 4 of this thesis.

In order to compute the correspondences between the current image and the reference image the following steps are performed:

1. Store the current image into an external memory;
2. Receive the reference image from the host and store it into an external memory;
3. Detect corners as the incoming pixels are input from the camera;
4. Address the data-pixel corresponding to every selected corner;
5. Find correspondences of points from the current image in the reference image;
6. Send the correspondences and the current image to the host.

Figure 6.1 shows the block diagram of the implementation of the algorithm.

6.2.1 Corner detector: implementation details

The first step in solving the correspondence problem is the detection of a set of well-contrasted points in the current image. The incoming pixel is stored in an

external memory so that a memory address is associated to every pixel. The goal of the real-time corner detector is to provide the memory address corresponding to N interest points from the image. The corner detector algorithm proposed by Benedetti et al. [4] is chosen for the computation of the "cornerness" value. The steps of the algorithm are the following:

- Compute the image gradient components I_x and I_y by convolving the current image with the 3×3 Prewitt masks from equation (6.1).

$$H = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} V = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix} \tag{6.1}$$

- Compute a, b and c from matrix G (see equation (4.8) from chapter 4, for every pixel in an $m \times m$ window. The window size was experimentally set up to 3×3 pixels, as shown in equation (6.2). MATLAB® tests showed that larger windows provide quite similar results.

$$\begin{aligned} a &= \sum_{k=1}^{m} (I_x^k)^2 \\ b &= \sum_{k=1}^{m} (I_x^k \cdot I_y^k) \\ c &= \sum_{k=1}^{m} (I_y^k)^2 \end{aligned} \tag{6.2}$$

- Compute P_{λ_t} from equation (6.3).

$$P_{\lambda_t}(i,j) = (a - \lambda_t)(c - \lambda_t) - b^2 \tag{6.3}$$

- Select every pixel that satisfies the condition from equation (6.4)

$$P_{\lambda_t}(i,j) > 0 \quad and \quad a(i,j) > \lambda_t \tag{6.4}$$

  where λ_t has been experimentally chosen to be equal to 1000.

- Perform non-maximal suppression to replace each pixel with the maximum value within a 3×3 window.

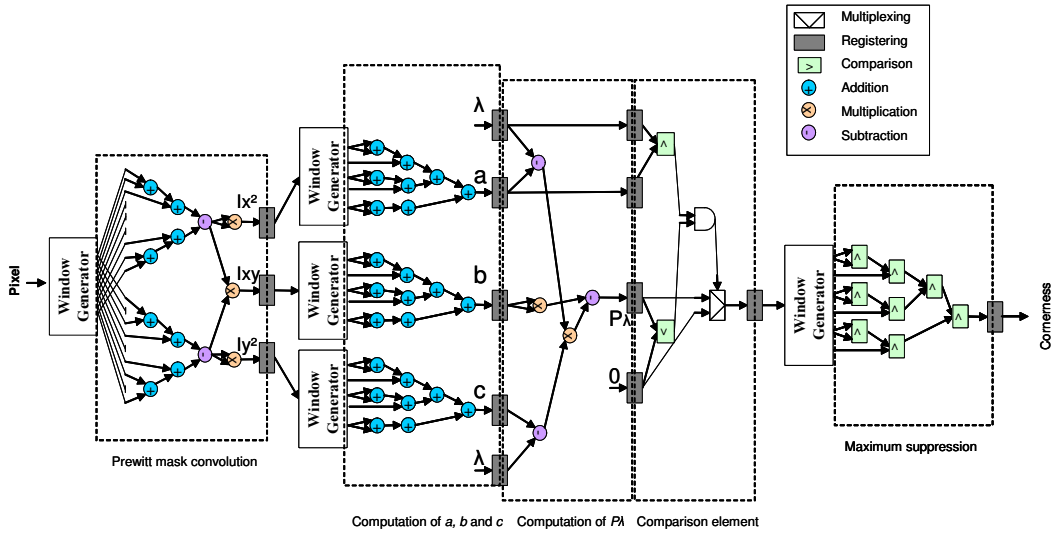


Figure 6.2: Corner detector: Data-flow.

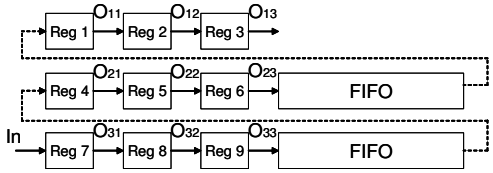


Figure 6.3: Window generator.

Figure 6.2 shows the data-flow corresponding to every step in the computation of P_{λ_t}. The *window generator* delays the incoming pixels and has as a result a 3×3 size image patch. Two FIFOs and nine buffers are used to delay the pixels as shown in figure 6.3. This method has often been used in the implementation of image convolution [5].

In order to retain the pixels with N maximum values for P_{λ_t}, a pipeline of N Sort-Processing Elements (SPE) is proposed. Every SPE compares the input pixel value with the one stored in its register and retains the biggest one (see figure 6.4). An external signal can empty the SPEs buffers at the end of each frame. For large bit representation of P_{λ_t} value, the array of processing elements can "eat up" a large amount of resources.

A compromise must be established between the number of interest points to be retained and the logic to be used up. Even in software, sorting algorithms are one of the most time and resource-consuming algorithms. While every incoming

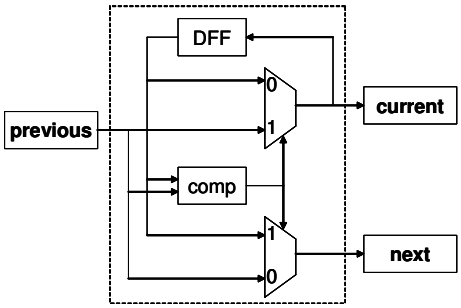


Figure 6.4: Sorting Processing Element: internal structure.

pixel must be compared with the N previous pixels, many comparisons and re-allocation operations must be performed in one cycle. One solution is to set up an off-line threshold and perform the matching algorithm for an unknown number of corners. The drawback of this method is that we cannot determine the complexity of the matching algorithm as we do not know the exact number of operations to be performed.

Another solution can be to reduce the number of comparisons. This new strategy is based on a *time-sharing* procedure. Two comparisons could be performed in one cycle. Thus, one SPE must store the two previous pixel values. The data must be stored in additional registers in such way that when an insertion is performed, the replaced datum will not be lost. This approach reduces the number of SPEs to $N/2$ but increases the complexity of the data-flow control.

Figure 6.5 shows the block diagram corresponding to each step in the corner detection.

Number of bits analysis

Fine-grained implementation offers us the possibility of number of bits optimisation. If the input image signal is sampled using *"signal_width"* number of bits, table 6.1 shows the bit-representation of the result corresponding to each step. The typical A/D conversion corresponds to an 8 bit representation of the pixel intensity value. This is the common bit-representation for gray-scale images. The maximum number of bits of the cornerness value corresponding to this pixel codification, is 48 bits. Of course, this size is invariant to image size and number of corners to be detected and only depends on the bit-representation of the pixel intensity value.

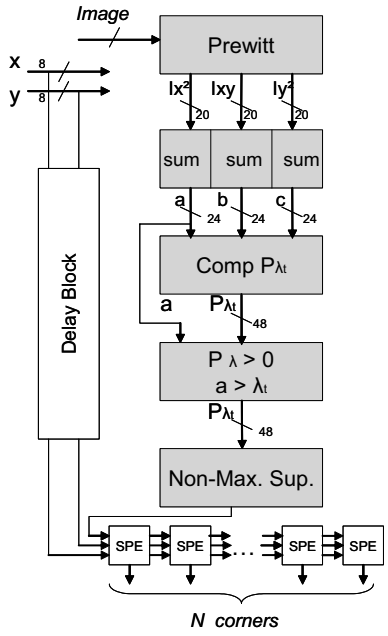


Figure 6.5: Corner detector: block diagram.

Steps	**Number of bits**	**Signal_width**= 8
$Prewitt$	$2 \times signal\_width + 4$	20
Sum	$(2 \times signal\_width + 4) + 4$	24
P_{λ_t}	$2 \times ((2 \times signal\_width + 4) + 4)$	48

Table 6.1: Bit Size depending on the input signal size.

MATLAB® tests performed using either synthetic image or several underwater images, proved that the maximum value for number of bits of the cornerness value lies between 24 and 34 bits (see table 6.2). Reducing the number of bits from 48 to 34 saves a considerable amount of resources (about 3000 LEs).

6.2.2 MNCC correlation criteria: implementation details

Once interest points are detected in the current image, the algorithm searches for correspondences in the reference image. The local gray-level correlation algorithm provides, for each interest point from the current image, its correspondent match in the reference image. The correlation score is defined as the covariance between the grey levels of a region defined by the *correlation window* in the current image and the same region defined in the reference image. The algorithm searches for all

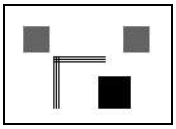
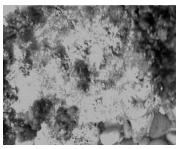
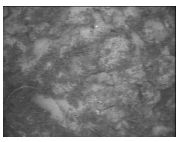
Image	Max. value of C	Max. number of bits
	11414000000	34 bits
	272192706	29 bits
	641717010	30 bits
	15100401	24 bits
	926914406	30 bits
	314415171	29 bits

Table 6.2: Bit size corresponding to maximum cornerness value for different image samples: synthetic, indoor and underwater images.

candidate windows inside the correspondent *search window*. The MNCC criteria C, which ensures that the result is not altered in presence of non-uniform illumination is recalled in equation (6.5).

$$C = \frac{\sum_{i=-\alpha}^{\alpha}\sum_{j=-\alpha}^{\alpha}(I_c(x_c+i,y_c+j)-\overline{I_c(x_c,y_c)})(I_r(x_r+i,y_r+j)-\overline{I_r(x_r,y_r)})}{(2\alpha+1)^2\sqrt{\sigma^2(I_c)\cdot\sigma^2(I_r)}} \tag{6.5}$$

where $\alpha = (n-1)/2$; $n \times n$ is the size of the correlation window. $\overline{I_c(x_c,y_c)}$ and $\overline{I_r(x_r,y_r)}$ are the average intensity and $\sigma^2(\cdot)$ defines the variance of both correlation windows. The algorithm compares the correlation score of each pixel within the search window and selects the highest one.

FSBMA's survey	**Our approach**
SAD criteria	MNCC criteria
Reference block	Correlation window
Search area	Search window
Reference block size= $n \times n$	Correlation window size= $(2\alpha+1) \times (2\alpha+1)$
Search area size= $(2p+n) \times (2p+n)$	Search window size= $(2p+1) \times (2p+1)$

Table 6.3: Corresponding notation and window sizes for our approach.

In chapter 4 we analysed a number of specialised architectures for motion estimation. For the sake of clarity, table 6.3 shows the corresponding terminology and parameter values used in our approach.

