FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Stereoscopic Hand-Detection System based on FPGA

Pedro Miguel Salgueiro dos Santos

Master's Thesis submitted for the
Mestrado Integrado em Engenharia Electrotécnica e de Computadores,
Major Telecomunicações

Supervisor: Prof. João Canas Ferreira

June 2009

U.PORTO

FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

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

A Dissertação intitulada

"SISTEMA BASEADO EM FPGA PARA DETECÇÃO DA POSIÇÃO DE UMA MÃO A PARTIR DE IMAGENS ESTÉREO"

foi aprovada em provas realizadas em 20/Julho/2009

o júri [illegible]

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

[illegible]

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

[illegible]

Professor Doutor João Paulo de Castro Canas Ferreira
Professor Auxiliar do Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto

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

Pedro Miguel Salgueiro dos Santos

Autor - **PEDRO MIGUEL SALGUEIRO SANTOS**

Faculdade de Engenharia da Universidade do Porto

Abstract

Electronic devices are more ubiquitous than ever: hand-held devices, portable computers and vigilance systems are a few examples. Human-computer interaction is becoming an important field of research, and the development of systems that provide a more natural and intuitive dialog between man and machine is the key target.

In that sense, we propose to take further the idea of *FingerMouse* (Hebb, J.; Thomas, K.; Kuonen, S.: “VLSI Implementation of the FingerMouse Algorithm”. Swiss Federal Institute of Technology, 2004/2005.): a system that automatically detects your hand. Extreme ease of interfacing with the machine is achieved if your hand is the mouse itself, turning intermediate devices useless. It is a device to be worn on the user’s chest, and its design has strict requirements of weight, compactness and power consumption.

The system locates the hand using the visual information provided by two cameras, disposed side by side, that capture a scene from two slightly shifted perspectives. This stereoscopic setup allows, with the use of appropriate algorithms, to compute a depth map, and extract focal planes of the scene. If a object is known to exist in one of those focal plane, the object can be isolated in this manner. Thus, stereoscopy allows for object detection, based on the object’s distance to the cameras. In this particular project, the goal is to use stereoscopy to stand out the hand and compute the coordinates of its center of gravity.

The used algorithm is computationally expensive, but it can highly benefit from parallelism. An FPGA was considered as the ideal platform for developing a hardware implementation of the system, due to its reconfiguration and high parallelism capabilities.

The development platform was a Xilinx Virtex 2-Pro XC2VP30, operating at 100 MHz. The developed system computes four dense disparity maps, two having the right image as the reference image, and two having the left image as the reference one. Of those two, one uses SAD as the cost-function, and the other one CENSUS. The system’s current operation is done upon images of 640×480 pixels, at a maximum rate of 40 fps (although limited to 25 by the cameras update frequency), reaching a maximum disparity of 135 and using windows of 3×3. Hand coordinates are sent to an external device through the serial RS232 port. The proposed architecture is quite flexible, as it allows easy scalability, namely in disparity range and window size, and trade-off of operating conditions - the system could as well operate at 160 fps, over 320×240 pixels images and a maximum disparity of 55. Moreover, it was designed having in mind a memory limited environment: it only requires, at a minimum, memory enough to store 4 lines of each image. Finally, a new post-processing stage, based on image coarsing, that robustly enhances areas of interest and discards noisy zones, is presented. It can be used by any system that has no need for extremely sharp visual representation of the target object.

Resumo

Os aparelhos electrónicos são, cada vez mais, uma parte importante da nossa vida quotidiana; PDAs, computadores portáteis, sistemas de vigilância são alguns exemplos. Como tal, novos meios de interacção entre máquinas e pessoas têm vindo a ser desenvolvidos.

Neste âmbito, o conceito de *FingerMouse* é apresentado (Hebb, J.; Thomas, K.; Kuonen, S.: "VLSI Implementation of the FingerMouse Algorithm". Swiss Federal Institute of Technology, 2004/2005.). Trata-se de um sistema que detecta automaticamente a mão do utilizador. Como não é necessário qualquer dispositivo intermédio, pois o sistema localiza e segue a mão directamente, a interacção homem-máquina torna-se mais intuitiva. Contudo, o dispositivo é concebido para ser usado ao peito, o que coloca restrições importantes de peso, tamanho e consumo de energia.

O sistema localiza a mão através da informação fornecida por duas câmaras, dispostas lado a lado, que capturam a cena de duas perspectivas ligeiramente desviadas. Esta configuração estereoscópica permite, com recurso aos algoritmos adequados, calcular a distância às câmaras dos pontos físicos da área abarcada. Dessa forma, é possível calcular um mapa de profundidade da cena, e identificar planos focais. Se existir um único objecto nesse plano focal, o objecto pode ser individualizado. Como tal, a estereoscopia permite segmentar uma imagem em objectos, baseando-se na distância destes às câmaras. Neste projecto, o objectivo é usar esta técnica para individualizar a mão, e posteriormente calcular as coordenadas do seu centro de gravidade.

O algoritmo utilizado é computacionalmente bastante exigente, mas também bastante predisposto a paralelização. Nestas condições, uma FPGA apresenta as características ideais para suportar o desenvolvimento duma implementação física deste sistema, graças à sua capacidade de reconfiguração e ao elevado paralelismo que permite.

A plataforma utilizada foi uma Xilinx Virtex 2-Pro XC2VP30, que opera a uma frequência de 100 MHz. O sistema desenvolvido usa o algoritmo de procura de blocos semelhantes para calcular quatro mapas de disparidade densos: dois usando a imagem direita como referência, e os outros dois usando a esquerda como referência. Deste conjunto de dois, um usa como métrica a Soma das Diferenças Absolutas, e o outro CENSUS. O sistema opera neste momento sobre imagens de 640×480 pixéis, a uma taxa de 40 fps (neste momento limitada a 25 por causa da taxa de refrescamento das câmaras), atingindo uma disparidade máxima de 135 e usando uma janela de 3×3 pixéis. As coordenadas da mão são então enviadas por porta série RS232 para um aparelho externo. A arquitectura apresentada é bastante flexível, apresentando fácil escalabilidade, em particular no que toca à disparidade máxima e ao tamanho da janela, e também a capacidade de fazer compromissos entre características do sistema - por exemplo, o sistema podia igualmente operar a 160 fps, sobre imagens de 320×240 pixéis, atingindo uma disparidade máxima de 55. Além disso, foi desenvolvido tendo em mente plataformas parcas em recursos de memória: basta que seja possível guardar 4 linhas de cada imagem. Um novo estágio de pós-processamento é também apresentado, baseado no cálculo duma versão mais grosseira da representação inicial. Tal abordagem permite melhorar áreas de interesse e descartar zonas de ruído, e pode ser utilizado por qualquer sistema que não exija uma representação visual muito bem definida do objecto-alvo.

Acknowledgements

To my colleagues and friends Carlos Resende, João Pereira, João Rodrigues, João Santos and Nuno Pinto, for their inputs and support.

To professor José Carlos Alves and Pedro Alves, for the help and support in making the PCB for the cameras.

To my supervisor João Canas Ferreira, for the support and confidence.

And most of all, to my family.

Contents

List of Figures

List of Tables

Abbreviations and Symbols

FPGA	Field Programmable Gate Array
HCI	Human-Computer Interface
LR	Left image as reference, right image as candidate
PCB	Printed Circuit Board
RL	Right image as reference, left image as candidate
SAD	Sum of Absolute Differences
XUP	Xilinx University Program

Chapter 1

Introduction

This chapter starts by presenting the context under which this project is being developed. Afterwards, the system is briefly described, and some key features of the system are addressed. On the end of the chapter, the structure of this document is presented.

1.1 Context

Human-machine interaction has always been an area of the highest interest since computers came around. Take a simple example: the mouse. Hardly will anyone nowadays interact with a computer, and give up on the tremendous ease of interfacing a mouse provides. Such a simple device, that we now see as a common part of our lives, made a lot in bringing men and machine closer. And people do not expect this interaction to end up with the mouse. New ways of interaction between humans and computers are continuously being proposed and discussed.

Just to mention a few pratical examples, consider Nintendo's Wiimote and Microsoft's recent Project Natal. The first one makes use of a remote controller capable of detecting the user's movements and emulating them virtually, whereas the latter does not even require the user to bear any kind of equipment: the user's gestures (and even faces) are recognized solely by image treatment, acquired from two cameras. These two projects are the proof that human-computer interaction does not need to be restricted to the keyboard or the mouse, or other controller devices. New ideas are coming out of the research stage that will yield a much more natural and intuitive dialog between man and computer, getting people's interest and even showing themselves economically profitable.

This project aims at giving one more contribute to this technology area. The system proposed here is expected to determine the position of a hand, by relying on information given by two cameras. Much information can be transmitted from the user to the computer in this manner, and intermediate devices, such as mice or pointing devices, turn useless.

The most illustrative example of application for this concept is, perhaps, a presentation. While performing an presentation based on slides, one has often the need to emphasize some elements, like pictures or sentences. Remote pointing devices, such as small laser pointers, become useless if all you need is your hand. Moreover, one can think further ahead and propose the system to identify the hand's posture, thus associating each gesture with an action of the system (e.g., passing to the next slide), and therefore offering the user a much wider range of functionalities. Other applications are "virtual painting" (making drawings by just moving your hand), or virtual reality, where the position of your hands is a relevant issue.

1.2 System Description

1.2.1 Project Objectives

The system's objective is to compute in real-time the spatial coordinates of a hand, as seen by the perspective of a set of two cameras, and transmit that information to a computer, where the mouse cursor will be moved accordingly.

The main difficulty about achieving the proposed objective is to ensure that, with the present technology and available resources, the information provided by the system is acceptable by human standards. This means that the cursor's movement should be smooth and follow the hand avoiding jerky movements (jumping quickly and randomly from one place to another).

However, how true is the cursor's movement is a rather subjective issue. No plain metric exists to assess this performance, and therefore there is not a set of clear requirements to be meet; the project is to evolve according to what, in each stage, seems to be the most effective approach to better the system. But there is at least one requirement that is most relevant: the ability to work real-time.

1.2.2 Approaching the problem

The system's core task is to individualize the hand from the rest of the objects in the scene. Different methods exist for achieving that goal, and they will be presented and discussed in the next chapter 2. For now, and for the sake of a brief system description, it will be said that the method on which the system relies is to compute the objects' distance to the cameras, and **assume** the closest object is the hand.

The main problem here is the uncertainty associated to the distance-computing algorithms. One should expect the hand not to be perfectly segmented, or to have spurious areas in the foreground. The objective of enhancing those results is the purpose of a considerable part of the system.

1.2.3 Algorithm Description

The algorithm's several stages are hence the following:

1. Receive data from the cameras;

2. Compute the distance of the objects to the cameras;

3. Individualize the closest object of the scene (ideally the hand);
4. Implement methods to improve robustness, overcoming possible errors or oversights of the previous step;
5. Compute the hand's coordinates;
6. Deliver them to the computer, which will move the cursor accordingly.

1.2.4 Distinctive features

Distance-computing algorithms are an important area in the overall field of image algorithms. Step 2 of the previous list will make use of widely-used, widely-tested, well-known algorithms; nevertheless, some post-processing remains to be done, in order to better those algorithms' results and adapt them to this project specificities.

It is on this post-processing stage that this project expects to give some contribution. One assumption is made for better tackling the issue: there is no need to sharpen the hand's visual representation, considering the expected final output of the system are the hand's coordinates, from to the cameras' perspective. This means that the visual representation can be severely distorted, as long as the hand's location is still correct.

This fundaments the new approach this project presents: to compute a more coarse-grained representation of the result. This will turn the whole system less susceptible to noise, and enhance the areas of interest, while giving room for more complex system-enhancement features. Also, effort has been put into not depending of color information, in order to turn the system more generalizable.

1.3 Document Structure

The remainder of this document has the following structure.

In chapter 2, the necessary background is presented, in order to understand the following chapters. Parallelly, the state-of-the-art of the system's fundamental aspects is discussed.

In chapter 3, the concepts presented in the previous chapter are brought together under the scope of this project specific needs. Also, the physical system setup is addressed, as well as the project flow.

In chapter 4, the hardware structure of the system is presented. Architecture options are discussed here, in close relation with the system's internal operation.

In chapter 5, the results are presented. Comments are made on those, and also on the system's real-time operation. The FPGA resource usage is also discussed in this chapter.

Finally, in chapter 6, some ideas that could complement or extensively enhance the system's quality in future works are addressed. Also, a closure is made on this project description.

Chapter 2

Technical Background

In this chapter, the background of this project is presented. Firstly, the specific work that served as a basis for this project is presented. Afterwards, an introduction to the project's fundamental theory is given, namely stereoscopy and area-match algorithms. Afterwards, reconfigurable systems are discussed, and implementations of stereoscopic image processing algorithms on reconfigurable systems are the last subject of this chapter.

2.1 FingerMouse

2.1.1 Introduction to FingerMouse

This project is very loosely based on the concept developed at ETH Zurich (the Swiss Federal Institute of Technology pole at Zurich) Department of Information Technology and Electrical Engineering, for some years until now. FingerMouse aims to be a Human Computer Interface (HCI) Device, helping the user to interact with a computer. The underlying idea is to be able to use your hand (or finger) as a mouse; the FingerMouse system becomes aware of the direction you are pointing at, providing extreme ease of interface with the computer

Having this in mind, the ETH researchers have come up with an interesting concept: to have a little device placed on the user's chest, that can capture the location of the user's hand while it is moving and transmit that information to the computer. Figure 2.1 depicts the concept.

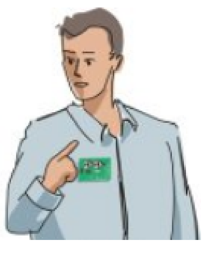


Figure 2.1: The *FingerMouse* concept. Image drawn from [1].

However, such implementation option is not free of problems, and an important requirement strikes us right from the start: an HCI cannot obstruct the user in any way, by compromising the user's liberty of movements or behaviour. In some sense, the user must not even be aware of its existence. This means it should be lightweight, small, of low power consumption, and easy to plugin and start to use. Therefore, the ETH researcers always looked for hardware platforms that would yield a good compromisse between computation capabilities and physical characteristics, like weight and size.

2.1.2 How to implement

How is this objective to be achieved? How is one able to locate a hand in a scene? As said before, the FingerMouse project has been active for a few years, and the ETH researchers have studied different approaches. The table found in the article [1], authored by the ETH authors, lists some options, as well their advantages and disadvantages:

1. Static background subtraction (Temporal subtraction)
2. Color segmentation
3. Active lighting
4. Structured lighting
5. Time of flight camera
6. Contour tracking
7. Stereo vision image subtraction
8. Stereo vision depth mapping

Some ideas are relatively simple. The idea pointed as number one, Static Background Subtraction, concerns temporal analysis. Assuming a video camera with a sampling rate of x frames per second, one can compare the image of instant *(t)* with the image of instant *(t-1)* and extract information about the *moving* object's shape and momentum. Figure 2.2 depicts the situation.

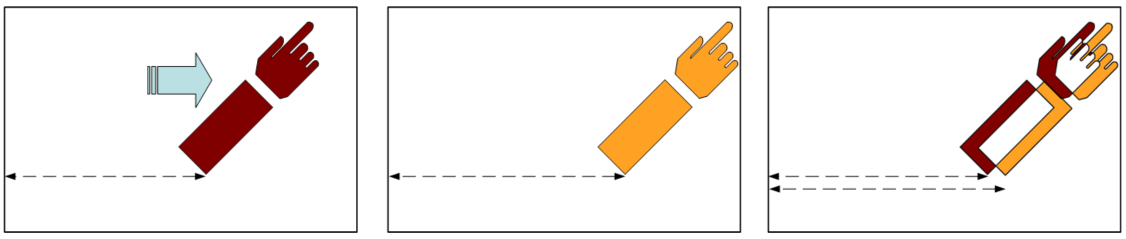


Figure 2.2: Left image shows object in instant *t=0*. Center image shows object in instant *t=2*. Right image shows which pixels were covered or uncovered.

This approach, nevertheless, suffers from a great shortcoming: the hand would have to be the only moving element in the picture. This is not acceptable, if the device is to be carried in the user's chest. Temporal comparison is discussed further in [2].

Another relatively simple idea makes use of not one, but two cameras. It is the case of item number 7, Stereo Vision Image Subtraction. It consists of subtracting the images received from both cameras, in the same temporal frame. The underlying assumption is that, if only the background is present and provided it is far enough, the image captured by both cameras will be similar, thus yielding the remainder of the subtraction null. However, if the hand is in the cameras' range of view, it will be seen in different locations by each cameras due to being very close. Subtracting the images will yield a difference that allows to locate the hand.

This was in fact the approach chosen for one of the initial FingerMouse projects. The use of a DSP processor met, in one hand, the physical requirements of the system (light and compact), but on the other hand did not allow for computationally expensive algorithms. The full system was composed of a PCB featuring the two cameras, the DSP processor and a distance sensor. More information on this implementation can be found in [3].

Other approachs were analyzed. The ETH researchers tried an approach related to Laser Triangulation, as a mean of obtaining a 3D model of the hand (it corresponds to item number 4, Structured Lightning). Their conclusions are presented in [4]. Also color segmentation was discussed, but the use of that method alone would not yield the necessary robustness to the system. It is used in FingerMouse 6, for confirmation of the main algorithm output. Object recognition was also disregarded from the beginning due to the high computational requirements it implies. Nevertheless, a very simple Contour Traking (number 6) was implemented in one of the FingerMouse project versions [3].

Many options have therefore have been considered. FingerMouse 6, the latest version, makes use of Stereo Vision Depth Mapping, number 8 on the list of possible approaches. It is based on computing the distance of the objects on the scene to the cameras, and in that way differentiate them. Theoretically, it is expected to be a rather powerful technique for the purpose in mind, but the high computational effort this algorithm requires restricts the range of hardware platforms that can handle the amount of processing needed. In fact, the ETH researchers considered an ASIC implementation to be the only one that could respect the above mentioned requirements: lightweightness, compactness, and no obstruction to the user.

The system FingerMouse 6 is described in detail in [5], a master thesis authored by Hebb, Koch and Kuonen (for convenience, this document will be refered from now on as the "ETH Report"). This project proposes using the same algorithm, stereo vision depth extraction, but to be implemented in a FPGA. Stereo vision depth mapping makes use of the special features of stereoscopic setups, which will be discussed next.

2.2 Stereoscopy

2.2.1 Introduction to stereoscopy

Stereoscopy is a simple solution for distance computation, commonly designated by depth computation (with respect to the cameras). It is in fact a method that nature uses recorrently, as many animals are two-eyed. It is a simple, efficient and low energy consuming mechanism for determining objects' distance.

It consists of having *two* images (or eyes, or cameras) set side by side (and, in some setups, focusing the same point). If the distance between the cameras is known, and an object is present in both images, then it is possible to approximately compute the distance of the object to the straight line that connects both cameras. Figure 2.3 depicts a stereoscopic setup seen from above, and figure 2.4 presents the scene as viewed by both cameras.

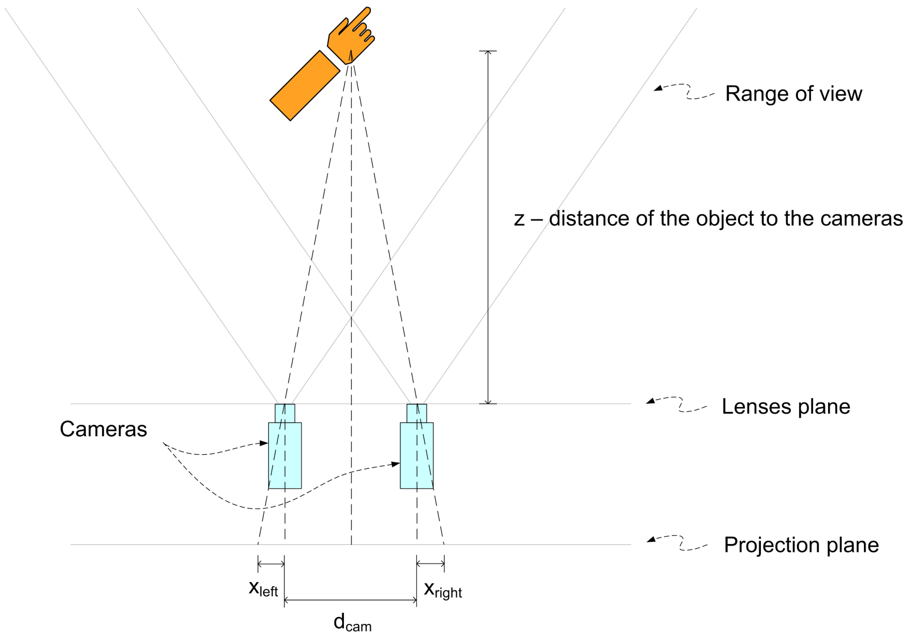


Figure 2.3: A stereoscopic setup. z refers to the object's distance to the cameras.

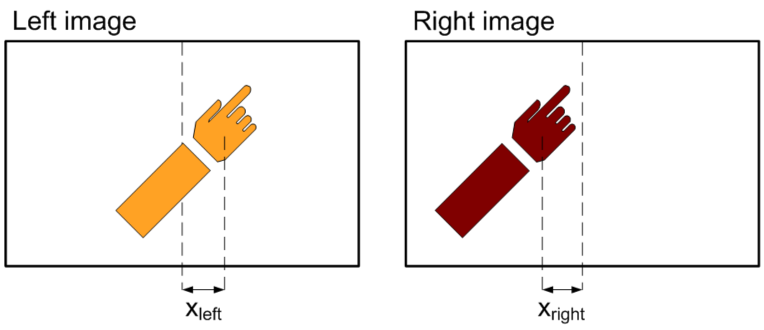


Figure 2.4: The object as viewed by both cameras. Left camera perspective on the left, right on the right.

The calculation of the distance of the objects to the cameras is based on simple euclidian mathematics, thanks to the trignometric features of a stereoscopic setup. Observe figure 2.5. On the left scheme, triangle OL_lL_r is equivalent to the sum of triangles $L_lP_lP_{cl}$ and $L_rP_rP_{cr}$. d_{cam} is the distance between the cameras, z the distance from the cameras to the object, f the focal distance, and x_{left} and x_{right} the distance of the focal projection of the object to the focal center.

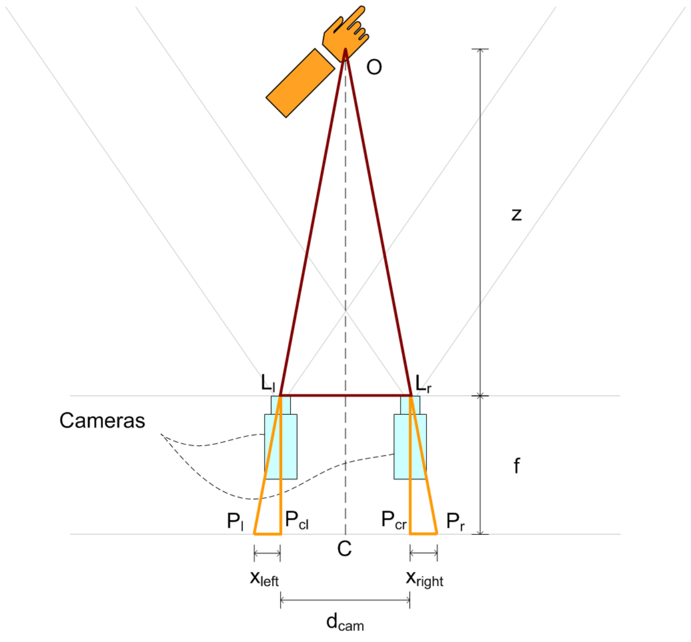


Figure 2.5: Trignometric characteristics of stereoscopic setups.

The following equations holds:

$$\frac{f}{z} = \frac{x_{left} + x_{right}}{d_{cam}} \tag{2.1}$$

For clarification issues, f, the focal distance, is a characteristic of any camera. Figure 2.6 depicts this concept.

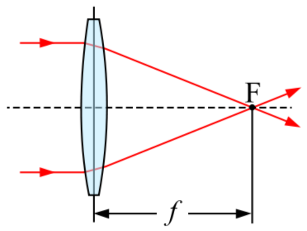


Figure 2.6: Focal distance. Image drawn from [6].

In order to compute the distance, it is of relevance to introduce one important concept: the one of *disparity*. Figure 2.7 presents again the scene as viewed by both cameras, and a third figure of the two previous ones overlapping. Consider the right image, and pick a point. Now search the left image for the same point. You will find the same point is now closer to the right border of the image. The distance between the original position and the shifted one is called *disparity*.

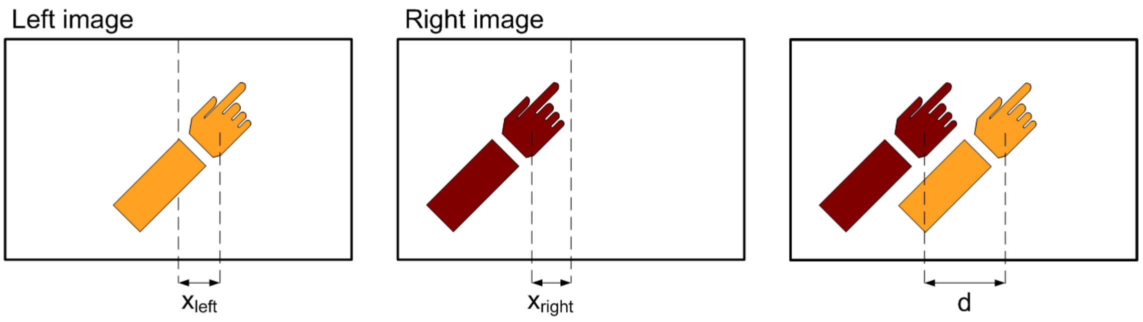


Figure 2.7: Disparity on two overlapping images belonging to a stereoscopic set.

The main characteristic about disparity is that it is inversely proportional to the object's distance to the cameras. The further the object, the smaller the disparity. To better depict this fact, take a look at a well-known set of stereoscopic pictures - Tsukuba's. Take for example the left tip of the box where the head lays. Considering an horizontal axis starting on the leftmost edge of the image, one will notice that tip is not in same position in both images. It has a significant disparity. A point in the background, however, does have a smaller disparity value.

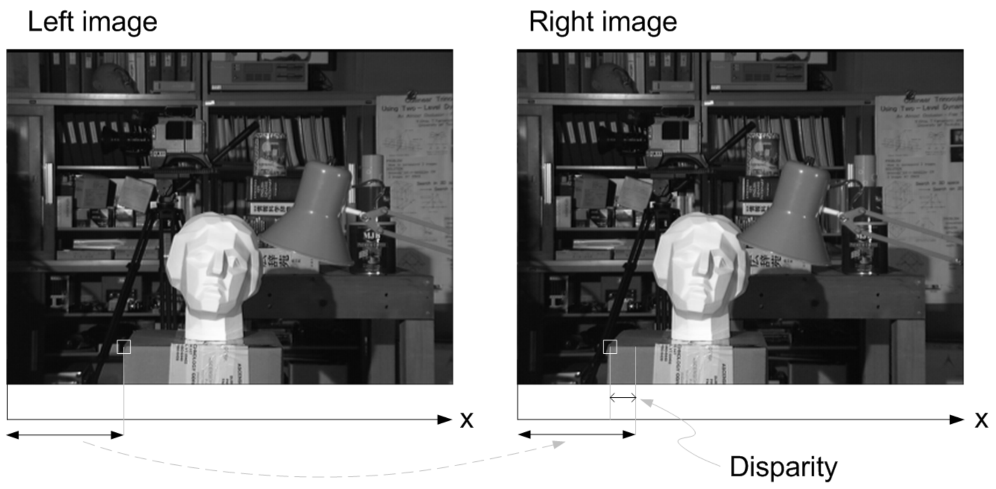


Figure 2.8: Stereoscopic pair of images and the concept of disparity. Drawn from [7].

The computation of a disparity value for *all* the pixels on one of the image allows the creation of **dense disparity maps**. It is an image where the intensity of each pixel reflects the disparity of the pixel that occupies that same position. Figure 2.9 presents the outcome of such operation. It is based on the right image of figure 2.8; almost all physical points in it can be found in the left image, and therefore each one has an associated disparity. The disparity map just translates those disparity values into a visual representation, by attributing high intensity values to points with high disparity, and low intensity values to areas with small disparity.

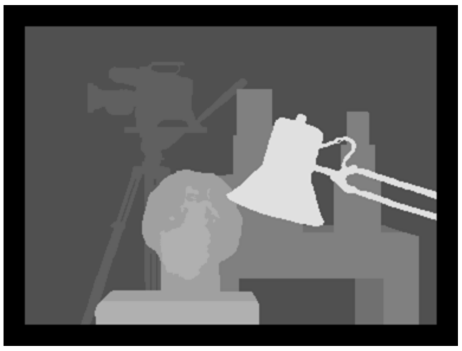


Figure 2.9: Disparity map for the Tsukuba left image. Drawn from [7].

Probably, the most striking feature of figure 2.9 is how related disparity and distance are. However, this is not a true disparity map. Normally, the same object would have different disparity values, although all of them within a certain range. One will notice, though, that in this image the objects' luminosity values are very homogeneous, meaning this image suffered a thresholding operation: pixels with luminosity (or similarly, disparity) within a upper and a lower bound were normalized to one single value, thus giving areas with similar disparity its homogeneous look. One could say the image's focal planes were enhanced.

This helps to introduce one of the basic fundaments of this work: the capability of segmenting the objects in one image through focal differentiation. If it is known that an object of interest has a disparity within a certain range (or, similarly, it is at known distance of the cameras), the image can be segmented into focal planes, through the aforementioned thresholding operation, and the object's *shape* can be retrieved. Beware, though, that caution is needed. The focal plane of the desired object has to be known beforehand. Moreover, focal plane segmentation is not object segmentation, meaning there can be more than one object in each focal plane. Nevertheless, this is still a segmentation method considerably less demanding, in computational terms, than most object recongnition methods.

Stereo depth mapping, followed by a thresholding operation, is the basic idea underlying this project, as conceived by the ETH researchers. The assumption is that the hand can be individualized this way, and in doing that its center of gravity can be computed. It will be seen next which algorithms can be used to perform depth extraction.

2.2.2 Stereoscopic image algorithms

Having discussed the particular characteristics of stereoscopic setups, one addresses now the image algorithms developed to make use of such features. Stereo-matching algorithms will be presented and given a concise review from literature.

A very detailed but concise listing and description of "dense, two-frame stereo methods" can be found in [8]. The work of Scharstein and Szeliski provides an impressive overview over most of these algorithms, and emphasizes the algorithms' capacity of obtaining good quality disparity maps. The authors enumerate two main categories of methods, global and local, and list a set of four fundamental operations. All the methods perform at least a subset of those operations.

According to document [8], global methods operation is concisely described as “solve an optimization problem”. The variety and complexity of the algorithms makes them unsuited for a more detailed explanation, and furthermore for implementation in this system. To support this option, article [9] states that global methods “produce very accurate results but are very time and computational demanding due to their iterative nature”. Local methods, on the other hand, account normally for a less computationally expensive implementations. According again to [9], “Local methods are usually fast.” Local methods are also commonly called area-match algorithms.

Parallelly, a good overview of the fundamentals of computational stereo operation can be found in [10]. The document puts forth two ways of computing dense disparity maps: feature (sparse) matching or area (dense) matching. On what concerns the first option, the same document [10] lists the two main types of features that are commonly matched: “semantic features (with known physical properties and/or spatial geometry)” or “intensity anomaly features (isolated anomalous intensity patterns not necessarily having any physical significance)” [10].

The latter case, within the scope of our project, would mean to have the hand to be heavily enlighted, or to wear a uniformously colored glove (white, for example) so that it would turn a homogeneously colored area in both images. Afterwards, an algorithm would search a wide intensity-homogeneous area in both images, compute its center of gravity and calculate the area’s disparity. This approach, although theoretically not very hard to implement on a hardware description level, would steal some generalization to the system, by constraining it to very specific environments. The first option, “semantic features (with known physical properties and/or spatial geometry)”, brings us back to the field of, if not Object Recognition, at least Feature Recognition. It requires having a database for feature matching in order to locate the desired objects. In some sense, it requires segmenting the image into objects even before performing the disparity computation, therefore subverting the reason why stereoscopy was chosen in the first place.

Area-match algorithms, the other option for stereo-matching, present simpler implementations, at the cost of an high computational effort. They allow for the generation of what is called “dense disparity maps”, similar to the one in figure 2.9. The word “dense” emphasizes the fact that each pixel of the stereo pair of images is fully processed by a rather “feature-indifferent” algorithm, as opposed to looking for specific groups of pixels in both images, like in the feature match operation. The simple but highly parallellizable operation of area-match algorithm makes it an ideal candidate for implementation in high-parallellism platforms.

The first step of area-match algorithms is to pick one matrix of pixels from one of the stereo pair images. By pixel, one means the *intensity value* (or luminance value) of a pixel, or also chrominance values if color information is available. This matrix represents a physical point in the first image. The algorithm then starts searching for a matrix with similar intensity values in the other image. Several matrixes are tested on this second image, and the one that is considered the most similar is assumed to be the representation of the same physical point, on the second image. The distance between the location of the best matching matrix, in the second image, and the original matrix, in the first image, with respect to the horizontal axis discussed before (starting on the leftmost edge of the images and heading right) is the so called disparity. Normally, although the full matrix is compared, the disparity is only associated to the center pixel. Figure 2.10 depicts the basic idea.

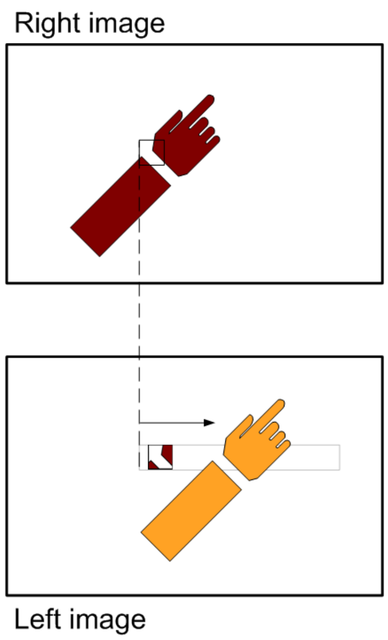


Figure 2.10: The left image is being scanned for a matrix drawn from the right image.

Again, because there is no *a priori* information of where might the interesting objects be, *all* the pixels from the initial image must scanned into the second image. On the end, each pixel will have an associated disparity. This is inherently a highly parallellizable operation. The algorithm only needs a very specific amount of data and performs a well time-defined operation. Area-match algorithms will now be discussed in more detail.