The first step in deriving a VLSI architecture for the given algorithm is to break down the C criteria from equation (6.5) into simple operations which can be executed in parallel. We can observe that there are five sums to be computed in equation (6.5): sum_1, sum_2, sum_3, sum_4 and sum_5.

$$\begin{aligned}
sum_1 &= \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} I_c(x_c+i, y_c+j) \\
sum_2 &= \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} I_c(x_c+i, y_c+j)^2 \\
sum_3 &= \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} I_c(x_c+i, y_c+j) \cdot I_r(x_r+i, y_r+j) \\
sum_4 &= \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} I_r(x_r+i, y_r+j)^2 \\
sum_5 &= \sum_{i=-\alpha}^{\alpha} \sum_{j=-\alpha}^{\alpha} I_r(x_r+i, y_r+j)
\end{aligned} \tag{6.6}$$

Then, equation (6.5) becomes:

$$C = \frac{sum_3 - \frac{1}{(2\alpha+1)^2} \cdot sum_1 \cdot sum_5}{\frac{1}{(2\alpha+1)^2} \cdot \sqrt{[(2\alpha+1)^2 \cdot sum_2 - sum_1^2] \cdot [(2\alpha+1)^2 \cdot sum_4 - sum_5^2]}} \tag{6.7}$$

This breaking down simplifies the parallel implementation while each Processing Element (PE) of the architecture can execute in parallel the computation of these five sums. A Post Processing Element (PPE) performs the remaining computation.

Finding correspondences is the most time consuming part of our algorithm. Let us consider N interest points detected in the current image. For every interest point, we are searching for $[(2p+1)-2\alpha]^2$ possible correspondences, where $p = (q-1)/2$

and $q \times q$ is the size of the search window. Considering the breaking down of the correlation criteria into five sums as described in equation (6.6), two accumulations and three multiplication-accumulations have to be computed for every pixel from the correlation window. Grouping these sums by means of correlation criteria from equation (6.5), twelve additional computation steps for every candidate block are required. For a frame-rate f_r, the complexity of the correspondence problem becomes:

$$O_p = [(3 * 2 + 2) \times (2\alpha + 1)^2 + 12] \times [(2p + 1) - 2\alpha]^2 \times N_p \times f_r \qquad (6.8)$$

Therefore, for $N_p = 100$, $\alpha = 7$, $p = 14$, at a frame-rate of 25 frames per second we have $O_p \simeq 2036$ GOPS (Giga Operations Per Second). Our approach tries to speed up the execution by means of a parallelization of the correspondence problem.

6.2.3 Array of processing elements

VLSI architectures for motion estimation presented in chapter 5 can be adapted easily to our algorithm. The complexity of the processing element made us decide to set apart the quadratic arrays, so that only linear arrays are analysed in this proposal. Both AB1 and AS1 structures are analysed. Figure 6.6 represents the AB1-type and AS1-type architectures adapted to our design. The architectures correspond to the experimental correlation window size of 3×3 and search window size of 7×7. When comparing them, it can be stated that AS1 strategy reduces the number of time cycles but increases the number of processing elements. On the other hand, AB1-type architecture is suitable for applications where large motion vectors must be estimated, implying large search areas. When the application requires a faster processing speed, AS1-type architecture can achieve higher performance than AB1-type for small search window sizes.

Our approach is based on the idea introduced by Yang et al. [98] which provides an important contribution towards reducing the memory throughput compared with both AB1 and AS1 array structures. Figure 6.7 shows this strategy applied to our algorithm. Another advantage of this architecture is that it also reduces the number of time-cycles necessary for the computation of the correlation criteria. There is only a one cycle delay between the results corresponding to each candidate in a line, while in an AB1 structure there is a delay of $(n-1)$ cycles. The drawback of this approach

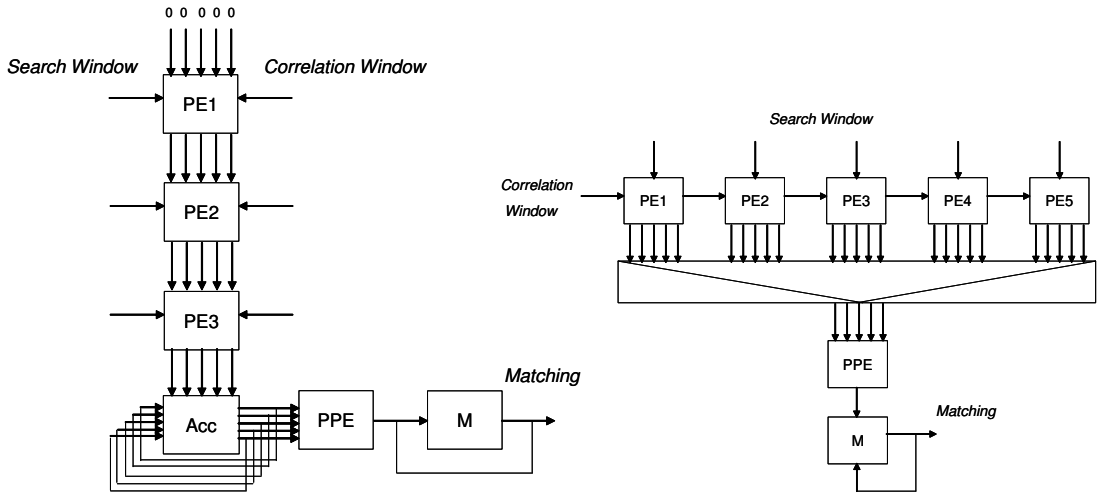


Figure 6.6: Linear architectures to implement normalised correlation algorithm. a) AB1-type; b) AS1-type.

is that it processes one column less than the others, so that "dummy" data should be input into the array. This can be easily understood by drawing the data-flow. For the maximum efficiency of the architecture, the size of the search window must depend on the size of the correlation window by the relation $p = 2\alpha$. Table 6.4 shows the data-flow corresponding to one line of candidates for the experimental window sizes: 3×3 correlation window and 7×7 search window. This proposal is an efficient architecture, due to its low memory throughput, small delay and the fact that the result of the PE's array can easily be pipelined into a PPE without extra control requirements. Therefore, we have chosen this array structure for our further implementation.

6.2.4 Processing element

A processing element may execute two accumulations and three multiply accumulations in parallel. Figure 6.8 presents the internal structure of one PE in both cases: AB1-type and AS1-type (Yang's architecture). In AB1 structure the PE performs three multiplications and five additions as the accumulation is done in a separate block at the end of the PE's array. On the other hand, in AS1 structure the accumulations are done in the PE. This increases the size of the PE but simplifies the control, while each PE has the same structure.

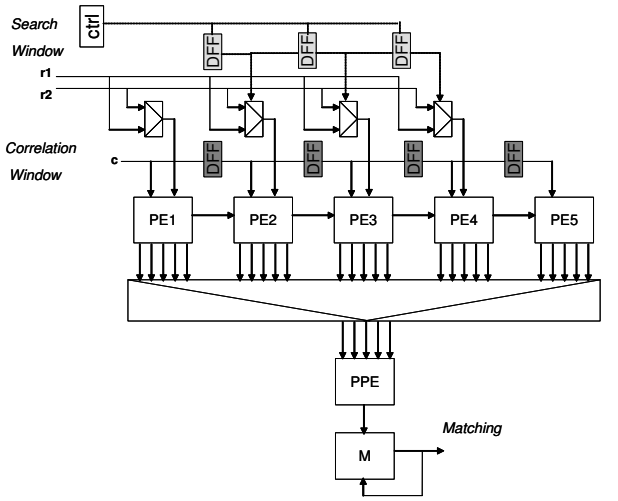


Figure 6.7: Proposal of a VLSI architecture to implement normalised cross-correlation.

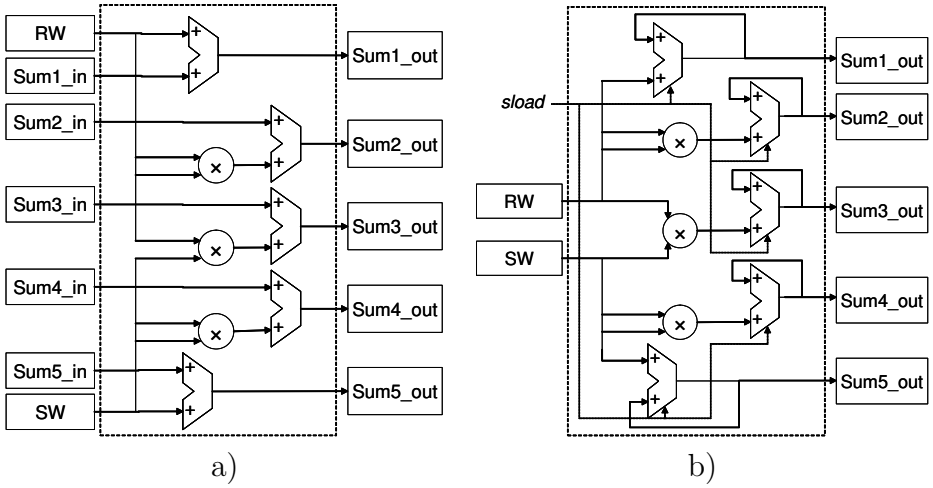


Figure 6.8: Processing element: internal structure. a) AB1-type; b) AS1-type(Yang).

It is obvious that a single AB1-type PE requires less silicon area than AS1-type PE, where several multiply-accumulations must be performed in parallel, at the price of increasing the control complexity and the memory throughput. This helps the designer to choose between a solution to saves hardware and a second choice to reduce memory access.