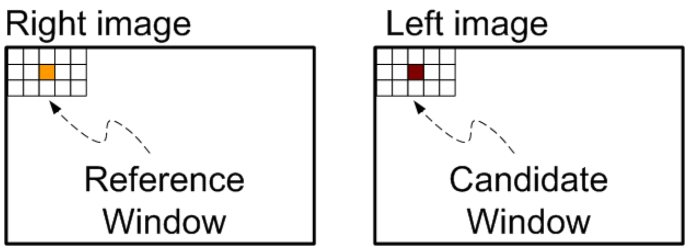
2.2.3 Area-match algorithms

Area-match algorithms can normally be divided into two components: the common framework that allows every pixel from the initial image (commonly called the *reference image*, as it is providing the matrices for search) to be scanned into the second image (or *candidate*); and the metric, or cost-function, that is used to assess the "similarity" between two matrices. An introduction to area-match algorithms can be found in [11].

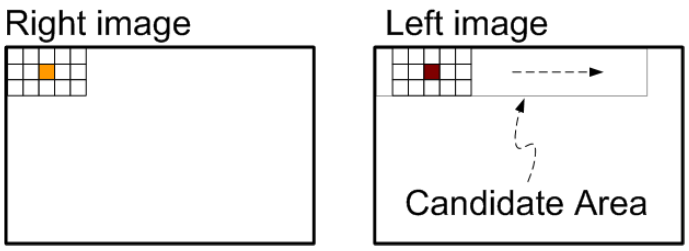
Image sweep

The methodology for an exhaustive application of the framework is now described, as well the associated nomenclature: in order to compute the disparity for the first pixel of the top left edge of the right image, pick the matrix of 3 pixels wide for 3 pixels tall that surrounds it. This will be the *reference window*. In the left image, a matrix in the same horizontal and vertical position is also considered: it is the *candidate window*. Both matrices are tested for “similarity”, and a quantitative value for how similar they are is computed. This is presented in figure 2.11(a).

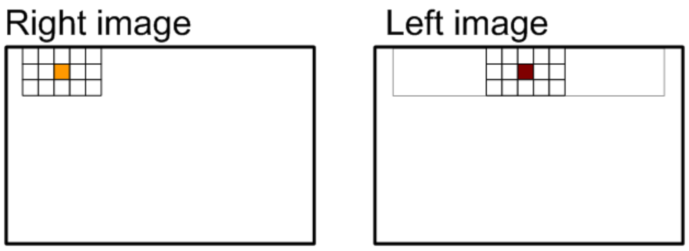
Afterwards, the candidate window is only displaced by one single pixel. Figure 2.11(b) depicts the new situation. Again, the new candidate matrix undergoes the same comparison operation with the reference matrix, and a new value is computed.



(a) Reference and candidate windows are in the initial position.



(b) Reference window is inthe initial position; candidate window is in a random position.



(c) Reference and candidate windows are in random positions.

Figure 2.11: As the reference window shifts over the reference area, a whole new candidate area has to be swept. Shifts are of one pixel each.

This process is repeated until several candidate matrices have been tested. All the area that is being tested for similarity is called the *candidate area*, pictured in grey in figure 2.11(b). When the last matrix is tested, the similarity values are compared, and the best one will indicate the candidate matrix which resembles the reference window the most. The difference between the positions of the reference window and the best-match candidate window is the disparity (the same concept which was introduced in the later section) for that particular pixel.

At this point, it is time for the reference window to be shifted for one pixel. The process starts all over again. (In figure 2.11(c), the candidate window is at a random position.) All things considered, talking in Computer Science terms, it is very similar to a `for` cycle nested within another `for` cycle.

Finally, note that the operation can be performed either having the right image as the reference image, and the left image as the candidate image, or vice-versa. The above example was made according to the first situation; in the second, it would suffice to perform the scan (or sweep) from right to left. In order to clarify matters from the beginning, the following nomenclature will be used from now on: **LR sweep** stands for the version of the framework that has the left image as the reference one, searching on the right image; otherwise, it will be **RL sweep**. The first letter of the acronym indicates the reference image, the second letter stands for the candidate image. For example: RL = Right image is the reference one, Left image the candidate.

Cost-functions

Now, a topic requires attention at this point: how does one define “similar”? Although both images capture the same object, they do so from different perspectives, and therefore it is unreasonable to assume two areas, one from each image, will ever match perfectly on luminance and chrominance values. A measure of “similarity” is therefore needed, and it can be given by what is often called a “cost-function” (or “metric”).

Cost-functions are normally based on performing a mathematical (or of other kind) operation over all the pixels of the reference window and the candidate window, from which commonly a single value is produced. Performing this operation on several candidate windows provides a set of “similarity” values that are then compared. The candidate window that yields the best value, according to the cost-function in use, will likely be the most resemblant to the reference window.

Cost-functions are classified according to the type of operation they implement. For this project, two classes are of relevance: parametric and non-parametric. The first group normally relies on some mathematical operation to compare the global luminance (and chrominance, if that is the case) between pixel matrices, while the latter performs a “relative ordering of the pixel intensities” [12] for each window and compares the results, therefore disregarding global properties of the window. A more detailed comparison on the two types of metrics can be found in [12]. Table 2.1, adapted from the same document [12], presents a few examples and their mathematical implementation.

For a quick explanation of the equations, consider the one for SAD. The sum index n is equal to the number of pixels in the windows; moreover, I_1 represents a pixel belonging to the reference window, situated in position (u,v), whereas I_2 simbolyzes a pixel that is a part of the candidate window, situated in the same or another spatial location (hence the x and y addition). From the above, one can infer from the formula that SAD boils down to summing up the absolute differences among the two pixels, one from the reference window and other from the candidate image, that occupy the same position in both windows (but not necesssarily the same position in the image). If another candidate window is used, the window the most similar to the reference window will be the one that produces the smallest SAD, meaning there is lesser difference in the pixels intensity

Sum of Absolute Differences	SAD	$\sum_{k=1}^{n} \lvert I_1(u,v) - I_2(x+u,y+v)\rvert$
Zero mean Sum of Absolute Differences	ZSAD	$\sum_{k=1}^{n} \lvert (I_1(u,v) - \bar{I_1}) - (I_2(x+u,y+v) - \bar{I_2})\rvert$
Sum of Squared Differences	SSD	$\sum_{k=1}^{n} (I_1(u,v) - I_2(x+u,y+v))^2$
Zero mean Sum of Squared Differences	ZSSD	$\sum_{k=1}^{n} ((I_1(u,v) - \bar{I_1}) - (I_2(x+u,y+v) - \bar{I_2}))^2$
Normalized Cross Correlation	NCC	$\dfrac{\sum_{(u,v)\in W} I_1(u,v)\cdot I_2(x+u,y+v))}{\sqrt[2]{\sum_{(u,v)\in W} I_1^2(u,v)\cdot \sum_{(u,v)\in W} I_2^2(x+u,y+v)}}$
CENSUS	-	$\sum_{(u,v)\in W} Hamming\lvert I_1'(u,v), I_2'(x+u,y+v)\rvert$

Table 2.1: Some cost functions for area-matching algorithms. I_1 refers to the pixel of the reference window and I_2 to the one of the candidate window.

values. Figure 2.12 depicts the process. Most of the other functions follow the same reasoning. For NCC, however, the bigger the value obtained, the more likely the resemblance (unlike SAD).

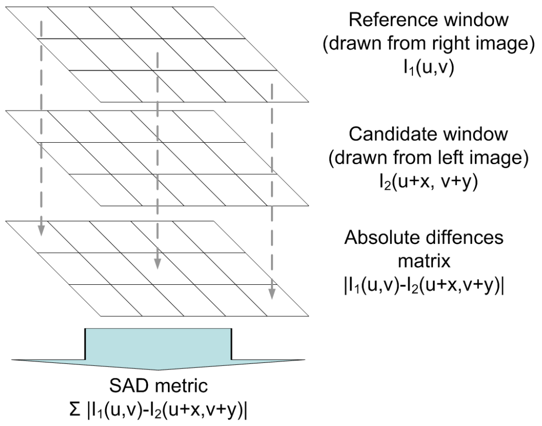


Figure 2.12: The SAD cost-function operation on a 5x3 matrix.

CENSUS stands out of the group for being the only non-parametric metric. The “Hamming” function it uses demands some explanation. The idea underlying the CENSUS metric is to pick the reference matrix, extract the central pixel, and then classify all the other pixels according to if they are larger or smaller than the central one. A 1 or a 0 is attributed for each one, and the set forms a string of bits, commonly named codeword. The same operation is performed on the candidate matrix, and afterwards both codewords are compared by the “Hamming distance”: the number of bits in the two words that are equal provides a measure of “similarity” between the reference and the candidate windows. Different candidate windows will yield different “Hamming distance” values, and the bigger the value the most resemblant the reference and candidate windows should be. Figure 2.13 displays the operation of CENSUS.

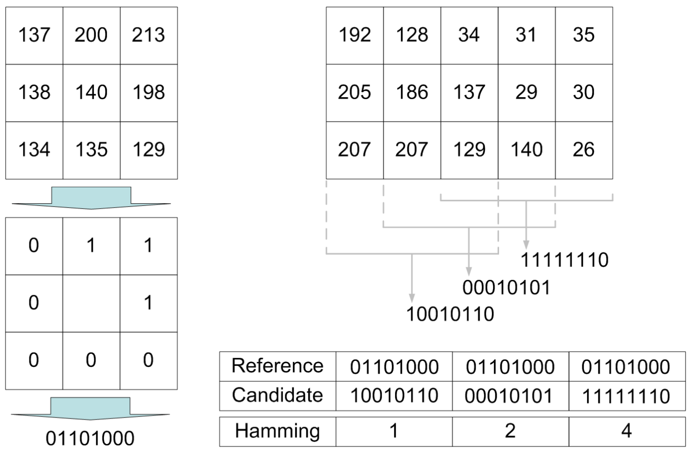


Figure 2.13: CENSUS cost-function operation.

The ZSAD and ZSSD are two particular cost-functions. They appear as enhancements of SAD and SSD, respectively, in order to overcome "radiometric distortion" [12], a phenomenon that occurs when SAD and SSD are applied to a set of images where "the pixel values in one image differ from the pixel values in the other image by a constant offset and/or gain factor".

2.2.4 The occlusion problem

A common problem when using area-match algorithms is the problem of occlusion. When a match is being searched for an area of the reference area, and that area is covered in the candidate window by some object, a incorrect match happens. The imediate consequence is that there will be a set of pixels from the reference image to which a higher disparity value will be attributed, because the best match is only found further in the sweep. Figure 2.14 depicts the situation.

This means that, after performing the thresholding operation, some pixels will likely be considered closer to the cameras than they really are. In order to overcome this problem, a simple but demanding solution exists. It implies performing the area-match algorithm not only having an image as the reference one, and the other as the candidate one, *but also* switching the images role - having the second as the reference one, and the first as the candidate one. Reattaining the already mentioned nomenclature, one will need to perform the RL sweep (right image is the reference one, left image is the candidate image) and LR sweep (mind the acronym - L for left image as reference, R for right as candidate image).

After thresholding, the two results are then matched for eliminating occluded areas, and one single output is produced. But how is one to combine the two results? The explanation follows: pick the pixel in position x from one of the results (take, for now, the RL sweep); it should have the value of either 0 or 1, according to if it was qualified as a foreground (hand) or a background pixel. Also consider its associated disparity, *disp(x)*. The underlying idea is to check if, in the other result (in this case, the LR sweep), the pixel in position *x+disp(x)* was also considered a

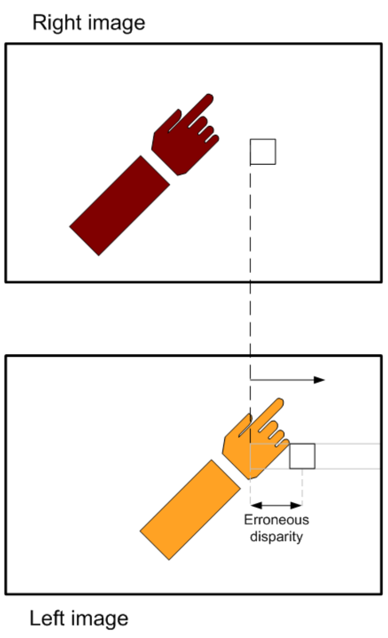


Figure 2.14: Erroneously matched areas due to occlusion.

foreground pixel. If so, the pixel in position x will be qualified as a foreground pixel; if any of the results did not classified the pixel as foreground, the pixel is discarded and considered a background one.

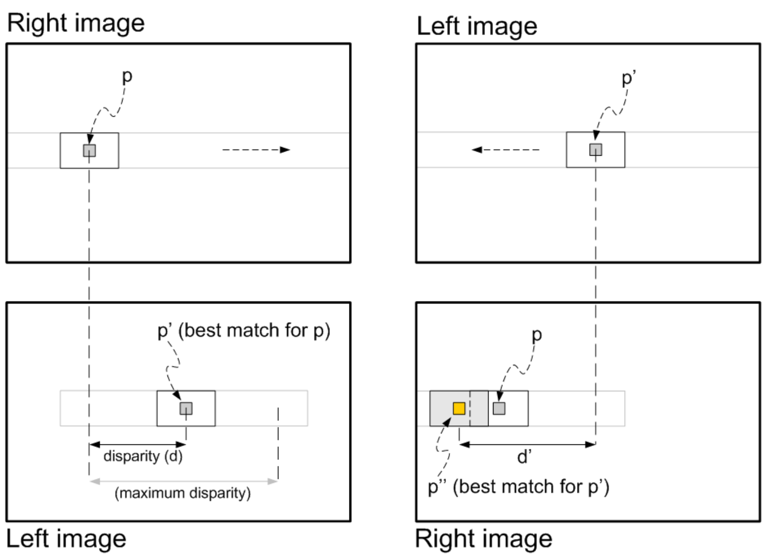


Figure 2.15: Matching the results of right-left scanning and left-right scanning.

Figure 2.15 depicts the operation. The center pixel, p, for a particular position of the reference window, finds its most similar pixel, p', d pixels ahead (ahead means here "more to the right"), while the candidate window is sweeping left. The idea here is to check if p', when being the center pixel of the reference window that is scanning the lines leftwise on the LR sweep, will yield p as its most similar pixel. If so, then the pixel is considered as belonging to the foreground. If not (as it is the case in the figure), the pixel is classified as background.

How does this method helps to resolve the occlusion problem? Figure 2.16 helps to explain the concept. Consider one is using the LR sweep as the pivot image, and performing a consistency check on the RL sweep output. An area of the LR sweep that belongs to the "true" foreground will have associated a disparity that will unveil, in the RL sweep, an area that was also considered foreground. Such is the case of Area 1, in figure 2.16; the blue arrow represents its associated disparity, and if the RL sweep is checked, a "true foreground" part of the hand is to be found. However, if an area was incorrectly considered as foreground by the LR sweep, because the right image did not match the left image due to the occlusion caused by an object, a wrong disparity will be associated to Area 2. Yet, when the RL sweep is checked in the area that the disparity given by the LR sweep announces, no foreground pixels will be found, meaning that area of the LR sweep is in fact an occluded and can be discarded (given a disparity of 0).

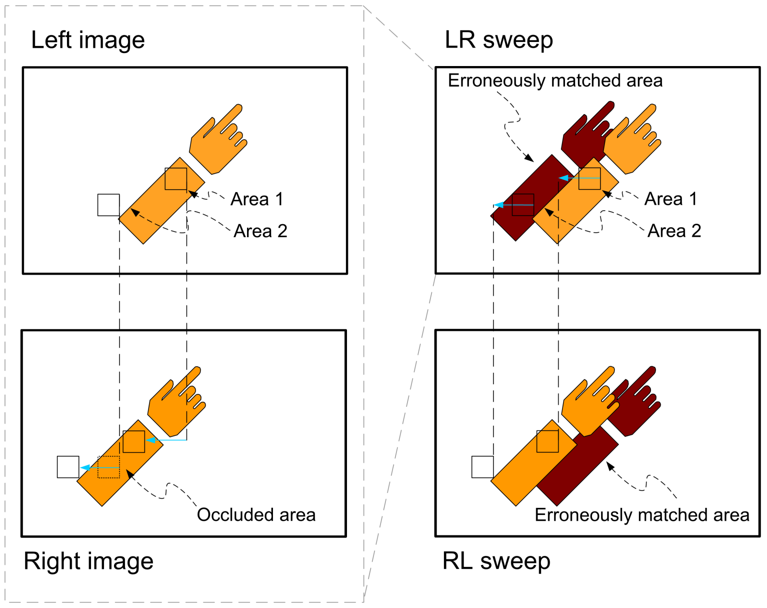


Figure 2.16: Area 1 was correctly considered as foreground; area 2 was it too but incorrectly, due to occlusion. Matching the bi-directional sweeps helps to overcome the problem.

2.3 Reconfigurable Systems

2.3.1 Introduction to reconfigurable systems

The compromisse between software and hardware that reconfigurable systems present is also a key factor in this project. A software implementation could certainly be simpler, but it would not be suited for implementation in a small hardware platform, as the *FingerMouse* concept proposes. An ASIC, thanks to its specificity, would be the most suited platform, but its high cost makes this option unreasonable.

FPGAs (acronym for *Field Programable Gate Array*), however, are reconfigurable systems that allow for a in-between solution, allowing for powerful functionalities. This electronic hardware device consists of a 2D array of tiny processing (and storage) elements that can be interconnected in order to implement an equivalent digital circuit, provided it is within the FPGA capabilities. Re-mapping of the interconnections can happen anytime, and circuits can be dedicated either to processing or storage. Henceforth, FPGAs present excellent characteristics for prototyping, achieving a trade-off between the speed of ASIC-solutions and the flexibility of software solutions.

2.3.2 Stereoscopic image algorithms in reconfigurable systems

The selection of the specific algorithms to implement must always be made having in mind the target platform. Aiming at this objective, articles describing previous implementations of stereoscopic image processing algorithms in FPGAs were studied. However, there is a great diversity of objectives and features in each proposal of implementation.

Some systems use more than one FPGA. The article [13] depicts a powerful system composed of 16 Xilinx 4025 FPGAs, capable of computing 24 stereo disparities on 320 by 240 pixel images at 42 frames per second, by using the CENSUS stereo algorithm. However, the article is dedicated mainly to describing the system; CENSUS is shown as an example application, due to the high parallellism it can benefit from. Moreover, multi-FPGA implementations are off this project's scope.

Other systems aim for more than depth computation. The article [14] describes a hand-sized trinocular system based on a FPGA, that performs a metric similar to SAD on a rate of 120 frames per second for 320×240 sized images, achieving a disparity of 64 pixels, and much attention is given to rectification of images.

Speed is also an issue. A 150 pixels disparity range using SAD is reported to be achieved, for 450×375 pixels images, on a FPGA system capable of attaining 700 frames per second [15]. Yet, the speed-optimized architecture is inadequate for this project's purposes.

A similar system to the one of this project is described in [16]. It is implemented on a Spartan-3, and performs the CENSUS transform on 320×240 pixels images at a rate of 30 (eventually up to 150, with the adequate cameras), on a 7×7 pixels window and achieving a disparity of 20 pixels. However, the larger FPGA that is to be used in this project certainly allows for better specs. In particular, the attained disparity should be considerably larger.

Therefore, frame rate, disparity range, image dimension, used algorithm and window dimension are normally the features to be compared. The system described in this document has characteristics not very different from those presented in many articles; moreover, depth mapping optimization is not the goal of this project. Several tables comprising the characteristics of a few hardware implementations of image algorithms can be found in [17] and [15].

Also articles addressing the general theory of stereo-matching algorithms implementation in FPGAs were analyzed. Some examples are [11] and [9]. But the generic nature of these articles lead to no specific enhancement of the structure presented in this document.

Chapter 3

System Design

This chapter starts by presenting the design flow of the project. Afterwards, the algorithm's overall operation is presented. It will be shown how the concepts presented on the previous chapter 2 will be used to implement the desired system. Finally, the physical system setup is presented, to better contextualize the hardware implementation of the algorithm.

3.1 Design Flow

In order to predict the system's behaviour, the algorithm was fully simulated in software, using C++ language. C++ was chosen against C for having a more "user-friendly" interface when it comes to dealing with files. MATLAB was also considered, but no relevant advantage was found. The type of files used was ".pgm"; with those, each pixel of the image is defined in a 255-level grayscale by a byte (and hence the name "**P**ortable **G**ray **M**ap"). The image's dimensions are given by a header, which also informs of the type of file.

This approach allowed for the test of different cost-functions, parameter variation assessment (like maximum disparity achievable), problem anticipation, data storage and processing structures prevision (having a higher level visualization of the data flow before mapping it into an hardware structure), and validation of the algorithm's Verilog implementation.

Concerning the Verilog language modules, Mentor Graphics' "ModelSim" tool was used to analyze and validate the algorithm before implementing it in the FPGA. It was used not only to verify the signals of the developed hardware modules, but also to output visual results to be compared with the software simulation results.

Upon implementation on the FPGA, the visualization of the outputs in the monitor and the reception of values in the computer through the serial port were used to assess the correct operation of the algorithm.

3.2 Algorithm Description

In this section, the full algorithm is presented. The background knowledge provided in chapter 2 is now reviewed in the scope of this project. Several implementation options are discussed for each stage, and simulation results are used for reinforcing decisions. The algorithm's stages are the listed next. Figure 3.1 presents them in a visual manner.

1. Computing dense disparity maps
2. Applying the threshold
3. Matching the results
4. Post-processing
5. Computing the coordinates and outputting the results

For the following discussion, simulation images will be used. The original test images will be the ones of figure 3.2.

3.2.1 Computing the disparity map

Computing the disparity map is the first step for identifying the hand's location. Area-match algorithms came into discussion as an highly parallellizable approach for segmenting an image into focal planes. A analysis now follows on some implementation aspects of the algorithm according to the project's specific requirements.

Disparity Range

The framework for applying the area-match algorithm, in which an area is chosen on the reference image and searched for in the candidate image, has one important implementation aspect to be discussed: how large should the candidate area be? This is, how large should be the area searched, in the candidate image, for the reference matrix. Note that the width of the candidate area defines the maximum disparity attainable. This matter is of concern because the maximum disparity range limits the distance of the hand to the cameras.

(Consider, for this discussion, the RL sweep: the right image is the reference image, and the left image is the candidate one. Recall that, in this case, the sweep is made from left to right, and that it only makes sense to start searching for a candidate window in the same horizontal location.)

The upper bound for the candidate area width is the distance from the reference window location to the rightmost edge of image. However, the reference window will be shifting towards right, meaning the candidate area will gradually be getting smaller. This variable-size candidate area approach is not often used, unless the first focal plane objects are really very close to the cameras. Instead, a fixed-width candidate area is normally used.

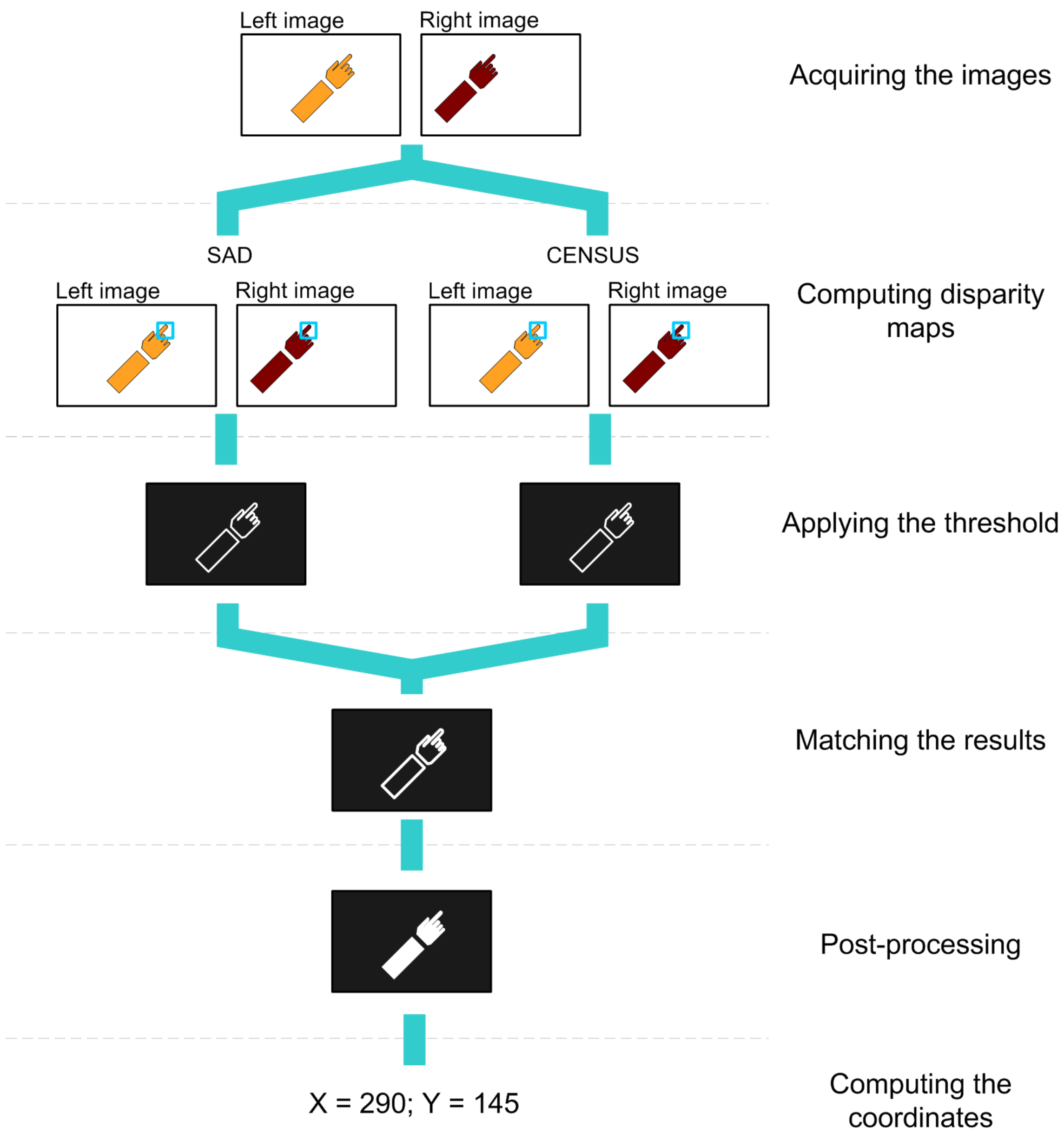


Figure 3.1: All the stages of the processing sequence, in a flow diagram.

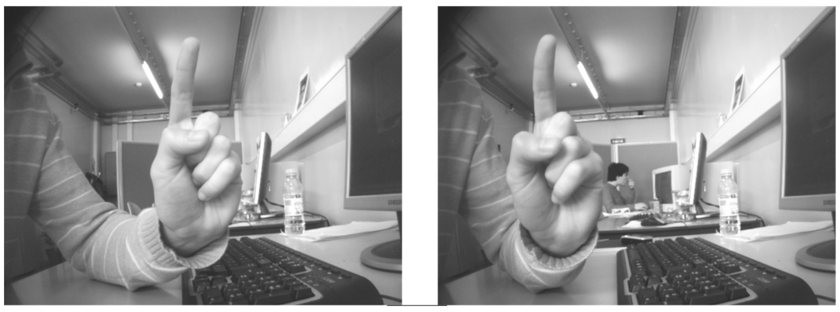


Figure 3.2: The original left and right image used for simulation.

If one is to use a fixed disparity range, it must be contemplated that, the wider the candidate area, the larger the disparity that can be computed. Also, that disparity is inversely proportional to the distance of the object to the cameras - the closer it is, the larger the disparity. Considering the scope of this project, it is not expected the hand to be extremely close to the cameras, thus allowing for relaxing the maximum disparity to be achieved. Moreover, distance between cameras and their focal distance (a concept discussed in section 2.2) must be considered in defining a width for the candidate area.

Experience from other reports was also an influential factor in striking a value. Putting forward some numbers, the ETH Report system, that works on 320 pixels wide images, reports a disparity of only 47 pixels. The system depicted in [15] presents a 150 pixels disparity range on 450 pixels wide images. Other systems, as discussed in chapter 2.3.2, do not also present much higher values. All things considered, and bearing in mind this system will use 640 pixels-wide images, a disparity range of 160 pixels was seen as reasonable. This value also emerges naturally from the adopted architecture, as it will be discussed in the next chapter 4. For information, using a disparity value of 135 (the effective disparity range of the system), objects can be as close as 40 cm.

Cost-functions

As to the second component, the cost-functions: which one is to be used? Each metric's resource consumption and quality must be assessed; the one that strikes the best compromise among these two criteria is expected to be the best suited for implementation. The tested cost-functions were SAD, SSD and CENSUS. Alternatives have a more complex operation and aim at better results, but it has been stated before that no extremely sharp visual representation of the hand is needed, as its location is the aim. The option for a FPGA as the development platform also restrained the set of options to those that did not presented a very complex operation.

As to the first criterion, recall table 2.1. SAD is clearly the least computationally expensive, as it requires only comparisons and subtractions. SSD requires multiplications, which is still within the FPGA's resources: the Xilinx Virtex 2 Pro features 18×18 multipliers. Therefore, it was considered for analysis. NCC, however, demands divisions and square roots, which are rather complex operations (although possible to implement, using for example BRAMs as look-up tables) and therefore has not been considered. CENSUS cannot really be compared, as it is a non-parametric cost-function, but the operations it involves seem to be within the FPGA's basic capabilities (comparisons, bit-wise XNORs). The article by Porter and Bergmann on [11] performs a comparison of SAD, SSD, NCC, CENSUS and Rank implementation on FPGAs, arriving to the conclusion that SAD and CENSUS are the least space consuming.

On the metric's quality, the simulation results are shown from figure 3.3 to figure 3.5, for a window of 3x3 and a disparity range of 135. On the left side of the figures is the LR sweep (the left image is the reference and the right image the candidate one), and on the right the RL sweep. No substantial increase in quality is to be expected from the least computationally demanding metrics to the most demanding. Again, CENSUS stands out for its particular operation.

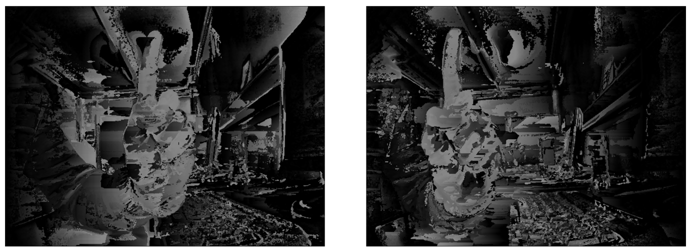


Figure 3.3: Disparity map using SAD over a 3×3 window.

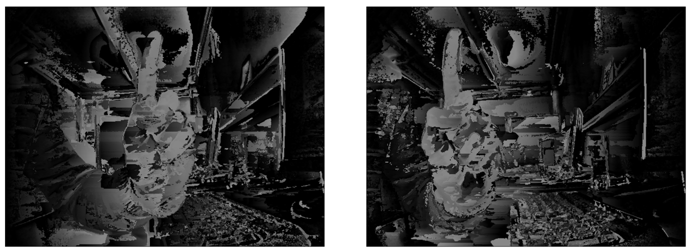


Figure 3.4: Disparity map using SSD over a 3×3 window.

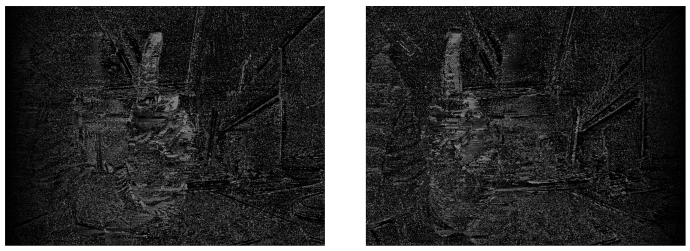


Figure 3.5: Disparity map using CENSUS over a 3×3 window.

Having these results in mind, the ETH researchers [5] propose an interesting approach: to use several cost-functions simultaneously. Having different methods performing the same task adds redundancy and therefore better system robustness is obtained. Furthermore, the two (or more) cost-functions can be complementary.

This complementarity translates into several actions: reinforce good decisions made by the first metric; erode incorrectly classified zones; add foreground areas that were overlooked by the first metric; avoid incorrect classification of zones. But combining certain cost-functions will yield better results than combining others. For example, take SAD and SSD (presented in table 2.1): both are quantitative methods, and the information provided by both will always be similar; just more refined in the second case. However, the CENSUS operation, which is considered non-parametric, is fundamentally different from SAD and SSD, thus giving information from a different perspective. Intertwining the results of SAD and CENSUS would thereby be more effective.

In order to make this matter clearer, one has to analyze more deeply the nature of the cost-functions. Take SAD, for a start. The SAD cost-function only captures information about the total luminance of the areas on scrutiny. In a simple interpretation, it will tell us if the areas in the reference window and in the candidate window have similar brightness, which is useful for detecting surfaces. However, it contains no spatial information. Suppose a picture composed of a black square in a white background. An area on the leftmost border of the square will yield the same results than those of an area on the rightmost border, according to SAD.

CENSUS, however, is best suited for edge detection. The binary coding of the matrix encloses some spatial information, as it discriminates, to some extent, the two or more surfaces present in the reference window. If a candidate window in the other image yields the same binary codification, one can pinpoint that area as having a similar geometry. On the other hand, CENSUS comtemplates no information on the absolute intensities. Take this case: if the reference window depicts an edge between a black surface and a white surface, CENSUS might generate the same binary codification for some similar edge, but where the surface colors are simply white and beige.

This latter characteristic of CENSUS can be considered a drawback, as well the SAD difficulty to indidualize edges. This helps explaining the usefulness of more than one cost-function: one alone would not provide information enough for a robust system operation, and the SAD and CENSUS cost-functions are, in some sense, complementary. All thing considered, SAD and CENSUS seem to be, for the quality of their results, the low resource consumption they imply, and the complementarity of their operation, the ideal cost-functions for implementation.

Window Size

A rather more powerful system parameter is the window's dimension. The minimum size for the window is naturally 3 pixels wide by 3 pixels tall, but simulation tests show how the algorithm's capability for individualizing focal planes is bettered by increasing the window size, as more pixels help individualizing the surroundings of a pixel. The ETH Report [5] reinforces this idea. On the other hand, excessively big windows may cause locality to be lost.

Figures 3.6 through 3.9 presents tests using SAD with 5x3, 5x5, 7x7 and 15x15-sized windows. A thresholded version of the same image, with a threshold of 120 (in a range of 135), on the right side of the figures, helps visualizing the consequences of altering the window dimensions. In particular, the hand area gets increasingly more coerent as the window dimensions enlarge; simultaneously, noisy areas better rejected.