PE: analysis of the number of bits

Every PE has five outputs corresponding to sum_1, sum_2, sum_3, sum_4 and sum_5. The number of bits of sum_1 is equal to the number of bits of sum_5 and the number

time	c	r1	r2	PE0	PE1	PE2
0	a(0,0)	b(0,0)		a(0,0); b(0,0)		
1	a(0,1)	b(0,1)		a(0,1); b(0,1)	a(0,0); b(0,1)	
2	a(0,2)	b(0,2)		a(0,2); b(0,2)	a(0,1); b(0,2)	a(0,0); b(0,2)
3	a(1,0)	b(1,0)	b(0,3)	a(1,0); b(1,0)	a(0,2); b(0,3)	a(0,1); b(0,3)
4	a(1,1)	b(1,1)	b(0,4)	a(1,1); b(1,1)	a(1,0); b(1,1)	a(0,2); b(0,4)
5	a(1,2)	b(1,2)	b(0,5)	a(1,2); b(1,2)	a(1,1); b(1,2)	a(1,0); b(1,2)
6	a(2,0)	b(2,0)	b(1,3)	a(2,0); b(2,0)	a(1,2); b(1,3)	a(1,1); b(1,3)
7	a(2,1)	b(2,1)	b(1,4)	a(2,1); b(2,1)	a(2,0); b(2,1)	a(1,2); b(1,4)
8	a(2,2)	b(2,2)	b(1,5)	a(2,2); b(2,2)	a(2,1); b(2,2)	a(2,0); b(2,2)
9			b(2,3)		a(2,2); b(2,3)	a(2,1); b(2,3)
10			b(2,4)			a(2,2);b(2,4)

→ Correlation window data flow

b(1,5) Dummy dada

b(2,3) Multiplexer switched to r2

Table 6.4: Data-flow for the proposed architecture.

of bits of sum_2 is equal to the number of bits of sum_3 and sum_4. In the case of AB1, one PE applies the arithmetic operators to the outputs of the previous PE, therefore the number of bits of the inputs and the outputs of each PE vary through the array of the processing elements. On the other hand, in AS1-type every PE has the same structure and therefore, the corresponding results have the same number of bits. While three accumulation of pixels from an image patch is performed by every PE, the resulting number of bits depends on the correlation window size ($(2\alpha+1)\times(2\alpha+1)$) and the bit-representation of the data-pixel. The way to compute the number of bits in the case of accumulation (b_{acc}) and multiply-accumulation (b_{m_acc}) when the data-pixel is represented using 8 bits, is by applying the $\log_2$ function and rounding the value to the nearest integer towards infinity (see equation (6.10). Table 6.5 shows, for several correlation window sizes, the corresponding number of bits.

$$\begin{aligned} b_{acc} &= \lceil \log_2(255\times(2\alpha+1)\times(2\alpha+1))\rceil \\ b_{m\_acc} &= \lceil \log_2(255\times 255\times(2\alpha+1)\times(2\alpha+1))\rceil \end{aligned} \tag{6.9}$$

Correlation window	sum_1, sum_5	sum_2, sum_3, sum_4
7×7	14 bits	22 bits
9×9	15 bits	23 bits
11×11	15 bits	23 bits
13×13	16 bits	24 bits
15×15	16 bits	24 bits

Table 6.5: Number of bits corresponding to different correlation windows.

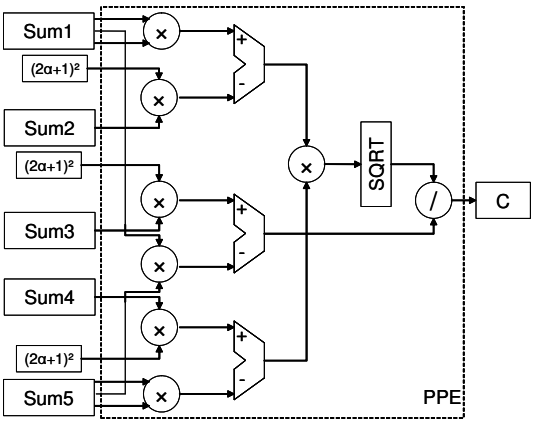


Figure 6.9: PPE: internal structure.

6.2.5 Post processing element

The Post Processing Element (PPE) is one of the critical parts of our design. The results from the array of PEs are pipelined into the PPE. The PPE computes the correlation criteria defined in equation (6.7). Seven multiplications, three substraction, one 64-bit square root and one 32-bit division have to be implemented in hardware. Parallel implementation of these operations is performed. The internal structure of the PPE is shown in figure 6.9. Square root (SQRT) is the most time and silicon consuming operation. This arithmetic operation is important in many image processing applications. Solutions such as Look Up Tables tried to replace this operation in some early image processing applications [30]. However, the flexibility and the performance of recent reconfigurable devices allow the implementation of efficient arithmetic operations for individual applications. In general, to calculate $Q = \sqrt{D}$, an approximate value can be obtained through iteration. Methods like Newton-Raphson and a non-restoring square root algorithms were adopted by many authors for VLSI implementation of SQRT [60, 61, 83, 58]. Our implementation is

based on the non-restoring algorithm proposed by Li [60]. The advantage of this method is the area and time efficiency of the implementation which uses only a single adder and subtracter for the iterative operations and generates an exact result value.

Let us assume that we want to compute a square root of 64 bits unsigned radicand $D = D_{63}D_{62}D_{61}...D_1D_0$. The resulting quotient can be represented using 32 bits: $Q = Q_{31}Q_{30}Q_{29}...Q_1Q_0$ and the reminder will be $R = R_{32}R_{31}R_{30}...R_1R_0$. 32 iterations should be performed. The steps of the non-restoring SQRT algorithm are as follows:

1. Set $Q_{31} = 0$, $R_{32} = 0$ and then iterate from $k = 31$ to 0

2. If $r_{k+1} \geq 0$ then $r_k = r_{k+1}D_{2k+1}D_{2k} - q_{k+1}01$
   else $r_k = r_{k+1}D_{2k+1}D_{2k} + q_{k+1}11$

3. If $r_{k+1} \geq 0$ then $q_k = q_{k+1}1(i.e., Q_k = 1)$
   else $q_k = q_{k+1}0(i.e., Q_k = 0)$

4. Repeat steps 2 and 3 until $k = 0$.
   If $r_{k+1} < 0$ then $r_0 = r_0 + q_01$

The result of the non-restoring algorithm is given by the integer values of the quotient Q and the reminder R. Tests using MATLAB® proved that neglecting the remainder, the results are similar.

Division is the last operation performed by the PPE computational block. As in the square root algorithm, the hardware implementation of the division provides a quotient and a remainder. The floating point division gives values between $[-1, 1]$. In order to simplify the implementation, the numerator is shifted 20 bits to the left and the remainder of the division is ignored. Tests with real underwater image sequences proved that this transformation guarantees the right precision for the good functionality of the matching algorithm. Of course it affects the cost of the implementation, doubling the circuit area used by this operator.

The quotient of the division operation is correlation measurement (C). Maximisation of the C corresponding to each candidate is performed in the last block (M from figures 6.6 and 6.7). The "good match" corresponds to the candidate with the

Correlation window	b_{M_1}	b_{M_2}	b_{M_3}	b_{SQRT}	b_{DIV}
7×7	28 bits	28 bits	56 bits	28 bits	48 bits
9×9	29 bits	29 bits	58 bits	29 bits	49 bits
11×11	30 bits	30 bits	60 bits	30 bits	50 bits
13×13	31 bits	31 bits	62 bits	31 bits	51 bits
15×15	32 bits	32 bits	64 bits	32 bits	52 bits

Table 6.6: Number of bits corresponding to the result of the arithmetic operations applied to different correlation window sizes.

maximum value of C. The result of the algorithm is the address of the pixel from the reference image characterised by the maximum value for the correlation score.

PPE: analysis of the number of bits

The number of bits in PPE depends on the results of the array of PEs. Depending on the number of bits three types of multiplication can be considered: M_1, M_2 and M_3, one square root (SQRT) operation and one division (DIV):

$$\begin{aligned} b_{M_1} &= \lceil \log_2((255 \times (2\alpha+1) \times (2\alpha+1) \times (255 \times (2\alpha+1) \times (2\alpha+1)) \rceil \\ b_{M_2} &= \lceil \log_2((255 \times 255 \times (2\alpha+1) \times (2\alpha+1)) \times ((2\alpha+1) \times (2\alpha+1)) \rceil \\ b_{M_3} &= b_{M_1} + b_{M_2} \\ b_{SQRT} &= b_{M_1}/2 \\ b_{DIV} &= b_{SQRT} + 20 \end{aligned} \tag{6.10}$$

Table 6.5 shows, for several correlation window sizes, the number of bits corresponding to every operation in PPE.

6.2.6 Memory access

Our approach uses two external memories for storing the current frame and the previous frame. While the incoming pixel is stored in one memory, the data corresponding to the previous frame is processed by the array of PEs. In this way the system can process every frame. In order to be accessed at the same time as the current image, the reference image is stored in a separate external memory.

The corner detector provides the memory addresses corresponding to N interest

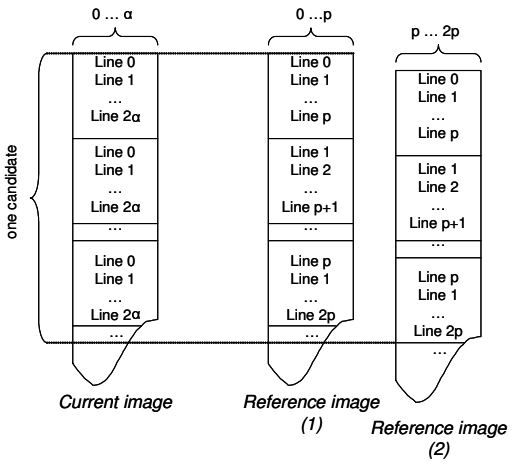


Figure 6.10: Data-flow: current and reference image.

points. The matching algorithm requires image patches surrounding every interest point. The *correlation window* characterises the point from the current image and the *search window* is a searching area from the reference image. The addresses corresponding to the data-pixel from both, correlation window and search window can be easily computed from the memory address of the interest point. Let us consider point P selected as a corner. ${}^cA_P$ is the corresponding address from the memory storing the current image and ${}^rA_P$ is the corresponding address from the memory storing the reference image. If the correlation window has a size of $(2\alpha + 1) \times (2\alpha + 1)$, the search window size is $(2p + 1) \times (2p + 1)$ and $line\_size$ is the number of pixels of an image-line, then the memory address corresponding to an arbitrary point A_M can be obtained as shown in equation (6.11).

$$\begin{aligned} {}^cA_M &= {}^c A_P \mp k({}^c line\_size + 1), \quad where \quad k \in [-\alpha, \alpha] \\ {}^rA_M &= {}^r A_P \mp j({}^r line\_size + 1), \quad where \quad r \in [-p, p] \end{aligned} \tag{6.11}$$

In our proposed structure for the array of PEs, data-pixel flows into the array in a line-scan mode as it is shown in table 6.4. Line-scan mode makes addressing control easier, because consecutive locations are read in a line. Two reference data-pixels must be accessed in parallel, therefore a second external memory is used to store the reference image. Figure 6.10 represents the corresponding data-flow to feed the array of PEs.

6.3 Results

6.3.1 Prototyping platforms and CAD tools

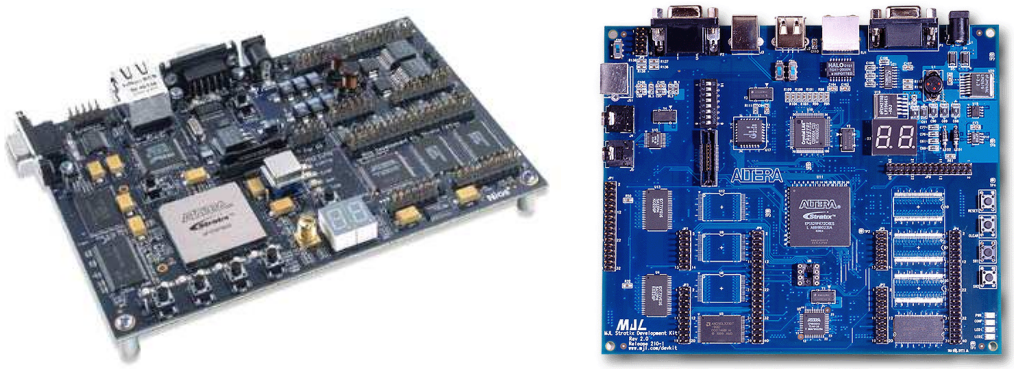
Testing and validation of the design proposed in this chapter were carried out through the implementation of the proposed methods for motion estimation on a target reconfigurable platform based on FPGA device. This was accomplished by describing the algorithm in VHDL and then synthesising it for the FPGA device.