The window dimension to be used is uniquely determined by the FPGA's capacity. Finally, to close this subsection, it is sensible to draw attention to the fact the output of this stage is not one, but 4 disparity maps: SAD and CENSUS over the LR sweep, SAD and CENSUS over the RL sweep.

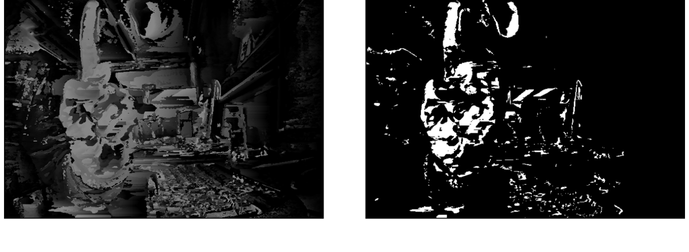


Figure 3.6: Disparity map using SAD over a 5×3 window and its thresholded version.

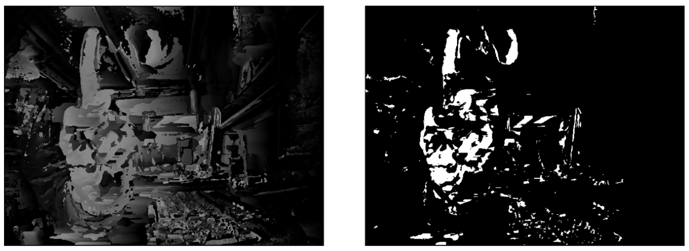


Figure 3.7: Disparity map using SAD over a 5×5 window and its thresholded version.

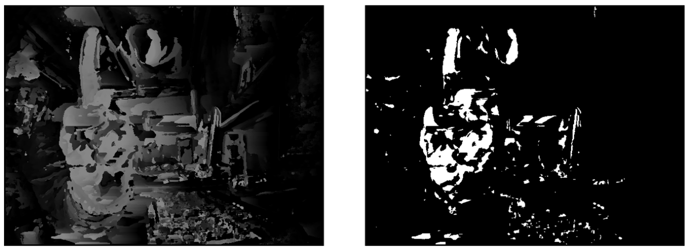


Figure 3.8: Disparity map using SAD over a 7×7 window and its thresholded version.

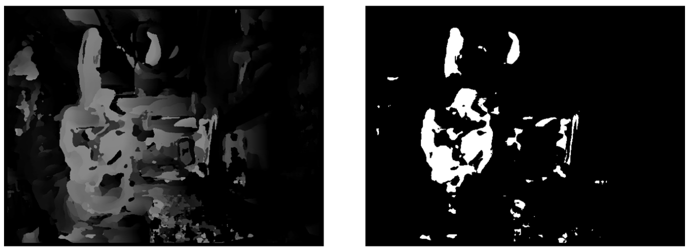


Figure 3.9: Disparity map using SAD over a 15×15 window and its thresholded version.

3.2.2 Thresholding

The implementation of the area-match algorithm has been presented. The outcome of such stage, a disparity map, shows the depth of each physical point to the cameras. However, in order to individualize the hand (or any object), a thresholding operation must be made to differentiate focal planes. To keep matters simple, it is assumed in this project that the hand is the closest object to the cameras. In this scenario, the disparity map's intensity distribution is a large amount of low-intensity pixels, that correspond to the background (due to having a small disparity), and a small number of higher intensity pixels, corresponding to the hand, which is closer to the cameras. Figure 3.10 explains the concept.

The assumption that the hand is the nearest object means that only a lower threshold is needed: only the pixels whose intensities are higher than a value are considered. This was already done in figures 3.6 through 3.9. Nevertheless, the importance of this operation must be emphasized. Note that the value the threshold is set up to controls much of the information that is passed to the following stages. To understand how, the results for the area-match algorithm, after thresholding,

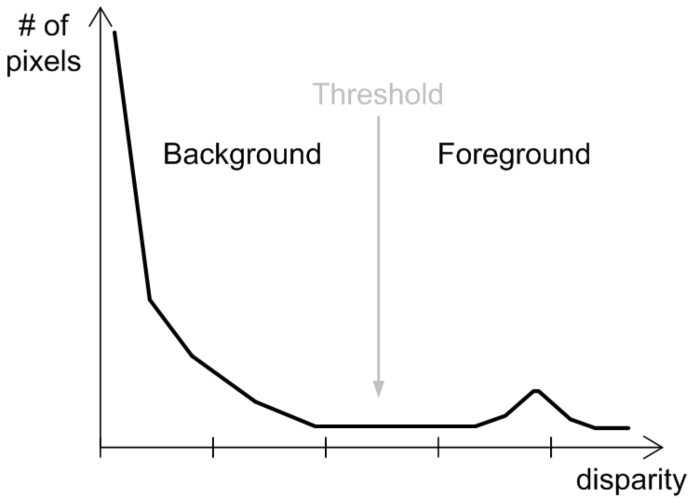


Figure 3.10: An example graph for the expected number of pixels for each disparity value. Figure adapted from [5].

are presented in figure 3.11, for either SAD and CENSUS operation. The maximum achievable disparity is 135, and the threshold is set to 120.

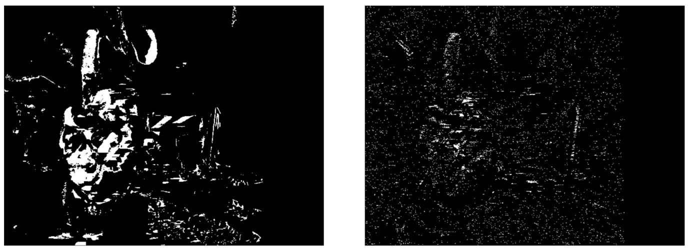


Figure 3.11: Disparity maps for SAD and CENSUS thresholded at 120.

The same output, but now for a threshold of 80, is presented in figure 3.12. One will notice that, in one hand, the hand area is larger, but on the other hand a considerable amount of noise was added. The conclusion is that setting the threshold upper or lower is no trivial task, and a compromise must be struck.

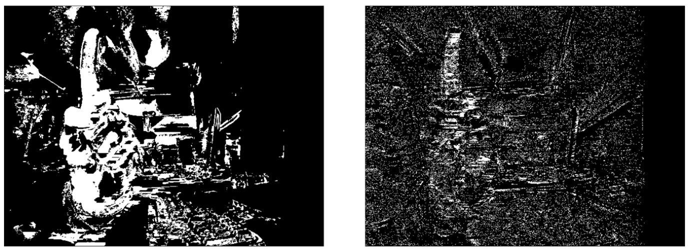


Figure 3.12: Disparity maps for SAD and CENSUS thresholded at 80.

The output of the thresholding operation are 4 bitmaps, one for each disparity map. Pixels need no longer to be represented by 8-bits (which conveyed their disparity); one single bit suffices to inform if each pixel belongs to the foreground (the hand), or not. This simplification also sets the conditions for merging the 4 results into only 2. A OR operation performed between the SAD and CENSUS results of each sweep. If the pixel is foreground in any of the bitmaps, it remains as a foreground. This way, the information provided by both metrics is carried in one single bitmap. The result of merging the SAD and CENSUS is presented in figure 3.13, for a threshold of 120 over each result.

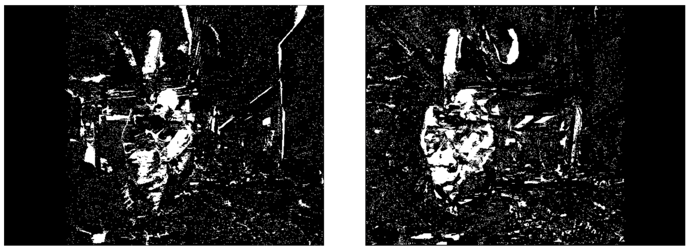


Figure 3.13: Bitmaps produced by merging SAD and CENSUS results of both sweeps. LR sweep outcome on the left, RL on the right.

3.2.3 Matching the results

The problem of occlusion has been addressed in the previous chapter 2. It happens when a area of the reference image finds its location occluded, in the candidate image. This means a best match for that area will only be found farther in the candidate area, as this is swept. This can be noted in figure 3.14, which shows, for the disparity maps of SAD over both sweeps at a threshold of 120, incorrectly classified areas. The original images are shown also for convenience.

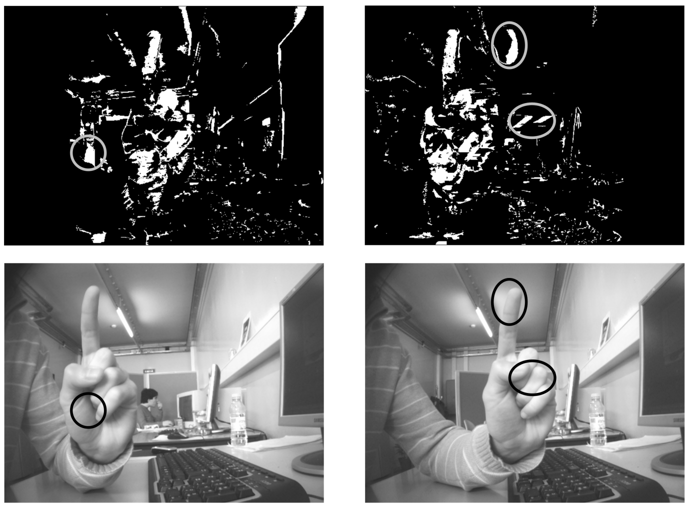


Figure 3.14: Occlusion areas in the simulation results. LR and RL sweeps using SAD are shown, as well as the original images for better comprehension.

The solution for this problem is to perform a consistency check. Take the LR sweep, for example. For each pixel, the corresponding disparity is used to verify the RL sweep. If the pixel in the LR and in the RL sweeps is a foreground one, it remains one. Otherwise, it is turned black. The process is explained better in 2.2.4, and the results of this stage are shown in figure 3.15.

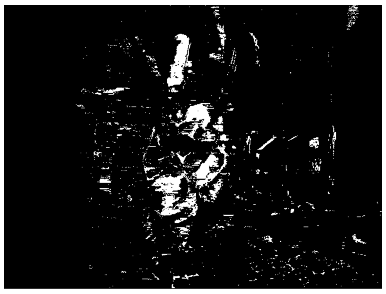


Figure 3.15: Bitmap produced by matching the two sweeps resulting bitmaps.

3.2.4 Post-processing

Having made the consistency check, a lot of undesired noise remains. The ETH researchers addressed this problem by performing a low-pass filtering of the consistency check's disparity map. Simply put, a pixel would be considered as a foreground pixel if its four neighbours, two to the left and two to the right, would be also foreground pixels. Afterwards, the resulting bitmap would be merged with the information provided by the color segmentation of the scene, where areas with chrominance values close to those of the hand would be pinpointed.

This project proposes taking this approach further. It is based on the assumption that one does not need to have a perfect contour of the hand in order to approximately know its position. With this in mind, the idea lies in dividing the 640 × 480 pixels in square blocks of 20 × 20 pixels. Afterwards, each block is classified according to *the number of foreground pixels it contains.* If it has more than a certain threshold, a 1 will be associated with that block; otherwise, it will be assigned a zero. One then gets a 32 × 24 bits matrix, which can be seen as a rough, less detailed, more coarse-grained, version of the former 640 × 480 pixels image. Figure 3.16 shows the idea.

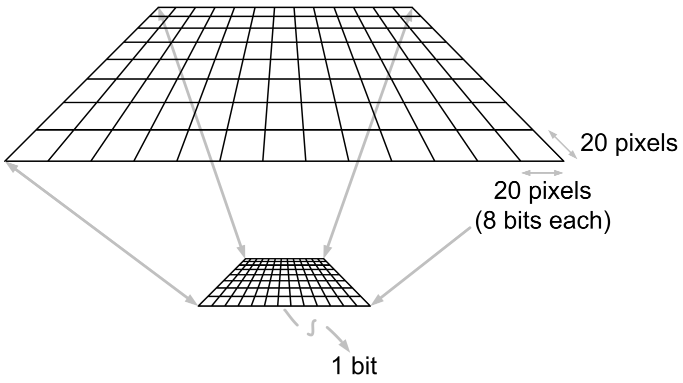


Figure 3.16: A smaller representation of the bitmap by the Matching stage.

The objective here is to enhance the areas that have high concentration of foreground pixels, by assigning the full 400 pixels of the block as foreground pixels. On the other hand, the zones that have not so many pixels will be fully shutdown, and those will not count for the coordinates computation. Naturally, one can talk of lose of sharpness, but that bears no relevant consequence unless one desires to see the bitmap outputted by the system. In that case, the hand shape will be severely distorted, but otherwise, there is no need for extreme sharpness. Even if the full 640x480 pixels were to be considered, the system's output, a set of coordinates, would always be just an approximation to the hands location. The results of such operation are exhibited in figure 3.17.

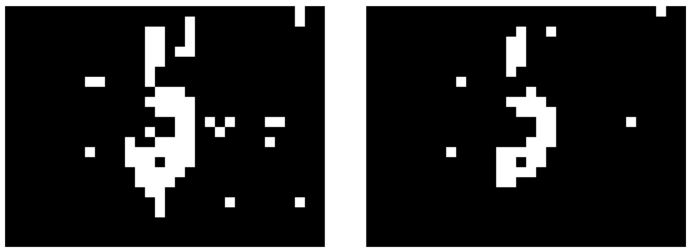


Figure 3.17: Bitmap after post-processing. On the left side, for a threshold of 50 foreground pixels; on the right, for 100.

This "coarsing" of the smallest entities considered (before a pixel, now a 20x20 pixels block) facilitates the implementation of some rather appealing functionalities, thanks to the low memory consumption it implies. They are addressed further in 6.

3.2.5 Computing the coordinates and outputting the results

Finally, the hand's coordinates are computed. In order to do so, the center of gravity of the foregroung pixels in the image outputted by the previous stage is calculated. This operation is equivalent to computing the center of mass of a multiple particle system, with the constraints that the particles rest in discrete positions and all of them have the same weight.

For each line, the weighted sum of all the pixels in it is computed. When all lines have undergone this operation, the sums of all line are summed up. This value is then divided by the total number of foreground pixels, thus yielding the Y coordinate. The same process is made for the rows, to obtain the X coordinate. Figure 3.18 represents the process.

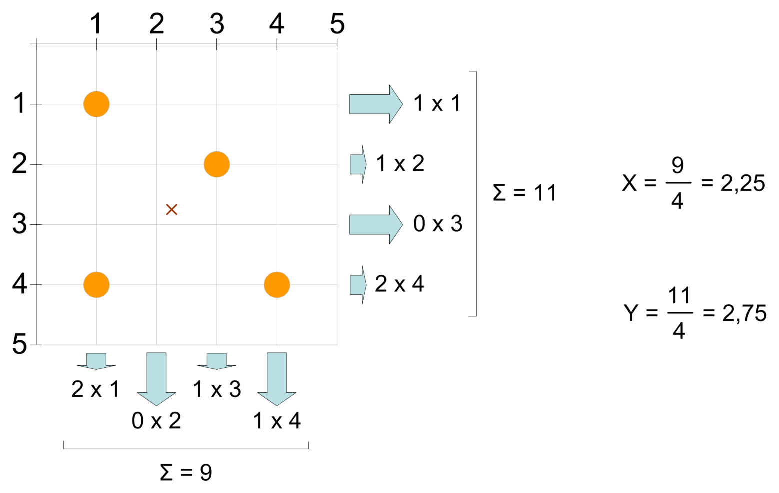


Figure 3.18: Multiply the line or row index by the number of objects in those coordinates. The total of all rows/lines sums, divided by the number of objects, yields the center of gravity.

3.3 Physical System Setup

The physical components that constitute the system are now presented.

Cameras: The cameras are two CMOS (digital) cameras, similar in every way, implemented in a fixed structure. They are disposed according to the stereoscopy principles, i.e.: they are pointing in the same direction, at a known distance from each other. The cameras can work either in Interlaced or in Progressive mode. The latter means that lines are given sequentially, while in the first one first all the odd-numbered lines are sent, and only then the even-numbered lines. Also, the cameras have two data buses: one for luminance values and other for chrominance values. However, if need be, both values can be sent by one single bus, at a double rate. For this project, only the luminance values will be used.

As to the IOS, the most relevant input into the cameras is "EXTCLK", a user-defined clock that controls the cameras' operation frequency (in this case it will be 25MHz); also, an initialization through a I2C bus is needed, but all it does is to change the cameras operation mode from Interlaced to Progressive. As to the cameras outputs, one can mention only the 3 more important: "pixel clock", a clock that is synchronized with the rate at which the pixels' intensity values come up in data bus; "href", which remains as a logical 1 while the pixels of a line are being sent (needed because the line width can change); and "vsync", that signals a new frame. Additionally, it should be mentioned that the cameras have an automatic gain adjustment that can be disabled. More technical information about the cameras can be found in [18].

Support-board for the cameras: A custom-made PCB board for supporting the cameras and interconnecting them with the FPGA's PCB was fabricated, within the scope of this project. It features buffers to restore the cameras signals, and it is connected to the XUP board through the 50-pins High-Speed Expansion header. This PCB is discussed further in section 5.1.

FPGA: The FPGA used in this work is a Xilinx Virtex II-Pro, in particular the XC2VP30. It features 2 PowerPC processors, 13,696 slices and 136 Block Rams, just to mention a few specs. The software working environment is Xilinx's ISE (Integrated Synthesis Environment) for synthesis, and Impact for streaming the produced bitstream into the FPGA through USB. The hardware description language is Verilog.