Prior to any hardware design we implemented a MATLAB® software version corresponding to every step of the algorithm. MATLAB® is a tool which facilitates procedural routines to operate on images represented as a matrix data. The software implementation of the algorithms has two important roles: to choose the most adequate algorithm for our application and to provide benchmark results for hardware implementation.

The application was targeted for FPGA devices for many reasons. One of the goals of this thesis is to propose a systolic architecture for motion estimation using matching criteria more robust than SAD criteria. FPGAs can be considered for high performance DSP systems. On the other hand, nowadays FPGA devices offer very attractive hardware facilities: great I/O pin-count, embedded memory blocks, large logic area, high clock speed and software and hardware embedded processors. Advanced software CAD tools are available for assisting every design stage.

The VHDL language was chosen for hardware design, foremost because of familiarity but also due to its wide supporting range. Parameterisable VHDL blocks were implemented allowing the application parameters to be changed. Modularity of the design makes possible the reuse of parts of it for other applications. The ModelSim® simulation tool was used for design verification together with Altera's QUARTUS® II design software. QUARTUS® II is a CAD design tool which can assist each step in the design-flow and provide an extensive analysis of timing and resources utilisation (embedded memory blocks, logic elements and dedicated multipliers).

Altera's Statix family was chosen as hardware target, mostly because its accessibility and low cost. Important characteristics such as embedded multipliers and memory blocks were also an influential factor. We benefit from the Nios Development Kit Stratix 10K Edition (figure 6.11 (a)) and MJL Nios Development Kit Stratix 25K (figure 6.11 (b)) hardware platforms to synthesise and test the algo-



(a) (b)

Figure 6.11: (a) Altera Nios Development Kit Stratix 10k edition (b) MJL Nios Development Kit Stratix 25K

.

rithm.

6.3.2 Practical considerations for the video signal

In image processing we think of an image as a two-dimensional array of intensity or colour data. A PAL camera, however, outputs a one-dimensional stream of analog data. The purpose of an analogue-to-digital (A/D) converter is to convert the video analogue signal into digital one using a certain sampling frequency. In order to be able to process image information, we need to know the structure of the video signal being acquired. An analog video signal consists of a low-voltage signal containing the intensity information for each line in combination with timing information used for synchronisation: *horizontal and vertical synchronism*. Figure 6.12 shows the characteristics of an analogue video line. The 82% of the analog line signal, equivalent to $52,48\mu s$, represents *active video* while rest is used for the synchronism and other signal information.

When converting to digital signal, the number of pixels in a line is defined by the sampling frequency. In a PAL standard video system, one frame is composed by 625 lines. The lines of one frame are distributed into two fields: "odd" and "even", which means $312,5$ lines per field. Subtracting the vertical synchronism length, equivalent to $25,5$ lines, 288 active video lines remain. Depending on the application, the image can be either processed for every frame (odd and even fields)

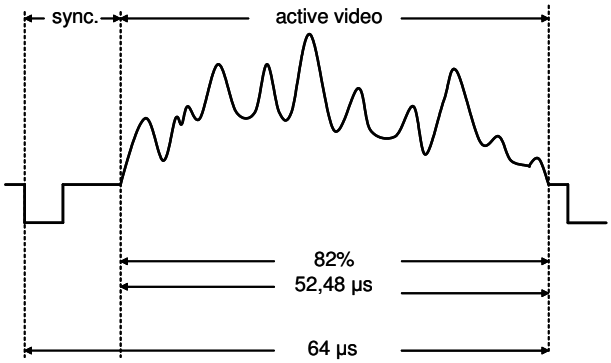


Figure 6.12: Analogue video line.

Image Size	**Sampling**	**Comments**
630×288	12 MHz	Large amount of data to be processed.
524×288	10 MHz	Enlarge Image, Distortion.
420×288	8 MHz	Good image size. Close to the $4:3$ ratio.
210×144	4 MHz	Reduced image size if the application requires.

Table 6.7: Sampling frequencies and corresponding image size processing one field of a standard interlaced PAL camera.

or we can only process one of the two fields. In the case of underwater mosaicking, the robot is moving at slow speed therefore, the necessity of processing every field is not justified.

Sampling at a frequency of 8 MHz we obtain 512 samples per line, 420 active pixels. Choosing another sampling frequency we can change the number of sampled pixels per line. Table 6.7 presents some of the possible image sizes. Different image sizes can be used, depending on the requirements of the application.

From the point of view of hardware implementation, we have two types of constraints imposed by the execution time and circuit area. In most of the cases, when increasing the computation speed, more parallelism is required, that means more circuit area to be used. When the real-time requirements allow more time to be spent in solving the computation, silicon can be saved. The figure 6.15 shows how tasks execution must be distributed along one frame. There is no need for high parallelism, when available time is not entirely spent.

In our approach we consider an 8 MHz sampling frequency and the corresponding 420×288 image size. The frame rate of 25 frames per second is equivalent to 0.04

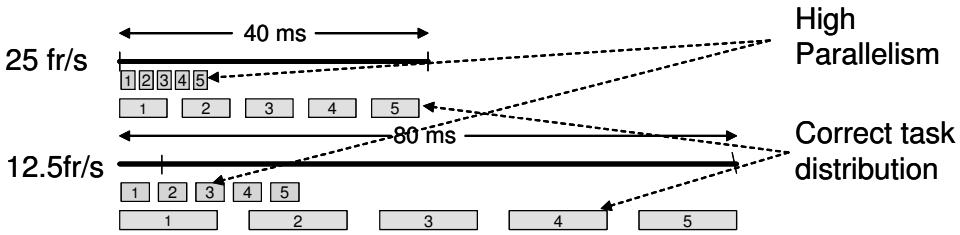


Figure 6.13: Correct distribution of tasks at frame-rate execution.

Parameters	**Value**
Sampling Frequency	8 MHz
Image Size	420×288 pixels
Bit-representation	8 bits
Frame-rate	0.04s
Bit-representation	8 bits

Table 6.8: Video signal parameters.

seconds, the time between the acquisition of the first and the last pixel of the frame. Changing the sampling time or reducing the image size can be carried out when processing requires more time. Concerning the analogue-to-digital signal conversion, an 8-bit sampling will be used in our application.

6.3.3 Timing and resources analysis

Timing and *resources* analysis are important issues when designing the architecture. Both, the corner detection and matching algorithms were implemented in parameterisable VHDL code. VHDL implementation functionality can be tested using compilation, synthesis and simulation tools for hardware design. Simulation offers the possibility of off-line testing and validation of the implementation. An efficient and common method for simulating VHDL code can be done through the use of *test bench* code. In our application it consists of simulating the *image*, the *horizontal and vertical synchronism* and the *clock* signals. The QUARTUS® II tool provides the possibility of simulating a RAM memory and loading the pixel values of real image data. The MATLAB® was used to generate the image data file. The horizontal and vertical synchronism were used as control signals to check the *end of the line* or the *end of the frame*.

Stage	**Time**	**Accumulated time**
Corner Detector	0.170 s	
Matching Algorithm	0.370 s	**0.540s**
Outlier Rejection	0.340 s	0.880 s
Motion Estimation	0.060 s	0.940 s
Mosaic Construction	0.210 s	1.150 s

Table 6.9: Time corresponding to different stages of the mosaicking algorithm performed off-line using an AMD-K7 1300 MHz processor.

The goal of this work is to accelerate the execution of low-level image processing tasks of the motion estimation algorithms in order to use them *on-line*. Table 6.9 shows the timing corresponding to an *off-line* execution of the algorithm on a general purpose processor platform. Our new strategy imposes *frame-rate performance* (0.04s) for the execution of the first two stages of the algorithm as described in table 6.9. In this way, the performance is increased more than 10 times.

Corner detectorion: timing analysis

The delay introduced by the corner detector is very important, since the matching algorithm needs the memory address of all selected corner instead of their value of cornerness. Every 3×3 window generator introduces a latency of two lines and two pixels. The delay introduced by the computation of the cornerness value is shown in equation (6.12) and being a function of the image size ($M_i \times N_i$), pixel sampling time (t_s) and the number of time-cycles necessary for the computational blocks illustrated in figure 6.2: *Prewitt* (t_{Pr}), *Sum* (t_{Sum}), *Compute* P_{λ_t} (t_P) , comparison element (t_{CE}) and *Maximum Suppresion*(t_{MS}).

$$T_{P_\lambda} = [3 \cdot (2 \cdot M_i + 3) + t_{Pr} + t_{Sum} + t_P + t_{CE} + t_{Ms}] \cdot t_s \tag{6.12}$$

Many operations can be done in a single clock cycle while the clock-cycle is high enough, being equal to the image sampling frequency ($4, 8, 10$ or 12 MHz). Synthesising the design for our test device, showed that the internal maximum frequency which can be used to clock the computations is 87.07 MHz. The dashed line from figure 6.2 represents the operations that can be executed in a single clock cycle. Thus, the time spent for computation of the cornerness value corresponding

Image Size	Sampling	$\mathbf{T_{P_\lambda}}$	Total
629×288	12 MHz	383.62 μs	396.12 μs
524×288	10 MHz	308.60 μs	321.10 μs
420×288	8 MHz	316.75 μs	329.75 μs
210×144	4 MHz	317.00 μs	330.25 μs

Table 6.10: Cornerness computation time corresponding to different image sizes and sampling frequencies.

to every pixel becomes:

$$T_{P_\lambda} = [3 \cdot (2 \cdot M_i + 3) + 5] \cdot t_s \tag{6.13}$$

The delay introduced by the computation of the cornerness corresponding to different image sizes is shown in table 6.10. Considering that N additional time-cycles are introduced by the array of SPEs, the delay introduced by the detection of N corners is shown in the last column of the table.

Corner detection: analysis of resources

Computation of the cornerness value represented by P_{λ_t} from equation (6.4) uses multiplication, addition and subtraction operations. Every incoming pixel is stored in a FIFO structure which is implemented using internal memory. The QUARTUS® II design tool offers us the possibility to synthesise the design for a special device and analyse the resource requirements. Table 6.11 presents the results of our design synthesis.

The critical part in the implementation of the corner detection hardware is the array of SPEs. There are two main factors which affect the cost of the sorting algorithm: number of corners and number of bits of P_{λ_t}. A number of $N \in [100, 200]$ corners can assure an accurate motion estimation. But sorting 200 corners requires approximately 18000 LEs which is about 72% of the Stratix EP1S25F672FPGA device used in our experiments. On the other hand when sorting 100 corners the SPE array occupies approximately 9000 LEs out of 25000 available on our testing device. An important fraction of the FPGA's resources is gained by reducing P_{λ_t} bit-representation from 48 to 34 (see table 6.12). The worse case is represented by the synthetic image from table 6.2 which corresponds to a size of 34 bits. Therefore,

Resources	LE	DSP blocks	Memory bits
Cornerness computation	1658	22	9352
Total available (EP1S25F672)	25660	80	1944576

Table 6.11: Resources analysis for the computation of P_{λ_t}.

Number of bits	Nr.of LE
48	9700
45	9100
34	6900
30	6100

Table 6.12: Resources used by the array of SPE corresponding to different bit representation of the P_{λ_t} when selecting 100 corners.

we choose 34 bits to represent the value of P_{λ_t}. Our solution propose the selection of 100 corners represented using 34 bits.

Matching algorithm: timing analysis

The delay introduced by the matching algorithm depends on the correlation window size $((2\alpha+1)\times(2\alpha+1))$ and search window size $((2p+1)\times(2p+1))$. Given N interest points detected in the current image, the algorithm is searching among $[(2p+1)-2\alpha]^2$ possible correspondences. Figure 6.14 shows the time cycles necessary to search for the matching of one interest point. When searching for the correct match of one interest point the accumulated delay depends on the delay introduced by the array of PEs (T_{PE}) and the delay introduced by the post processing (T_{PPE}). The delay introduced by the array of PEs is shown in equation (6.14):

$$T_{PE} = (((2\alpha + 1)(p + 1) \times (2p + 1 - 2\alpha)) + 2\alpha) \times f_c \tag{6.14}$$

where f_c is the computation frequency. Twelve additional arithmetic operations are necessary to compute the correlation measurement (see equation (6.7)). Multiplications can be computed in parallel. The square root algorithm requires as many time-cycles as half of the radicand's number of bits. The division can also be done in one cycle, so that the time introduced by the PPE mainly depends on the square root algorithm. The computation frequency is restricted by the internal maximum

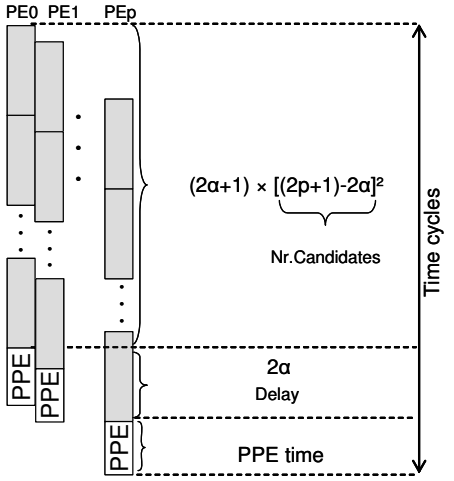


Figure 6.14: Time-cycles necessary to search for the matching of one interest point.

Corr. window	**Search window**	T_{PE}	T_{PPE}	**Total**
7×7	13×13	23.24 μs	2.05 μs	27.29 μs
9×9	17×17	48.64 μs	2.11 μs	50.75 μs
11×11	21×21	88.50 μs	2.18 μs	90.97 μs
13×13	25×25	145.79 μs	2.24 μs	148.03 μs
15×15	29×29	223.67 μs	2.31 μs	225.95 μs

Table 6.13: Delay introduced by the matching algorithm for a clock frequency $f_c = 15$ MHz.

frequency at which the design can be clocked (19.47 MHz). Table 6.13 shows the corresponding time cycles for different correlation and search window sizes.

Matching algorithm: analysis of resources

Many high cost arithmetic operations, such as multiplication, division or square root, must be performed. The circuit area used by the implementation of the array of PEs and the PPE element depends on the correlation window size as it is shown in table 6.14. Even if the operation are the same, by increasing the correlation window, more elements have to be accumulated. It increases the bit representation of the operands, in consequence, the size of the operator.

Corr. window	PE: LE	PE: DSP bl.	PPE: LE	PPE: DSP bl.
7×7	1032	21	1735	14
9×9	1243	27	1896	14
11×11	1461	33	1947	14
13×13	1815	36	2068	14
15×15	2454	42	2125	14

Table 6.14: Resource analysis: array of PEs and PPE.

Stage	LE	DSP blocks	Memory bits	Time
Corner detection	1658	22	9352	316.75 μs
Array of SPE	6900	0	0	12.50 μs
Array of PE	2454	42	0	223.67 μs
PPE	2125	14	0	2.31 μs
Total 100 points	**51.2% 13138**	**87.5% 78**	**0.48% 9352**	**64.9% 25.96 ms**
Total available	**25660**	**80**	**1944576**	**40.00ms ms**

Table 6.15: Resources and timing analysis for our proposal: Image size = 420×288; Number of corners: 100; Correlation window = 15×15; Search window = 29×29; Computation frequency $f_c = 15$ MHz.

6.3.4 Resource and timing analysis for our proposed architecture

Let us consider an image size of 420×288. When the sampling frequency is 25 frames per second, the time available for detecting the correspondences between two images is 40 ms. The goal of our design is to obtain frame-rate performance while restricting the implementation to the available resources. Table 6.15 shows the results for every stage of the implementation. The algorithm performs the matching of 100 points characterised by correlation windows of 15×15 pixels and looking for candidates in a search area of 29×29 pixels. The device used in our experimental tests is an FPGA from the Altera's Stratix family (EP1S25F672).

As the matching algorithm is the most time consuming part, its computation frequency is the main parameter influencing the final time. The available time is 40ms and, by clocking the computation at 30 MHz, only a third part of this time is spent for solving the correspondence problem. The rest of the time is spent establishing communication with the host. It implies getting the reference image and storing it into the external memories and sending, the pair of point-matching

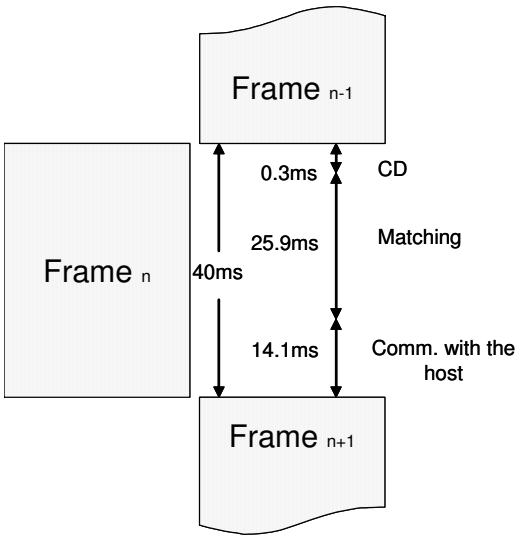


Figure 6.15: Task scheduling for solving the correspondence problem.

and the current image. If the system is based on a PCI communication, an image ($120960 data - pixels$) needs about 3.66 ms when considering PCI frequency of 33 MHz. The system needs to receive the reference image from the host computer and send the pairs of point-matching and the current image. The available time for these operations (about 14 ms) is enough to establish the right communication with the host.

6.4 Summary

This chapter proposes an architecture to accelerate tasks in the vision-based motion estimation algorithm. The two main parts of the algorithm are: the corner detector and the matching procedure.

- A real-time implementation for selecting N high-contrasted points from an image is proposed. As the data enters into the system, it computes the corresponding cornerness value and, at the end of each frame, provides the memory addresses of the N corners.

- A parallel implementation for the matching algorithm is also proposed. The algorithm is broken down into parts which can be computed in parallel in an array of complex processing elements. The array's structure is chosen in such

way as to reduces both the memory throughput and the execution time and to facilitate the post-processing of the data.

- An analysis of the algorithm's implementation including tests using real underwater images has been performed to select the optimal design.

Chapter 7

Conclusions and Future Work

7.1 Summary

The position and orientation of an underwater robot can be calculated by integrating the apparent motion of the images acquired by a down-looking camera carried by the robot. To estimate this motion, the images taken by the camera are processed, and a new composite image –known as photo-mosaic– can be created.

We have presented an approach for solving the correspondence problem in underwater imaging. The solution allows real-time computation of matched pairs of points between images.

The apparent motion of a camera mounted on an underwater vehicle can be estimated by describing the differences between successive frames of an image sequence. These differences can be found by correlating features extracted from the image. Matching algorithms are used to correlate these features. However, searching for correspondences is a time-consuming and error-prone process. Lack of well-defined contours caused by blurring of the elements of the image, as well as non-uniform illumination when using artificial light make underwater scenes much more difficult to be processed than normal images. Therefore, methods frequently used in standard image processing must be modified and adapted to these particular conditions.

The *correspondence problem* is one of these problems we analysed in this thesis. Intensity-based criteria such as sum of absolute differences (SAD) or sum of squared differences (SSD) have often been used to solve the correspondence problem. Exper-

iments using real underwater image sequences proved that more complex methods such as mean normalised cross-correlation (MNCC) provide much better results. Even so, when dealing with more difficult conditions such as rotation or scaling motions or moving objects (fishes, algaes, etc.) the presence of "bad-correlated" points (known as *outliers*) was detected. Texture information proved to be a rich source of information for features characterisation.

In this thesis we proposed a technique based on the exploitation of gray level information complemented with texture cues to eliminate bad-correlated points from the matching algorithm presented in chapter **??**. This has proved to be a robust method which can easily be applied to reject outliers. When compared to the the existing probabilistic methods, such as LMedS or RANSAC, our new approach does not need any forehand estimation of either fundamental matrix, in the case of a $3D$ scene, or homography in the case of a planar scene.

Finding matching points from a pair of images involves low-level and intermediate-level image processing tasks such as feature detection and correlation algorithms. These tasks are characterised by quite simple and repetitive operations applied to a large amount of data. In order to cope with the needs of a real application, powerful systems are required to perform this data-intensive operations in real-time. Considering their very attractive performance combined with a high-level of flexibility, reconfigurable devices, including Field Programmable Gate Arrays (FPGA), bring a new dimension to real-time image processing. A review of several existing FPGA-based image processing systems has been performed in chapter 3. Characteristics of FPGA devices such as reconfigurability, high input/output throughput or possibility of low-level configuration opened many perspectives to the research community. The state of the art in this field includes multiple FPGA systems, FPGA and microprocessor architectures, run-time reconfigurable systems, co-design approach and evolvable hardware. Advances in VLSI technology and design tools have had a great impact on the rapid evolution of reconfigurable devices based systems. Applications which required several FPGAs few years ago, may occupy only a small circuit area in the new devices. Integration of embedded multipliers, memory blocks and even processors offers new possibilities for algorithm's implementation.