This circuit is embedded in the XUP board (Xilinx University Program). It is a PCB that features other modules, as a DDR RAM memories slot, a Fast Ethernet port, a XSGA output, a serial port, a High-Speed connector I/O devices, among others.

Adapter board for VGA output: A PCB is used as an interconnector between the FPGA and the monitor. Although the XUP board features a module for XSGA output, it is significantly hard to synchronize the algorithm's output data rate with the VGA synchronism signals for the monitor. Therefore, the mentioned PCB features three FIFO Rams, which allow data to be written under a specific timing (the algorithm's output data rate), and data to be read under a different timing (the VGA timing).

The adapter board is connected to one of the expansion headers of the XUP board, in order to receive the data from the developed system, and features on the other end an VGA connector. The VGA synchronism signals are produced by a Verilog module in the FPGA.

Monitor: The monitor is a crucial element in validating the algorithm, and in particular evaluating each stage's outcome. It is connected to the adapter board FPGA-VGA.

Personal Computer: Finally, a PC is the recipient of the final output of the system. The hand coordinates will be sent through a serial port, and the mouse cursor will move accordingly. The own XUP board Serial Port is used to communicate with the PC. The protocol and data handling is taken care by a module inside the FPGA.

Finally, a figure 3.19 is shown exibiting the main components of the full project.

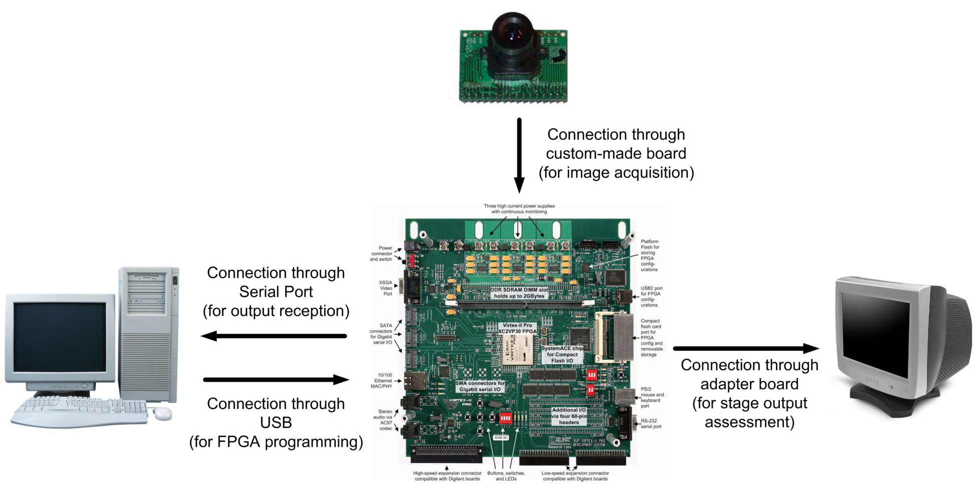

Connection through custom-made board (for image acquisition)
Connection through Serial Port (for output reception)
Connection through USB (for FPGA programming)
Connection through adapter board (for stage output assessment)
DDR SDRAM DIMM slot holds up to 2GBytes
Virtex-II Pro XC2VP30 FPGA
SystemACE chip for Compact Flash I/O
SMA connectors for Gigabit serial I/O
Additional I/O via four 60-pin headers

Figure 3.19: The system's main physical components.

Chapter 4

Hardware Architecture

Each stage of the brief processing sequence description of the previous chapter will now be reviewed in detail. The chosen architecture will be discussed: how it implements the algorithm and how were the data storage and processing structures adapted to best suit the resources offered by the FPGA.

The following table 4.1 maps the algorithm's stages that were presented in the previous chapter 3 into hardware modules. Figure 4.1 presents the global data flow.

Algorithm stage	**Hardware Module**
Acquring the two images	"mem_3x3"
Computing the disparity maps	"metric_fund", "global_ctrl"
Applying the threshold	"thresholder"
Matching the results	"matcher"
Post-processing	"enhancer"
Computing the hand's coordinates	"coordinate computer"

Table 4.1: The correspondence between the conceptual stages and hardware modules of the algorithm.

4.1 Acquiring the two images

The images are provided by the two cameras, disposed side by side in order to implement a stereoscopic setup. Their characteristics have been discussed previously, in section 3.3. The frame rate is of 25Hz, and the actual resolution of operation is 640 pixels wide per 480 pixels tall.

The cameras deliver 8 parallel bits per pixel, at a rate of 12.5 MHz. The 8 bits allow to define a 255-levels grayscale for luminance, and no color information is considered, although such would also be possible. Before being delivered to the memories, the information of each pixel passes through a Verilog module for clock domain transition, transiting from the cameras's pixel clock of 12.5 MHz to the 100 MHz of FPGA operation clock.

Afterwards, the pixels are stored in memory. The algorithm, as discussed in the previous chapter 3, needs the matrices to be at least 3 pixels tall. That means that three lines of both image

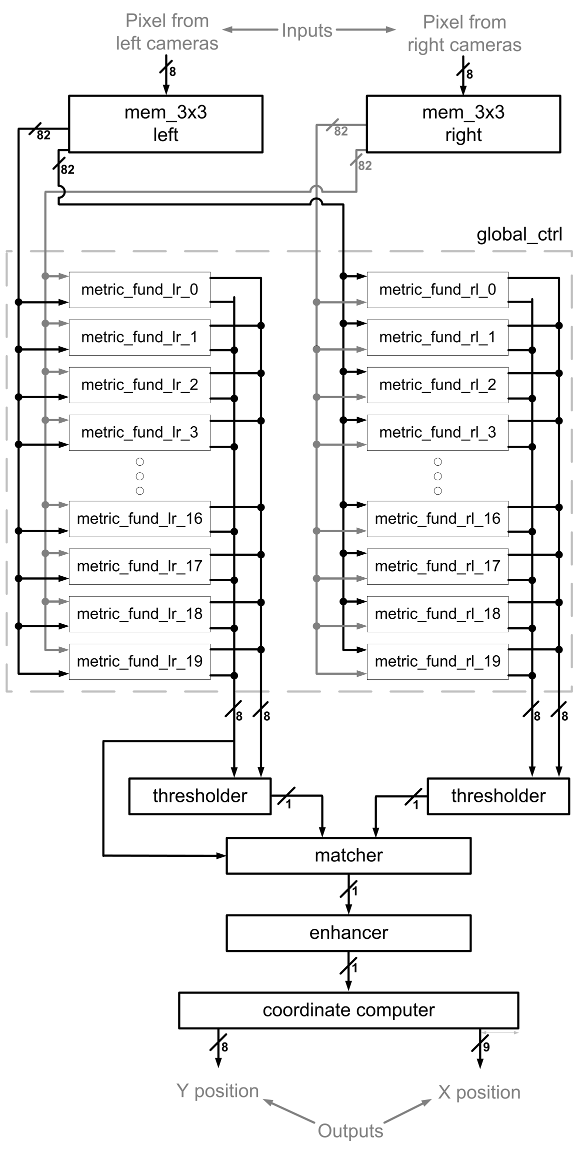


Figure 4.1: Global schematic of the hardware modules.

must be available at each moment. Having this in mind, each camera has assigned one Verilog modules “mem_3x3.v”. This module instantiates four 640×8 bits memories, which corresponds to 4 lines of the image. The fourth line memory is being written while the other 3 are at use. A round robin algorithm guarantees the line memories are given the correct function, as shown in figure 4.2.

The “mem_3x3” module receives from the respective camera the signals “href” and “pixel clock”, which were discussed in section 3.3. The signal “href”, outputted by the camera, is high during the whole transmission of the line pixels from the camera to the FPGA, and can therefore be used as the “write enable” signal for the memories. The signal “pixel clock”, on its turn, indicates when is a new pixel ready for storage. An internal pointer, “wr_ptr”, indicates the next position to be written, and the state variable “wr_state” implements the round robin algorithm.

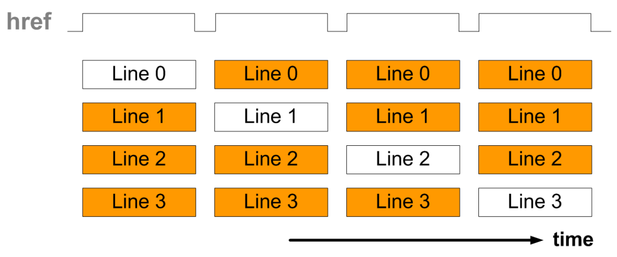


Figure 4.2: The white line is being written, while the others are at use, while "href" is high.

Now, recall that the algorithm, in order to operate, will need a stream of matrices at least 3 pixels wide per 3 pixels tall. Those matrices are displaced one from another by one pixel only. Considering that the line memory 0 is being written, the 3 pixels of the first line of the matrix are given by the line 1. The middle line is given by line 2, and the last 3 pixels are given by line 3. Figure 4.3 shows this.

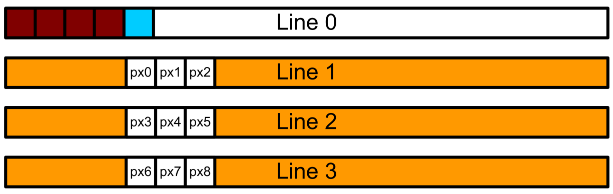


Figure 4.3: Location of the matrix in the 3 operating memories. Line 0 is simultaneously being written.

As said before, the consecutive matrix to be delivered is simply the last one, shifted by one pixel position. It is the case depicted in figure 4.4.

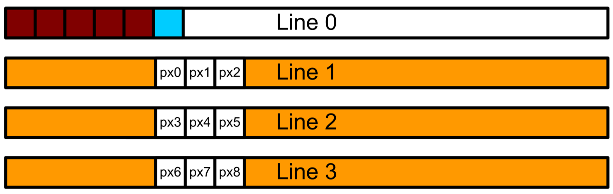


Figure 4.4: The same diagram, but now the matrix has shifted of one pixel position.

However, each memory can only provide one pixel per clock cycle. That means an additional structure is needed to hold the matrix and make it available to the subsequent modules. This structure implements a shift-register operation to make the new matrix available, as shown in figure 4.5.

Bear in mind, again, that the area-match operation will be performed having the left image as the reference image, and the right image as the candidate image, and vice-versa (the aforementioned LR sweep and RL sweep). This means that each memory module needs to have two read ports. Also, that only one shift-register structure does not suffice, but two are needed: one to serve as reference matrix, and the other as the candidate matrix for that image. If you consider to be in the right image "mem_3x3" module, the reference matrix will be used by the RL sweep, and

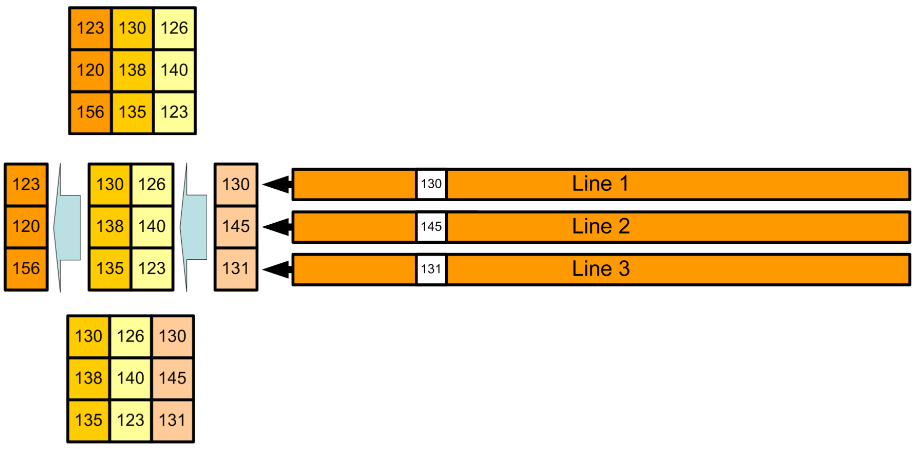


Figure 4.5: The shift-register structure, to implement the shifting window that feeds subsequent modules.

the candidate matrix by the LR sweep. Additionally, two read pointers are needed, but they are given by module “global_ctrl”. They are “rd_ptr_pivot”, for addressing the reference window, and “rd_ptr_test”, for the candidate window. Figure 4.6 shows the interaction of module “mem_3x3” with other modules, while figure 4.7 shows the internal structure of module “mem_3x3”.

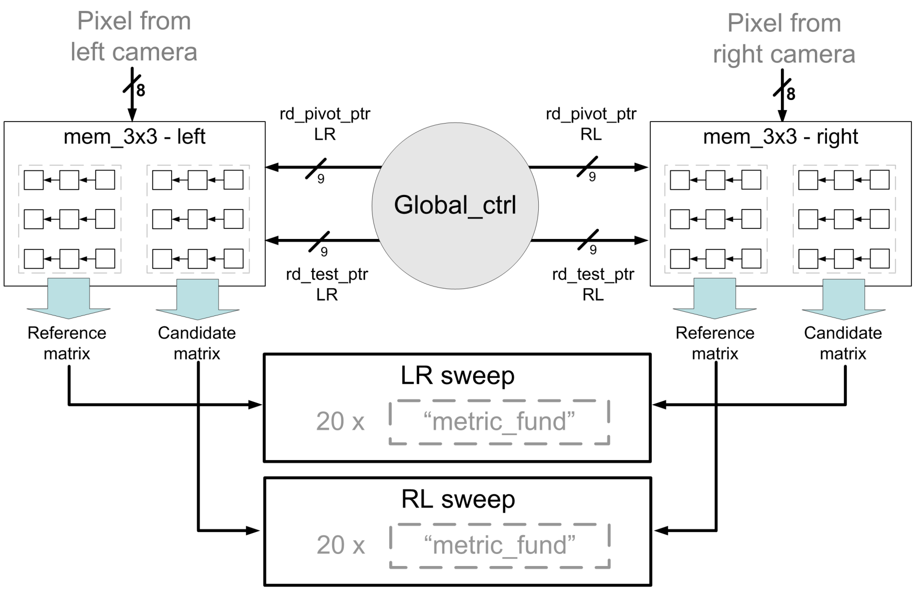


Figure 4.6: “mem_3x3” module outputs and relation with other modules.

On a more technical view: care was taken so that the FPGA’s native BRAMs would be synthesized. There are 136 BRAMs in the Virtex 2Pro XC2VP30, each one holding up to 2K of 9-bits words. Considering each line takes 640 bytes, each BRAM could store up to 3 lines; however, the BRAM constraint of only two simultaneous accesses per cycle does make that hypothesis inadequate, and therefore each line has one full BRAM assigned. This option does not take portability from the system, even if it would be taken into full-custom manufacturing.

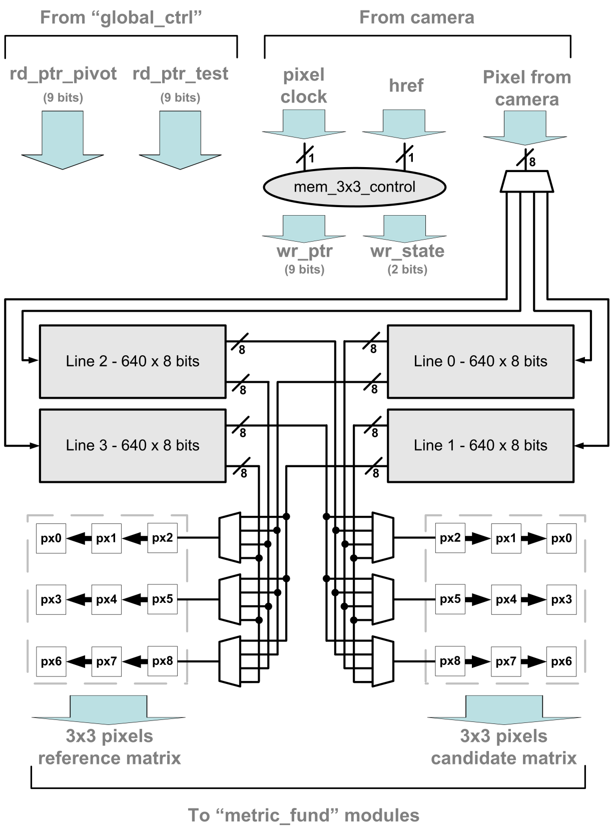


Figure 4.7: The module "mem_3x3". Control flow is not shown; all the multiplexers in the figure are controlled by the "wr_state" variable.

4.2 Computing the disparity maps

For this stage, area-match algorithms are key players. They have been introduced in previous chapters, but now they will be given a closer look under the scope of the project's hardware implementation. Next, it is discussed how SAD and CENSUS were materialized in this system. Finally, the hardware dedicated to the image sweep operation is explained.

As an initial remark, it should be stressed out that, as can be seen in table 4.1, two modules are in charge of implementing the area-match algorithm: "metric_fund" is the fundamental element in the algorithm, implementing the cost-functions, whereas the module "global_ctrl" is responsible for the area-match operation control.

Before anything else, recall the basic operation of area-match algorithms (in this case for the RL sweep - right image is the reference image, left image is the candidate one): a reference window (a matrix of pixels) is considered in the leftmost position of the reference image and compared to a candidate window, taken from the candidate image, from exactly the same horizontal coordinate, on the same line. The candidate window is then shifted right and compared again. The position of the best matching candidate window, relatively to the position of the reference window, sets the disparity for the central pixel of the reference window. Afterwards, the reference window is shifted right one pixel and the process is repeated, for every pixel of the reference image. Figure 2.10 is replicated for convenience, featuring the nomenclature, in figure 4.8.