A great research effort has been carried out to develop efficient architectures to make possible real-time execution and single chip implementation to solve the correspondence problem. Chapter 5 carries out an analysis and comparison of systolic

architecture for matching algorithms. These architectures have been widely applied in video coding applications. Applying concepts from this field, this thesis proposed a new specialised architecture for a matching algorithm taking into consideration a more complex criteria which proved to give satisfactory results in underwater imaging. The architecture has been tested on reconfigurable platforms (see chapter 6). Available design tools made possible an extensive analysis of resources and performance of our design. A parameterisable implementation of the algorithm allows the same architecture model to be applied to different image sizes, number of features to be selected and different parameters of correlation algorithm.

7.2 Contribution of this research

Two different research directions can be distinguished in this thesis, both with the same goal: to improve methods for motion estimation in the difficult conditions we face in underwater imaging. One contribution has been done in the algorithmic part and another one in the implementation of the algorithm:

a) We have proposed a method based on texture characterisation of points to reject outliers to solve the image correspondence problem;

b) A comparative study of the state-of-the-art of VLSI architectures for matching algorithms has been carried out;

c) We have proposed a VLSI architecture for the implementation of a feature-based matching algorithm. The mean absolute cross-correlation is used to correspondence problem while it has been proved it works much better than the other criteria in underwater imaging.

7.3 Future work

Since the use of underwater robots for the exploration of the submerged environment, underwater imaging is a relatively new subject. This thesis has opened several possible new areas for further investigation and research. The main topics for future work are described below.

- One of the possible further works is the realisation in hardware of the proposed texture based algorithm. Applying texture operator is a relatively easy task to be implemented in hardware. While these operators are applied over a large amount of image-data, parallel execution of the operations can speed up the computation time. The only limitation till now was the size of available reconfigurable platforms, but rapidly evolution of VLSI technology and design tools allows higher and higher integration of algorithms in reconfigurable devices.

- Concerning the integration of the proposed architecture in a real robot system, an optimal communication with the host processor is a must. It should allow the host to send the reference image to the hardware accelerator and the hardware system to send the data (pairs of point-matching) and the current image to the host. The host may actualise the mosaic image and extract the reference image to be send to the accelerator.

- A new trend in partitioning an algorithm over hardware an software platforms is the hardware/software codesign. This concept can be applied at a higher abstraction level. A great amount of research is carrying on to develop tools and concepts to make easier this process. One possible future work consist in applying such tools for the partitioning the mosaicking algorithm over hardware and software platforms.

7.4 Related publications

The publications listed below are direct consequence of the evolution of this dissertation in the last four years:

- V. Ila, P.Ridao, J. De La Cruz and J.Batlle "Implementation of a Real-time Target Tracking Behaviour Using Video Sensors", IFAC Conference on Control Applications in Marine Systems (CAMS2001), Glasgow, Scotland, U.K.

- V. Ila, P. Ridao and J. Batlle "A New Parallel Architecture to Deal With Real-Time Computer Image Processing" International Workshop on Systems, Signals and Image Processing (IWSSIP2001), Bucharest, Romania, June 2001.

- V. Ila, J. Batlle, X. Cufi, R Garcia "New Trends in FPAA Devices", International Workshop on Systems, Signals and Image Processing (IWSSIP2002), Manchester, UK, November 2002.
- R. Garcia, X. Cufí and V. Ila, "Recovering Camera Motion in a Sequence of Underwater Images through Mosaicking", Lecture Notes in Computer Science no. 2652, pp. 255-262, Eds. Springer-Verlag, 2003.
- V. Ila, R. Garcia, F. Charot, J. Batlle, "Proposal of a Parallel Architecture for a Motion Detection Algorithm", International Conference on Pattern Recognition (ICPR2004), Camridge, UK, August 2004.
- V. Ila, R.Garcia, F.Charot and J. Batlle "FPGA Implementation of a Vision-Based Motion Estimation Algorithm for an Underwater Robot", Field Programmable Logic and its Applications (FPL2004), Antwerp, Belgium, August 2004.
- V. Ila, R. Garcia, J. Batlle "Reconfigurable Architecture to Estimate the Motion of an Underwater Vehicle" Control Engineering and Applied Informatics (CEAI), vol:6, pg: 9-15, June 2004.
- V. Ila, R.Garcia and F.Charot "VLSI Architecture for an Underwater Robot Vision System", Oceans Europe, Brest, France, June 2005.
- V. Ila, R. Garcia, "Interest Point Characterization through Textural Analysis for Rejection of Bad Correspondences", Accepted to the Pattern Recognition Letters, April 2005.

Appendix A

Linear Array for Motion Estimation

The proposed linear architecture for the hardware implementation of the motion estimation algorithm is based on the system introduced by Yang et. al [84] and adapted to our application. In order to easily understand how the data flow through the array of processing elements, figure A.2 shows the data corresponding to each register from the architecture. For the sake of representation, the example considered in this figures corresponds to a correlation window size of 3×3 and a search window size of 5×5 (see figure A.1 (b)).

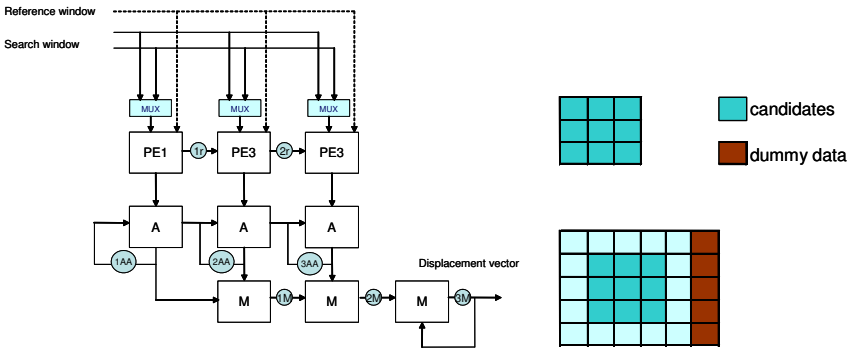


Figure A.1: (a) Array architecture for motion estimation algorithm. (b) Correlation and search windows

Search Window		Ref. Wind.	PE1	1r	PE2	2r	PE3
11		r11	s11-r11				
12		r12	s12-r12	r11	s12-r11		
13		r13	s13-r13	r12	s13-r12	r11	s13-r11
21	14	r21	s21-r21	r13	s14-r13	r12	s14-r12
22	15	r22	s22-r22	r21	s22-r21	r13	s15-r13
23	16	r23	s23-r23	r22	s23-r22	r21	s23-r21
31	24	r31	s31-r31	r23	s24-r23	r22	s24-r22
32	25	r32	s32-r32	r31	s32-r31	r23	s25-r23
33	26	r33	s33-r33	r32	s33-r32	r31	s33-r31
21	34	r11	s21-r11	r33	s34-r33	r32	s34-r32
22	35	r12	s22-r12	r11	s22-r11	r33	s35-r33
23	36	r13	s23-r13	r12	s23-r12	r11	s23-r11
31	24	r21	s31-r21	r13	s24-r13	r12	s24-r12
32	25	r22	s32-r22	r21	s32-r21	r13	s25-r13
33	26	r23	s33-r23	r22	s33-r22	r21	s33-r21
41	34	r31	s41-r31	r23	s34-r23	r22	s34-r22
42	35	r32	s42-r32	r31	s42-r31	r23	s35-r23
43	36	r33	s43-r33	r32	s43-r32	r31	s43-r31
31	44	r11	s31-r11	r33	s44-r33	r32	s44-r32
32	45	r12	s32-r12	r11	s32-r11	r33	s45-r33
33	46	r13	s33-r13	r12	s33-r12	r11	s33-r11
41	34	r21	s41-r21	r13	s34-r13	r12	s34-r12
42	35	r22	s42-r22	r21	s42-r21	r13	s35-r13
43	36	r23	s43-r23	r22	s43-r22	r21	s43-r21
51	44	r31	s51-r31	r23	s44-r23	r22	s44-r22
52	45	r32	s52-r32	r31	s52-r31	r23	s45-r23
53	46	r33	s53-r33	r32	s53-r32	r31	s53-r31
	54			r33	s54-r33	r32	s54-r32
	55					r33	s55-r33
	56						

1M	2M	3M
F1		
	min(F1,F2)	
		min(F1,F2,F3)
min(F1,F2,F3,F4)		
	min(F1,F2,F3,F4,F5)	
		min(F1,F2,F3,F4,F5,F6)
min(F1,F2,F3,F4,F5,F6,F7)		
	min(F1,F2,F3,F4,F5,F6,F7,F8)	
		min(F1,F2,F3,F4,F5,F6,F7,F8,F9)

1AA	2AA	3AA
s11-r11		
s11-r11+s12-r12	s12-r11	
s11-r11+s12-r12+s13-r13	s12-r11+s13-r12	s13-r11
s11-r11+s12-r12+s13-r13+s21-r21	s12-r11+s13-r12+s14-r13	s13-r11+s14-r12
s11-r11+s12-r12+s13-r13+s21-r21+s22-r22	s12-r11+s13-r12+s14-r13+s22-r21	s13-r11+s14-r12+s15-r13
s11-r11+s12-r12+s13-r13+s21-r21+s22-r22+s23-r23	s12-r11+s13-r12+s14-r13+s22-r21+s23-r22	s13-r11+s14-r12+s15-r13+s23-r21
s11-r11+s12-r12+s13-r13+s21-r21+s22-r22+s23-r23+s31-r31	s12-r11+s13-r12+s14-r13+s22-r21+s23-r22+s24-r23	s13-r11+s14-r12+s15-r13+s23-r21+s24-r22
s11-r11+s12-r12+s13-r13+s21-r21+s22-r22+s23-r23+s31-r31+s32-r32	s12-r11+s13-r12+s14-r13+s22-r21+s23-r22+s24-r23+s32-r31	s13-r11+s14-r12+s15-r13+s23-r21+s24-r22+s25-r23
s11-r11+s12-r12+s13-r13+s21-r21+s22-r22+s23-r23+s31-r31+s32-r32+s33-r33	s12-r11+s13-r12+s14-r13+s22-r21+s23-r22+s24-r23+s32-r31+s33-r32	s13-r11+s14-r12+s15-r13+s23-r21+s24-r22+s25-r23+s33-r31
s21-r11	s12-r11+s13-r12+s14-r13+s22-r21+s23-r22+s24-r23+s32-r31+s33-r32+s34-r33	s13-r11+s14-r12+s15-r13+s23-r21+s24-r22+s25-r23+s33-r31+s34-r32
s21-r11+s22-r12	s22-r11	s13-r11+s14-r12+s15-r13+s23-r21+s24-r22+s25-r23+s33-r31+s34-r32+s35-r33
s21-r11+s22-r12+s23-r13	s22-r11+s23-r12	s23-r11
s21-r11+s22-r12+s23-r13+s31-r21	s22-r11+s23-r12+s24-r13	s23-r11+s24-r12
s21-r11+s22-r12+s23-r13+s31-r21+s32-r22	s22-r11+s23-r12+s24-r13+s32-r21	s23-r11+s24-r12+s25-r13
s21-r11+s22-r12+s23-r13+s31-r21+s32-r22+s33-r23	s22-r11+s23-r12+s24-r13+s32-r21+s33-r22	s23-r11+s24-r12+s25-r13+s33-r21
s21-r11+s22-r12+s23-r13+s31-r21+s32-r22+s33-r23+s41-r31	s22-r11+s23-r12+s24-r13+s32-r21+s33-r22+s34-r23	s23-r11+s24-r12+s25-r13+s33-r21+s34-r22
s21-r11+s22-r12+s23-r13+s31-r21+s32-r22+s33-r23+s41-r31+s42-r32	s22-r11+s23-r12+s24-r13+s32-r21+s33-r22+s34-r23+s42-r31	s23-r11+s24-r12+s25-r13+s33-r21+s34-r22+s35-r23
s21-r11+s22-r12+s23-r13+s31-r21+s32-r22+s33-r23+s41-r31+s42-r32+s43-r33	s22-r11+s23-r12+s24-r13+s32-r21+s33-r22+s34-r23+s42-r31+s43-r32	s23-r11+s24-r12+s25-r13+s33-r21+s34-r22+s35-r23+s43-r31
s31-r11	s22-r11+s23-r12+s24-r13+s32-r21+s33-r22+s34-r23+s42-r31+s43-r32+s44-r33	s23-r11+s24-r12+s25-r13+s33-r21+s34-r22+s35-r23+s43-r31+s44-r32
s31-r11+s32-r12	s32-r11	s23-r11+s24-r12+s25-r13+s33-r21+s34-r22+s35-r23+s43-r31+s44-r32+s45-r33
s31-r11+s32-r12+s33-r13	s32-r11+s33-r12	s33-r11
s31-r11+s32-r12+s33-r13+s41-r21	s32-r11+s33-r12+s34-r13	s33-r11+s34-r12
s31-r11+s32-r12+s33-r13+s41-r21+s42-r22	s32-r11+s33-r12+s34-r13+s42-r21	s33-r11+s34-r12+s35-r13
s31-r11+s32-r12+s33-r13+s41-r21+s42-r22+s43-r23	s32-r11+s33-r12+s34-r13+s42-r21+s43-r22	s33-r11+s34-r12+s35-r13+s43-r21
s31-r11+s32-r12+s33-r13+s41-r21+s42-r22+s43-r23+s51-r31	s32-r11+s33-r12+s34-r13+s42-r21+s43-r22+s44-r23	s33-r11+s34-r12+s35-r13+s43-r21+s44-r22
s31-r11+s32-r12+s33-r13+s41-r21+s42-r22+s43-r23+s51-r31+s52-r32	s32-r11+s33-r12+s34-r13+s42-r21+s43-r22+s44-r23+s52-r31	s33-r11+s34-r12+s35-r13+s43-r21+s44-r22+s45-r23
s31-r11+s32-r12+s33-r13+s41-r21+s42-r22+s43-r23+s51-r31+s52-r32+s53-r33	s32-r11+s33-r12+s34-r13+s42-r21+s43-r22+s44-r23+s52-r31+s53-r32	s33-r11+s34-r12+s35-r13+s43-r21+s44-r22+s45-r23+s53-r31
	s32-r11+s33-r12+s34-r13+s42-r21+s43-r22+s44-r23+s52-r31+s53-r32+s54-r33	s33-r11+s34-r12+s35-r13+s43-r21+s44-r22+s45-r23+s53-r31+s54-r32
		s33-r11+s34-r12+s35-r13+s43-r21+s44-r22+s45-r23+s53-r31+s54-r32+s55-r33

Figure A.2: Data flow corresponding to an experimental correlation window size of 3×3 and a search window size of 5×5. The columns represent the registers from figure A.1 (a)

Bibliography

[1] H. Araújo and J. Diaz. An introduction to log-polar mapping. In IEEE Computer Society Press, editor, *Proceedings of the II Workshop on Cybernetic Vision*, pages 139–146, December 1996.

[2] P. M. Athanas and A. L. Abbott. Real-time image processing on a custom computing platform. *Computer*, 28:16–25, February 1995.

[3] P. Baglietto, M. Maresca, A. Migliaro, and M. Migliardi. Parallel implementation of the full search block matching algorithm for motion estimation. In *International Conference on Application Specific Array Processors*, pages 182–192, July 1995.

[4] Arrigo Benedetti and Pietro Perona. Real-time 2-d feature detection on a reconfigurable computer. In *Computer Vision and Pattern Recognition*, pages 586–593, 1998.

[5] A. Benedetti, A. Prati, and N. Scarabottolo. Image convolution on FPGAs: the implementation of a multi-FPGA FIFO structure. In *Euromicro Conference, 1998*, volume 1, pages 123–130, 1998.

[6] P. J. Bentley, T. G. W. Gordon, J. Kim, and S. Kumar. New trends in evolutionary computation. In *Evolutionary Computation*, volume 1, pages 162–169, 2001.

[7] P. Bertin and H. Touati. PAM programming environments: practice and experience. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 133–138, 1994.

[8] R. C. Bolles and M. A. Fischler. A RANSAC-based approach to model fitting and its application to finding cylinders in range data. In *6th Interna-*

tional Joint Conference on Artificial Intelligence, pages 637–643, Vancouver, Canada, August 1981.

[9] S. Bouchoux, E.-B. Bourennane, J. Miteran, and M. Paindavoine. Implementation of JPEG2000 arithmetic decoder on a dynamically reconfigurable ATMEL FPGA. In *IEEE Computer society Annual Symposium on VLSI*, pages 237–238, 19-20 February 2004.

[10] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–210, 2002.

[11] X. Cufí, R. García, and R. Ridao. An approach to vision-based station keeping for an unmanned underwater vehicle. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 799–804, Lausanne, 2002.

[12] A. DeHon and J. Wawrzynek. Reconfigurable computing: what, why, and implications for design automation. In *Design Automation Conference*, pages 610–615, 1999.

[13] R. Deriche and O. Faugeras. Tracking line segments. In *Proceedings of the First European Conference on Computer Vision*, pages 259–268, 1990.

[14] H. De Garis, A. Buller, L. de Penning, T. Chodakowski, M. Korkin, G. Fehr, and D. Decesare. Initial evolvability experiments on the CAM-brain machines (cbms). In *Congress on Evolutionary Computation*, volume 1, pages 635–642, 2001.

[15] H. De Garis, A. Buller, L. de Penning, T. Chodakowski, M. Korkin, G. Fehr, and D. Decesare. Initial evolvability experiments on the CAM-brain machines (cbms). In *Evolutionary Computation*, volume 1, pages 635–642, 2001.

[16] H. de Garis, L. de Penning, A. Buller, and D. Decesare. Early experiments on the cam-brain machine (cbm). In *The Third NASA/DoD Workshop on Evolvable Hardware*, pages 211–219, 2001.

[17] H. de Garis. The second NASA/DoD workshop on evolvable hardware. In *IEEE Transactions on Evolutionary Computation*, pages 298–302, June 2001.

[18] M. J. B. Duff. Thirty years of parallel image processing. In *Selected Papers and Invited Talks from the 4th International Conference on Vector and Parallel Processing*, pages 419–438. Springer-Verlag, 2001.

[19] R. Duncan. A survey of parallel computer architectures. *IEEE Computer*, (23):5–16, 1990.

[20] C. Ebeling, D.C. Cronquist, and P. Franklin. Rapid - reconfigurable pipelined datapath. In *FPL'96: Proceedings of the 6th International Workshop on Field-Programmable Logic, Smart Applications, New Paradigms and Compilers*, pages 126–135. Springer-Verlag, 1996.

[21] O. Faugeras, B. Hotz, Mathieu H., T. Vieville, Z. Zhang, P. Fua, E. Theron, L. Moll, G. Berry, J. Vuillemin, P. Bertin, and C. Proy. Real-time correlation-based stereo: algorithm, implementations and applications. Technical Report 2013, Institut National de Recherche en Informatique et en Automatique, August 1993.

[22] M. J. Flynn. Some computer organizations and their effectiveness. *IEEE Transactions on Computing*, 21(9):948–960, September 1972.

[23] C.J. Funk, S.B. Bryant, and P.J. Beckman Jr. Handbook of underwater imaging system design. In *Ocean Technology Department*, Naval Undersea Center, 1972.

[24] R. García, X. Cufí, and J. Batlle. Detection of matchings in a sequence of underwater images through texture analysis. In *International Conference on Image Processing*, pages 361–364, 2001.

[25] R. García, T. Nicosevici, and X. Cufí. On the way to solve lighting problems in underwater imaging. In *IEEE OCEANS Conference (OCEANS)*, pages 1018–1024, Mississipi, 2002.

[26] R. García. *A Proposal to Estimate the Motion of an Underwater Vehicle through Visual Mosaicking*. PhD thesis, University of Girona, October 2001.

[27] A. Giachetti. Matching techniques to compute image motion. *Image and Vision Computing*, (18):247–260, 2000.

[28] GigaOperations Corporation. *SPECTRUM Reconfigurable Computing Platform Documentation*, Release 2.5 1997.

[29] D. E. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning.* Addison-Wesley Publishing Company, 1989.

[30] M. Gorgon and J. Pryzybylo. FPGA based controller for heterogenous image processing system. In *Euromicro Symposium on Digital Systems, Design.*, pages 453–457, 2001.

[31] M. Gorgon. Universal reprogrammable architecture for implementation dedicated image processors based on FPGA. In *Sixth International Conference on Image Processing and Its Applications*, volume 2, pages 556–560, 1997.

[32] N. Gracias and J. Santos-Victor. Underwater video mosaics as visual navigation maps. *Computer Vision and Image Understanding*, 79(1):66–91, 2000.

[33] R. M. Haralick, K. Shanguman, and I. Dinstein. Textural features for image classification. *IEEE Trans. on Systems, Man and Cybernetics*, pages 610–621, November 1973.

[34] C. G. Harris and M. J. Stephens. A combined corner and edge detector. In *Proceedings of the Fourth Alvey Vision Conference*, pages 147–151, Manchester, 1988.

[35] W. Hartenstein. Coarse grain reconfigurable architecture (embedded tutorial). In *ASP-DAC*, pages 564–570, 2001.

[36] S. Hauck. The roles of FPGAs in reprogrammable systems. *Proceedings of the IEEE*, 86(4):615–638, April 1998.