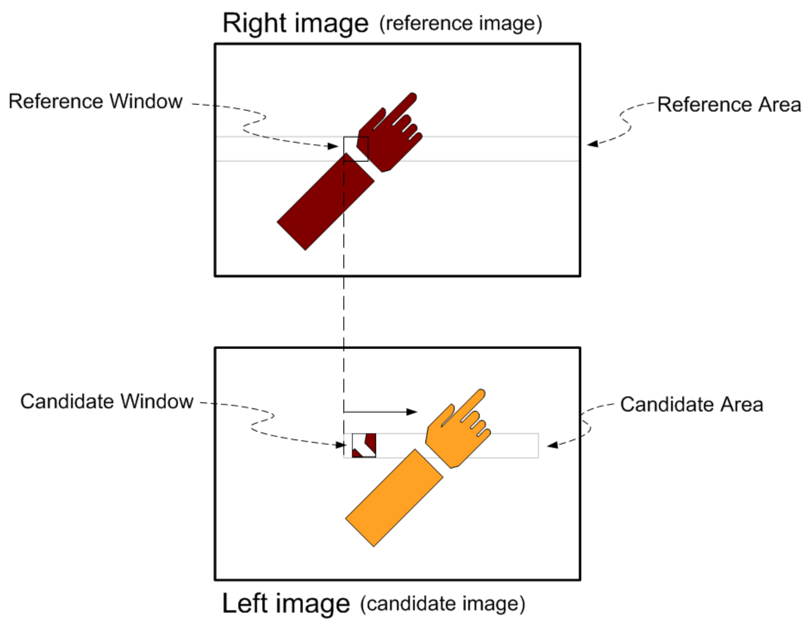


Figure 4.8: For each reference window, a candidate area is swept through. The best matching candidate window defines the disparity for the central pixel.

A constraint that played a major contribution on defining how the architecture would be was the input rate and the processing rate. While data is received at 12.5 MHz, the FPGA works at 100MHz. This means that each input pixel has 8 clock cycles for computation. If one was to use a more resource constrained architecture, where one single cost-function computation per cycle would be possible, the maximum disparity would be of no more than 8 pixels, which is clearly insignificant in a 640-pixel wide image. But naturally, one desires to take advantage of the FPGA's large parallel processing capacity. Two options then arise:

- To compute the full extent of the disparity range, for one pixel, within those 8 cycles.
- To process the disparity of several input pixels simultaneously.

Although this is probably the most significant implementation decision to be made, as it defines most of the area-match algorithm hardware structures and of the system's internal operation characteristics (like output rate, if the output is made in bursts or not, etc.), there are no relevant reasons for prefering one to another.

The first option was chosen by the ETH researchers. They compute a disparity of up to 47 pixels in 16 cycles, for one single pixel at a time. They make use of a high speed ring-buffer, which is 12 pixels wide and loaded with a 4-pixels wide block in every clock cycle. From that ring-buffer, 4 matrices comparisons are made and 4 disparities computed in every cycle. Note that the 4 pixels wide blocks, times 16 cycles, equals a 48 pixels range, thus attaining the desired disparity.

This approach seemed on a first glanced incompatible with the BRAM's capability of only presenting 2 words in one cycle. And most of all, the hardware structure for performing 4 matrices computations in one single cycle did not seemed very simple. The second option, however, presented a more intuitive implementation, as well as the possibility that the same candidate window could be used for the disparity computation of several reference windows, and a better interaction with the BRAMs specific operation.

Having made the choice of processing several pixels at a time, it came as reasonable that each pixel under computation would have its own "metric_fund" module. But how many pixels could be handled at one time? On a first approach, the number 20 striked as the best compromise value, for several reasons. First of all, there is a restriction: the window size (in this case, 640 pixels) must be divisible by the number of modules, in order to achieve full efficiency. Consider the other possibilities: 2, 4, 5, 8 or 10 would yield very small disparities (at most, $10 \times 8 = 80$). 16 modules would provide a good disparity range (128), but this project's FPGA still has capacity for a bit more. 32 modules, however, offer a disparity range excessive for the project needs: 256. Being 20 the in-between divisor of 640, and providing a disparity range of 160 (or, in other words, 160 cycles of computation per each pixel), it striked as the best compromise.

Secondly, one must consider that the area-match computation is not performed only in one direction, but in both. This means the initial 20 modules have now become 40. And naturally, all these numbers must be considered having in mind the resources taken by each cost-function-computing module, so that the FPGA capacity is not exceeded. One will see now what these modules do.

4.2.1 Cost-functions

It seems sensible to present, at this point, the hardware module that performs the fundamental computation of the area-match algorithm: the disparity computation. The module is easily understood, and knowledge of its internal operation will help to comprehend the section after.

"metric_fund" is the module that computes, for each pixel, its disparity. There will be 40 of these: 20 for the LR sweep operation, and 20 for the RL sweep. One of the inputs of "metric_fund" is the "enable" signal; the module only works when this signal is high; otherwise, it keeps the last values in its internal registers. Those are "disparity" and "best_match".

As soon its "enable" input goes high, the "metric_fund" module will store, in that cycle, the matrix that is at its *reference window inputs*. Additionally, it will reset its internal registers. "best_match" is filled with the worst possible value for the metric being used. This guarantees any new cost-function outcome will for sure alter that register. The register "disparity" requires no reset. "count" is a simple counter that is also reset and starts counting from zero every cycle from that moment on.

On the second cycle the "enable" signal is high, the first candidate window comes in. The reference window (which is stored on the register) and the candidate window undergo the cost-function operation. A value is computed and, if better than the value stored in "best_match", the value in register “best_match” is replaced by the newly computed metric value. Simultaneously, the “disparity” is updated with the value from “counter”.

While the “enable” signal is high, the module will keep computing metric values with the incoming candidate matrices. If any of them produces a metric better than the stored in “best_match”, the whole register updating operation is made again. Figure 4.9 depicts the module’s architecture, and figure 4.10 shows the module’s temporal operation.

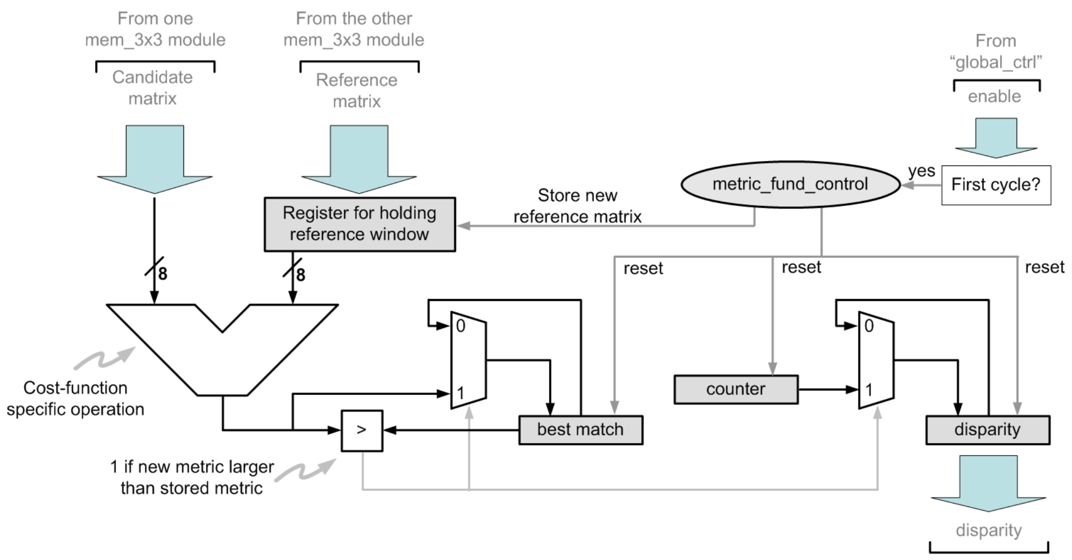


Figure 4.9: The architecture of module “metric_fund”.

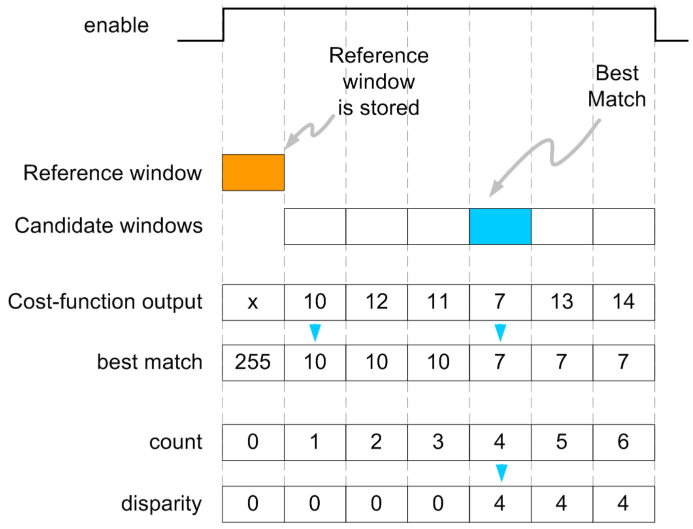


Figure 4.10: Temporal evolution of the “metric_fund” module variables.

The reason why “disparity” is updated by variable “count” is because the incoming candidate windows, that arrive every cycle, are given from positions farther and farther away from the reference window position. Therefore, “count” serves as a counter for the distance, on pixel positions, that the incoming candidate window is from the reference window. Figure 4.11 tries to explain this. The best matching candidate window disparity is stored, because ideally it corresponds to the location, on the other image, of the candidate window that most resembles the reference window.

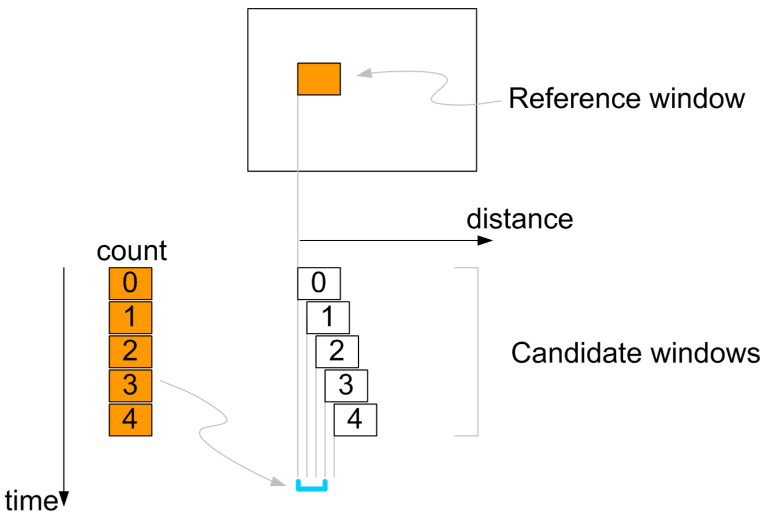


Figure 4.11: “count” is proportional to the candidate windows distance to the reference window.

This generic framework for disparity computation can be used by any metric. The arithmetic component, in the left of figure 4.9, is specific to the cost-function, and therefore there is no point in describing its architecture. Nevertheless, the operation for both SAD and CENSUS, the two implemented cost-functions, is explained next.

SAD, for example, will require that the sum of the absolute differences between each pixel of the reference window and the candidate window is perform, and those values summed. That total, the Sum of Asolute Differences, is the metric that allows to assess similarity among to windows. If some candidate window yields a SAD that is better than the one stored in “best_match”, then this register is updated, as well as the “disparity” register.

For CENSUS, it is necessary to verify which pixels of the candidate window are larger or smaller than the window’s center pixel, and a codeword is created. This codeword is then compared with the one of the reference window, and the Hamming distance is computed (for clarification, see subsection 2.2.3). For CENSUS, the Hamming distance is the value that is tested for a best match. For one reference window, and while the “enable” signal is high, the best Hamming distance is stored in “best_match”, and the location of the candidate window that yielded it *with respect to* the reference window is stored in “disparity”.

One important remark is that the modules latency is defined only by the cost-functions specific hardware; the generic framework takes one single cycle. At this moment, and as a result of the hardware structure used to implement SAD and CENSUS, the module’s latency is of 4 cycles. Finally, it should be mentioned that this structure allows the module to be used indifferently either by the LR sweep, either by the RL sweep.

4.2.2 Scanning the images

Now that the fundamental elements of the area-match algorithm have been presented, it will be discussed how are those modules managed, in coordination with the memory modules. The hardware module "global_ctrl" is responsible for "orchestrating" the all 40 "metric_fund" modules, in harmony with the 2 memory modules "mem_3x3".

The whole module revolves around one single internal variable: "ctr_1". It is a simple counter that increments from 0 up to 159 and then restarts. Most important actions are triggered by "ctr_1" achieving some specific value. The operation of the module is now explained. Assume, for the rest of this discussion and unless said otherwise, the reference image to be the right one, and the left image to be the candidate one.

Recall that the reference window, used by the "metric_fund" modules, is provided by the shift-register structure of "mem_3x3", as seen in section 4.1. The shift-register structure update is triggered when "ctr_1" arrives to 159. On that moment, the reference window pointer "rd_ptr_pivot" and the candidate window pointer "rd_ptr_test", are updated.

In the following cycle (when "ctr_1" = 0), the 3 line memories of the right image "mem_3x3" module will have each presented the pixel corresponding to the address indicated by the reference window pointer, "rd_ptr_pivot". Those pixels are fed into the first stage of the three-level shift-register. For the next cycles, the two pointers will keep incrementing and, when "ctr_1" equals 4, the 3 levels of the shift-register structure will be full. Figure 4.12 depicts the idea.

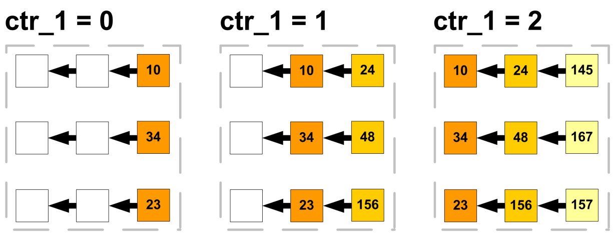


Figure 4.12: Shift-register structure from module "mem_3x3" being updated.

As to the "metric_fund" modules, bear in mind that the reference window will have to be stored for the next 160 cycles, to be compared with the incoming candidate windows. To indicate the first "metric_fund" module that a new reference matrix is ready to be stored, the signal "enable" of that module is put high by "global_ctrl". It does so when "ctr_1" equals 2, so that in the next cycle the matrix is stored in the "metric_fund" module own matrix storage register. Figure 4.13 shows the full operation until now.

Meanwhile, the reference window pointer continued to increase, meaning the reference window stored in the shift-register structure was shifted one pixel position to the right, as presented in figure 4.14. A new reference matrix is therefore available.

Simultaneously, when "ctr_1" equals 3, the "enable" signal of the *second* "metric_fund" module goes high, and the new reference matrix is stored in the second "metric_fund" module (note: not in the first one!). This is depicted in figure 4.15. The same procedure goes on until all the 20

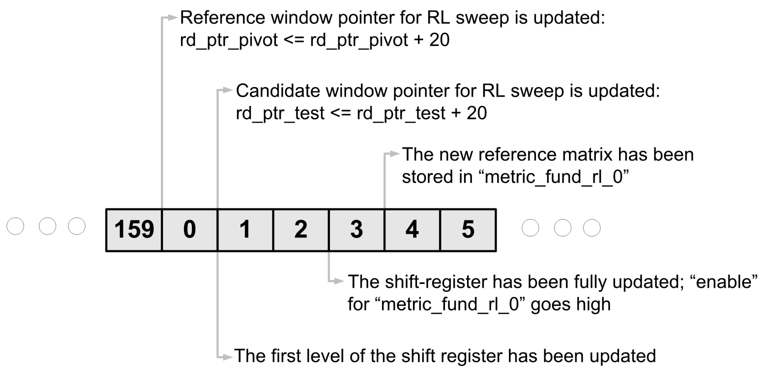

Reference window pointer for RL sweep is updated:
rd_ptr_pivot <= rd_ptr_pivot + 20
Candidate window pointer for RL sweep is updated:
rd_ptr_test <= rd_ptr_test + 20
The new reference matrix has been stored in "metric_fund_rl_0"
159
0
1
2
3
4
5
The shift-register has been fully updated; "enable" for "metric_fund_rl_0" goes high
The first level of the shift register has been updated

Figure 4.13: Chronology of several events, with respect to counter "ctr_1".

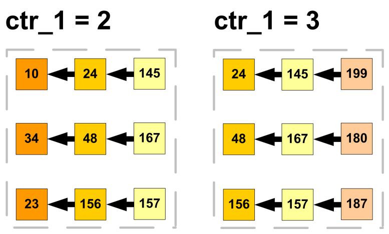


Figure 4.14: Shift-register structure, one shift after being updated again.

"metric_fund" modules have a new reference matrix. Figure 4.16 displays the global sequence for new reference matrices storage.

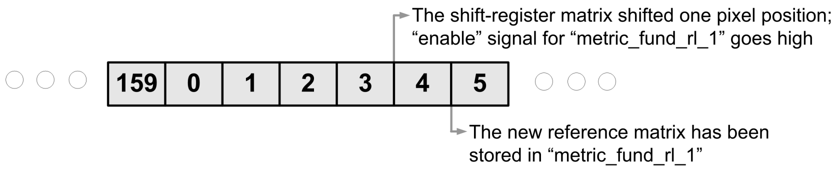

The shift-register matrix shifted one pixel position; "enable" signal for "metric_fund_rl_1" goes high
159
0
1
2
3
4
5
The new reference matrix has been stored in "metric_fund_rl_1"

Figure 4.15: Chronology of several events, with respect to counter "ctr_1".