[37] P. Heysters, G. Smit, and E Molenkamp. A flexible and energy-efficient coarse-grained reconfigurable architecture for mobile systems. *Supercomputing*, 26(3):283–308, 2003.

[38] R. H. Hord. *Parallel Supercomputing in SIMD Architectures.* CRC Press, Inc., 1990.

[39] K. M. Hou, A. Belloum, E. Yao, X.W. Tu, M. Shawky, M. Meribout, J. L. Mayorquim, A. Trihandoyo, and B. Jardin. A reconfigurable and flexible parallel

3D vision system for a mobile robot. In *Computer Architectures for Machine Perception*, pages 215–221, November 1993.

[40] C. H. Hsieh and T. P. Lin. VLSI architecture for block -matching motion estimation algorithm. *IEEE Trans. on Circuits and Systems for Video technology*, 2(2):169–175, June 1992.

[41] http://www.123industries.com/. Secad.

[42] http://www.altera.com/. Altera the programmable solution company.

[43] http://www.annapmicro.com/. Annapolis micro systems. wildfire.

[44] http://www.datacube.com/. Datacube maxrevolution TVAC.

[45] http://www.pactxpp.com/. PACT XPP technologies, solutions for high performance media processing.

[46] http://www.prodesigneurope.com/. Prodesign's visionspeedster2.

[47] http://www.titan.com/. Titan products vigravision.

[48] http://www.xilinx.com/. Xilinx home page for programmable logic.

[49] R. D. Hudson, D. I. Lehn, and P. M. Athanas. A run-time reconfigurable engine for image interpolation. In *IEEE Symposium on FPGAs for Custom Computing Machines*, volume 2, pages 88–95, 1998.

[50] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill Higher Education, 1992.

[51] J. S. Jaffe. The domains of underwater visibility. In *SPIE Ocean Optics VIII*, pages 287–293, 1986.

[52] P. P. Jonker. Morphological image processing. *Architecture and VLSI Design*, 1992.

[53] L. Kessal, D. Demigny, N. Boudouani, and R. Bourguiba. Reconfigurable hardware for real time image processing. In *International Conference on Image Processing*, pages 110–113, 10-13 September 2000.

[54] S. Khawam, T. Arslan, and F. Westall. Domain-specific reconfigurable array for distributed arithmetic. In *FPL*, pages 1139–1144, 2003.

[55] T. Komarek and P. Pirsch. Array architectures for block matching algorithms. *IEEE Transactions on Circuits and Systems*, 36:1301–1308, 10 October 1989.

[56] H. Kung and C. Leiserson. Systolic arrays (for VLSI). In *Sparse Matrix Proceedings*, pages 256–282, 1978.

[57] S. Y. Kung. VLSI array processors: designs and applications. In *IEEE International Symposium on Circuits and Systems*, pages 313–320, 7-9 June 1988.

[58] T. Lang and P. Montuschi. Very high radix square root with prescaling and rounding and a combined division/square root unit. *IEEE Transactions on Computers*, 48(9):Computers, August 1999.

[59] K. I. Laws. *Textured Image Segmentation.* PhD thesis, Processing Institute, University of Southern California, Los Angeles, 1980.

[60] W. Li and W. Chu. A new non-restoring square root algorithm and its VLSI implementations. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors*, pages 538–544, 7-9 October 1996.

[61] M. E. Louie and M. D. Ercegovac. A digit-recurrence square root implementation for field programmable gate arrays. In *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 178–183, 5-7 April 1993.

[62] P. I. Mackinlay, P. Y. K. Cheung, W. Luk, and R. Sandiford. Riley-2: A flexible platform for codesign and dynamic reconfigurable computing research. In *FPL*, pages 91–100, 1997.

[63] M. Meribout and K. M. Hou. Implementation of low level image processing algorithms on a reconfigurable perception system. In *Computer Architectures for Machine Perception*, pages 304–312, 1995.

[64] T. Miyamori and K. Olukotun. A quantitative analysis of reconfigurable coprocessors for multimedia applications. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, page 2. IEEE Computer Society, 1998.

[65] C. A. Moritz, D. Yeung, and A. Agarwal. Exploring cost-performance optimal designs of raw microprocessors. In *The 6th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, Napa, California, April 1998.

[66] S. Neema, J. Scott, and T. Bapty. Real time reconfigurable image recognition system. In *IEEE Instrumentation and Measurement Technology Conference*, volume 1, pages 350–355, 2001.

[67] T. Ojala and M. Pietikäinen. Unsupervised texture segmentation using feature distributions. *Pattern Recognition*, 32:477–486, 1999.

[68] A. Olmos, E. Trucco, K. Lebart, and D. M. Lane. Detecting ripple patterns in mission videos. In *MTS/IEEE OCEANS*, pages 331–335, Rhode Island, September 2000.

[69] D. Perrottet. Evolvable electronic for mobile robots. Master's thesis, At Micro-engineering department, Polytechnic Univercity from Lausanne, Swiss, winter 2000-2001.

[70] K. Plakas and E. Trucco. Developing a real-time, robust, video tracker. In *Proceedings of the MTS/IEEE Conference and Exhibition OCEANS*, pages 1345–1352, 2000.

[71] S. Plimpton, G. Mastin, and D. Ghiglia. Synthetic aperture radar image processing on parallel supercomputers. In *Proceedings of the ACM/IEEE conference on Supercomputing*, pages 446–452. ACM Press, 1991.

[72] J. Prewitt. Object enhancement and extraction. In B. Lipkin and A. Roselfed, editors, *Picture Processing and Psychopictorics*, pages 75–149. Academic Press, New York, 1970.

[73] N. K. Ratha, A. K. Jain, and D. T. Rover. FPGA-based coprocessor for text string extraction. In *5th IEEE International Workshop on Computer Architectures for Machine Perception*, pages 217–221, 2000.

[74] N. K. Ratha and A. K. Jain. FPGA-based computing in computer vision. In *Fourth IEEE International Workshop on Computer Architecture for Machine Perception*, pages 128–137, 1997.

[75] N. Roma and L. Sousa. A new VLSI architecture for full search block matching. In *IFIP International Conference on Very Large Scale Integration (VLSI-SoC'2001)*, pages 213–218, 2001.

[76] N. Roma and L. Sousa. Efficient and configurable full search block matching processors. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(12):1160–1167, December 2002.

[77] P. J. Rousseeuw and A. M. Leroy. *Robust Regression and Outlier Detection*. John Wiley and Sons, New York, 1987.

[78] Y. Rzhanov, L. M. Linnett, and R. Forbes. Underwater video mosaicking for seabed mapping. In *International Condefernce on Image Processing*, 2000.

[79] J. Schwartz and M. Sharir. Identification of partially occluded objects in two and three dimensions by matching noisy characteristic curves. *The International Journal of Robotics Research*, 6:29–43, 1986.

[80] I. K. Sethi and R. Jain. Finding trajectories of feature points in a monocular image sequence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9:56–73, 1987.

[81] L. Silva, A. Sampaio, and E. Barros. A constructive approach to hardware/software partitioning. *Formal Methods in System Design*, 24(1):45–90, 2004.

[82] D. B. Skillicorn. *Foundations of Parallel Programming*. Cambridge Series in Parallel Computation 6. Cambridge University Press.

[83] P. Soderquist and M. Leeser. Division and square root: choosing the right implementation. *Micro*, 17:56–66, July-August 1997.

[84] M.-T. Sun and K.-M. Yang. A flexible VLSI architecture for full-search block-matching motion-vector estimation. In *Proceedings of the IEEE International Symposium on Circuits and Systems*, volume 1, pages 179–182, 8-11 May 1989.

[85] D. Talla, R. Austen, D. Brier, C.Y. Hung, D. Huynh, D. Smith, B. Xiong, D. Talluri, and F. Brill. TMS320DM310 - a portable digital media processor. In *Proceedings of IEEE HOT Chips*, Stanford University, August 2003.

[86] D. Talla. Architectural techniques to accelerate multimedia applications on general-purpose processors. In *Department of Electrical and Computer Engineering, The University of Texas*. Austin, Texas, August 2001.

[87] A. Tchernykh, A. Stepanov, A. Rodríguez, and I. Scherson. Parallel computation in abstract network machine. *Revista Iberoamericana de Investigación, Computación y Sistemas*, IV(4):143–157, 2000.

[88] R. Tessier and W. Burleson. Reconfigurable computing for digital signal processing: A survey. *Programmable Digital Signal Processors*, October 2000.

[89] C. Tomasi and T. Kanade. Detection and tracking of point features. CMU-CS-91-123, Carnegie Mellon University, April 1991.

[90] A.M. Tyrrell. The third NASA/DoD workshop on evolvable hardware. In *IEEE Transactions on Evolutionary Computation*, pages 631–633, December 2001.

[91] R. R. Vemuri and R. E. Harr. Configurable computing: technology and applications. *Computer*, 33:39–40, April 2000.

[92] L. Vos and M. Stegherr. Parameterizable VLSI architectures for the full-search block-matching algorithm. In *IEEE Transactions on Circuits and Systems*, pages 1309–1316, October 1999.

[93] J. E. Vuillemin, P. Bertin, D. Roncin, M. Shand, H.H. Touati, and P. Boucard. Programmable active memories: reconfigurable systems come of age. In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, volume 4, pages 56–69, March 1996.

[94] J. Woodfill and B. Von Herzen. Real-time stereo vision on the PARTS reconfigurable computer. In *The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 201–210, 1997.

[95] G. Xu and Z. Zhang. Epipolar geometry in stereo, motion and object recognition: A unified approach. In Springer, editor, *Computational Imaging and Vision*, volume 6. 1996.

[96] X. Xu and S. Negahdaripour. Vision-based motion sensing from underwater navigation and mosaicing of ocean floor images. In *Proceedings of the MTS/IEEE OCEANS*, volume 2, pages 1412–1417, 1997.

[97] K.-M. Yang, H. Fujiwara, Y. Ishida, M. Maruyama, T. Sakaguchi, and H. Uwabu. A flexible motion-vector estimation chip for real-time video codecs. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 17.5/1–17.5/4, 13-16 May 1990.

[98] K.-M. Yang, M.-T. Sun, and L. Wu. A family of VLSI designs for the motion compensation block-matching algorithm. *IEEE Transactions on Circuits and Systems*, pages 1317–1325, October 1989.

[99] R. Zabih, J. Woodfill, and M. Withgott. A real-time system for automatically annotating unstructured image sequences. In *International Conference on Systems, Man and Cybernetics*, volume 2, pages 345–350, 1993.

[100] Z. Zhang, R. Deriche, O. D. Faugeras, and Q.-T. Luong. A robust technique for matching two uncalibrated images through the recovery of the unknown epipolar geometry. *Artificial Intelligence*, 78(1-2):87–119, 1995.

[101] Z. Zhang. Determining the epipolar geometry and its uncertainty: A review. *International Journal of Computer Vision*, 27(2):161–198, 1998.