However, how are the candidate windows fed into the "metric_fund" modules under this setup? The particular architecture that is being described is also well suited for fedding the candidate windows, as opportunity is created for that all the "metric_fund" modules use the same candidate window. To understand this, consider the following: take the reference window, in this case from the right image, located on position 0, and that is fed into module "metric_fund" #0. Only the 160 candidate windows of the left image, starting on position 0 and continuing to its right, are relevant for computation. However, for the next module (#1), the reference window will be the one located in position 1. Candidate window on position 0 is useless.

Figure 4.17 depicts this. The first reference window and its associated candidate area are depicted. One will notice, in the right part of the image, that the next reference matrix is shifted of one pixel with respect to the previous one, *and so* is the candidate area.

Having this characteristic in mind, and combining it with the fact the reference matrices are given to the "metric_fund" modules sequentially over 20 clock cycles (as opposed to giving them

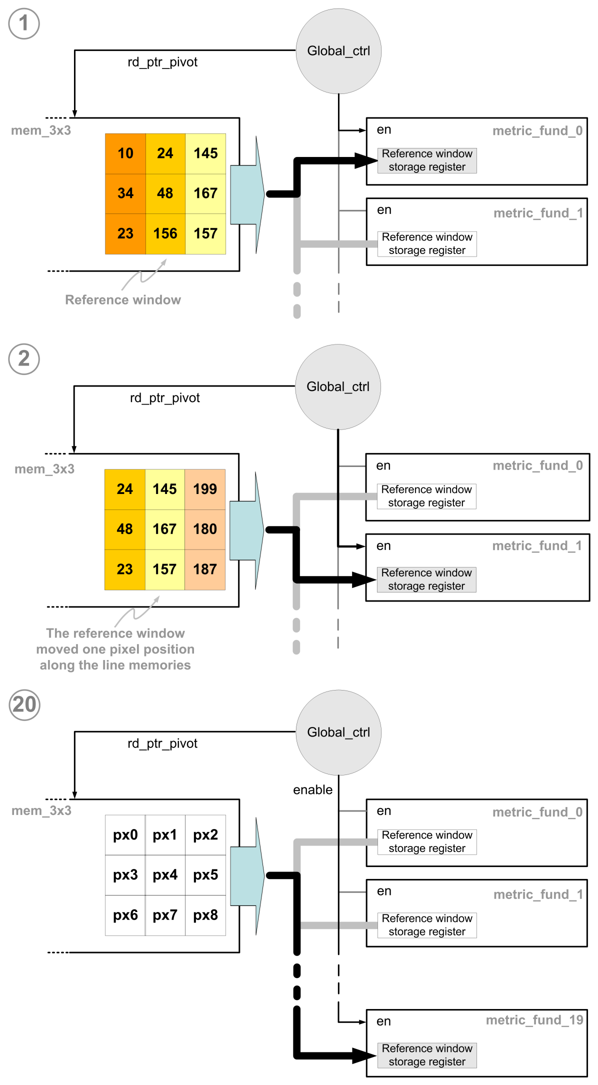

1
Global_ctrl
rd_ptr_pivot
mem_3x3
10
24
145
34
48
167
23
156
157
Reference window
en
metric_fund_0
Reference window storage register
en
metric_fund_1
Reference window storage register
2
Global_ctrl
rd_ptr_pivot
mem_3x3
24
145
199
48
167
180
23
157
187
The reference window moved one pixel position along the line memories
en
metric_fund_0
Reference window storage register
en
metric_fund_1
Reference window storage register
20
Global_ctrl
rd_ptr_pivot
enable
mem_3x3
px0
px1
px2
px3
px4
px5
px6
px7
px8
en
metric_fund_0
Reference window storage register
en
metric_fund_1
Reference window storage register
en
metric_fund_19
Reference window storage register

Figure 4.16: Temporal diagram of the data flow between the “mem_3x3” module and the “metric_fund” modules for the reference windows. Note that the “enable” signal activation and the effective storing of the reference matrix are not simultaneous.

simultaneously to all “metric_fund” modules, in one single clock cycle), it is possible for the same candidate window to be used by all the “metric_fund” modules. Figure 4.18 shows this. It exhibits the data present at the input of each “metric_fund” modules over time. The candidate windows are the same for all “metric_fund” modules.

Therefore, the same bus is used to send data from the shift-register structure of “mem_3x3” dedicated to storing the current candidate matrix, to all “metric_fund” modules. Figure 4.19 then shows the data flow for the candidate windows.

However, this option has a consequence. The full 160 disparity range is not fully attained. In fact, by providing each reference window sequentially, and considering there are 20 “metric_fund” modules, the maximum disparity range lowers from 160 to 140. The real effective disparity range of the system is 135, due to the number of cycles it takes to refresh the shift-register structure.

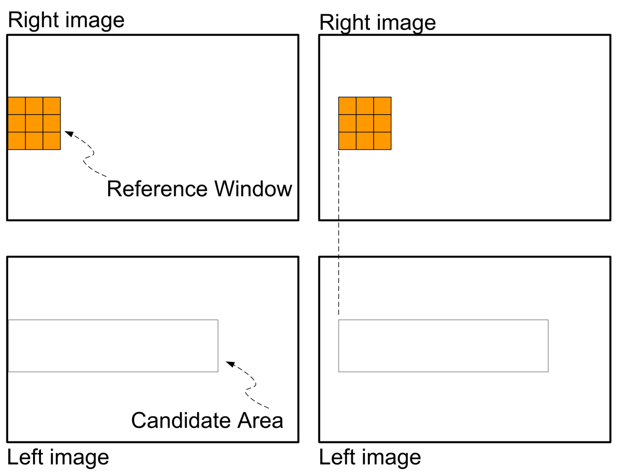


Figure 4.17: Shift of reference window and correponding candidate area. The candidate area has shifted only one pixel position, as the reference window.

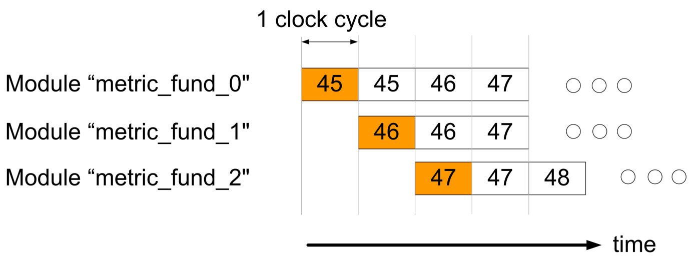


Figure 4.18: The data present at the input of each "metric_fund" modules over time. The orange blocks refer to the reference matrices, and the white blocks are the candidate windows.

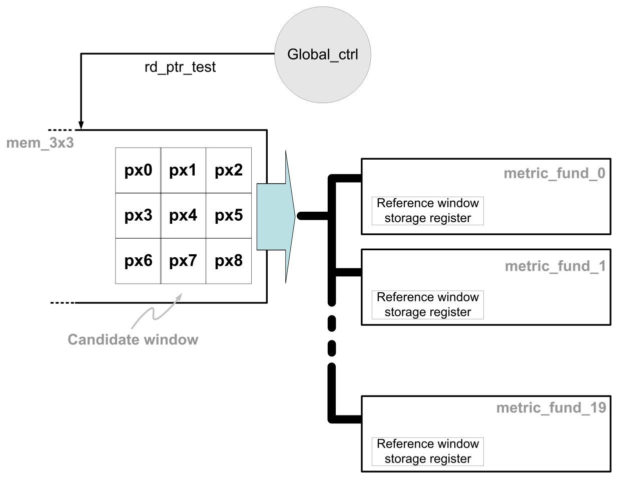


Figure 4.19: Temporal diagram of the data flow between the "mem_3x3" module and the "metric_fund" modules for the candidate windows.

Alternatives have been considered to avoid this lose of disparity range. For instance, memory is not a scarce resource in this FPGA system. One could assign a BRAM, or other kind of memory, for each "metric_fund" module, and fill it with the respective 160 candidate windows that particular module will be using. However, three drawbacks immediately arise from this option. First of all, there would be enormous redundancy: of the 160×3 bytes each memory would store, only 3 bytes would differ. Secondly, choosing this options makes the system lose portability, in particular if it were to be implemented on a ASIC. Finally, new hardware structures would have to be developed to deliver all the reference windows simultaneously to the "metric_fund" modules. This option is, all things considered, unefficient, and the resource consumption it requires does not live up to what it offers.

At this point, when are the results retrieved from each "metric_fund" module? This is done when each module has received 135 candidate windows. But recall that, at the beginning of the computation, module "metric_fund" #0 received one more candidate window than "metric_fund" #1, and 20 more than "metric_fund" #19. In order to provide all the "metric_fund" modules the same disparity range, the "enable" signal is deactivated sequentially for all modules, and the results from each one read on the following cycle. Figure 4.20 depicts the idea.

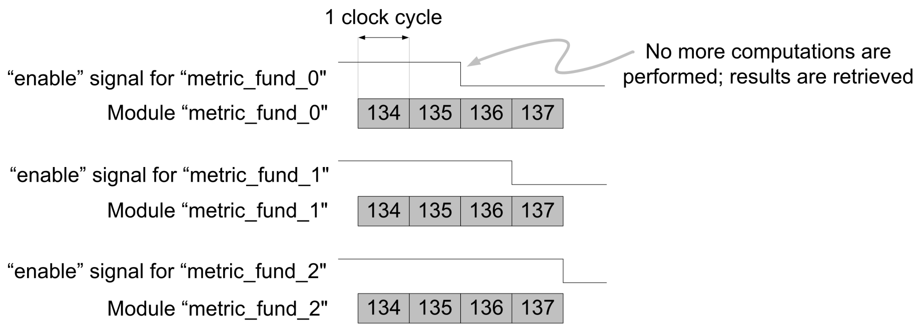

1 clock cycle
No more computations are performed; results are retrieved
"enable" signal for "metric_fund_0"
Module "metric_fund_0"
134
135
136
137
"enable" signal for "metric_fund_1"
Module "metric_fund_1"
134
135
136
137
"enable" signal for "metric_fund_2"
Module "metric_fund_2"
134
135
136
137

Figure 4.20: Timing for result retrieving from modules "metric_fund".

The results of the RL sweep pass through thresholding and are sent to a shift-register structure; the ones from the LR sweep are kept in a buffer. The reason for this disctinction is concerned with the Matching stage of the algorithm, and it will be explained later.

Figure 4.21 serves to illustrate the overall area-match algorithm operation over, and exhibit the diverse temporal scales of the disparity map computation. In each block of 160 cycles, 20 reference pixels have their disparities computed. The 20 results only start being available near the end of the 160 cycles, which means the results come out in bursts of 20 pixels. It is this characteristic of implementation that shapes most of subsequent operation of the algorithm.

Furthermore, there are 32 blocks of 160 cycles, thus assuring the operation of the 640 pixels of the image line (32 blocks $\times$ 20 pixels = 640 pixels). Temporally, $32 \times 160 = 5120$ cycles is the time it takes to process a full image line. It is also the time it takes to receive and store one image from the cameras. The FPGA works at 100 MHz, which means line treatment takes 51.2 microsseconds. The cameras frequency is 12.5 MHz; times 640 pixels, also 51.2 microsseconds are obtained. The same line of reasoning leds to the conclusion that the system could operate at 40 fps, if the images were to be given without interval. However, the cameras operation limit the system frequency to 25 Hz.

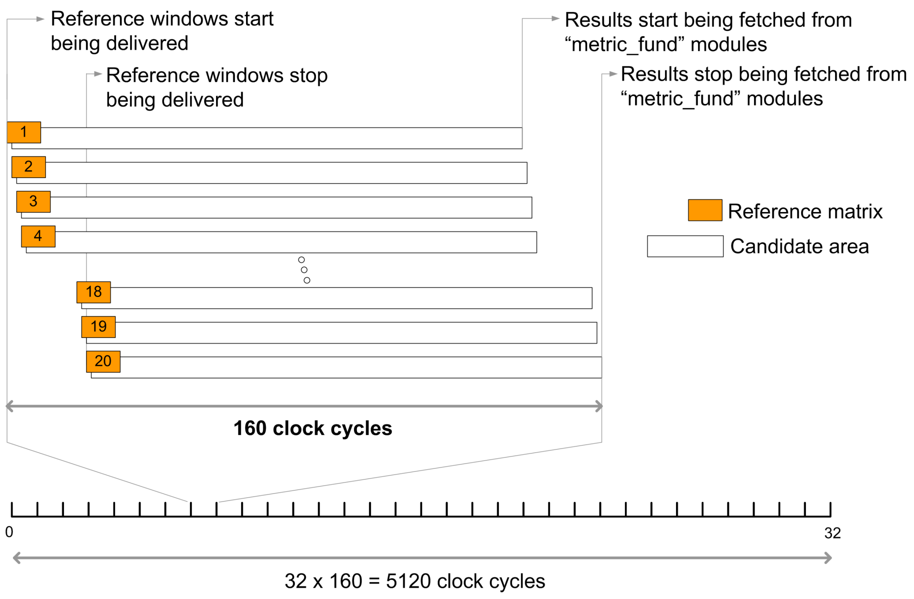

Reference windows start being delivered
Reference windows stop being delivered
Results start being fetched from "metric_fund" modules
Results stop being fetched from "metric_fund" modules
1
2
3
4
18
19
20
Reference matrix
Candidate area
160 clock cycles
0
32
32 x 160 = 5120 clock cycles

Figure 4.21: The two temporal scales of the system operation.

LR Sweep

Until now, the discussion has been made for the case of the RL sweep, but the inverse must also be implemented - this is, having the left image as reference and the right as the candidate one (the LR sweep). The most important alteration one must bear in mind is that the area match, in this case, is performed from the right to the left.

Figure 4.22 exhibits the differences in operation. The orange blocks correspond to 20 3×3 pixels reference matrices. The white blocks refer to the candidate area searched for those particular reference matrices. The blue arrows represent the direction of the shifts of the candidate windows.

Having this in mind, which alterations are needed in order to implement the LR sweep? A simple idea is to have the candidate window pointer running backwards. The temporal diagram for the data flow in the "metric_fund" inputs is shown again, but now considering the new situation, in figure 4.23.

One will notice the candidate windows are no longer common. This is avoided if also the reference window pointer runs backwards. It is the case of figure 4.24.

This way, one can re-use the structure developed for the RL sweep, with little modifications. The reference window pointer will start not on position 0, but in position 20, and for the first round of 20 reference matrices it will decrement to 0. When 20 new reference matrices are to be treated, the pointer will be updated to 40, and it will decrement to 20, thus providing 20 new reference matrices.

A consequence of this approach is that the first result to be ready will be the reference window with the highest address. Consider one is treating reference matrices from position 20 to 40. The 40th matrix will be the first to be computed, and the 20th the last one. This presents an inconvenience: the next stages of the algorithm (in particular Matching) require that the results are

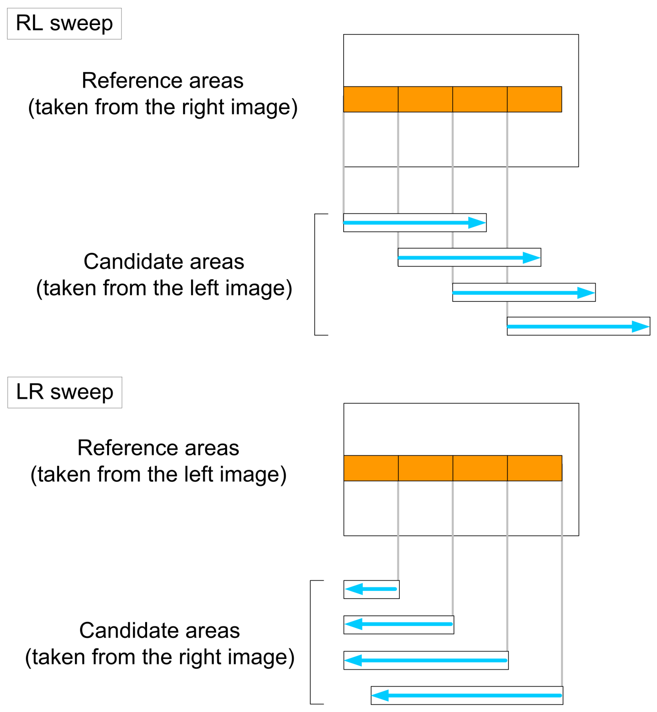

RL sweep
Reference areas
(taken from the right image)
Candidate areas
(taken from the left image)
LR sweep
Reference areas
(taken from the left image)
Candidate areas
(taken from the right image)

Figure 4.22: The blue arrows exhibit the direction of the sweep, for each case.

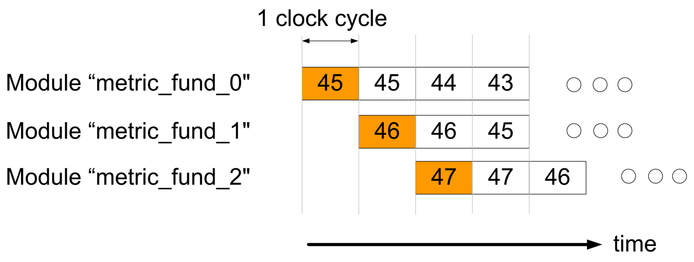


Figure 4.23: The data present at the input of each “metric_fund” modules that will perform the LR sweep. The orange blocks refers the reference matrix, and the white blocks are the candidate windows.

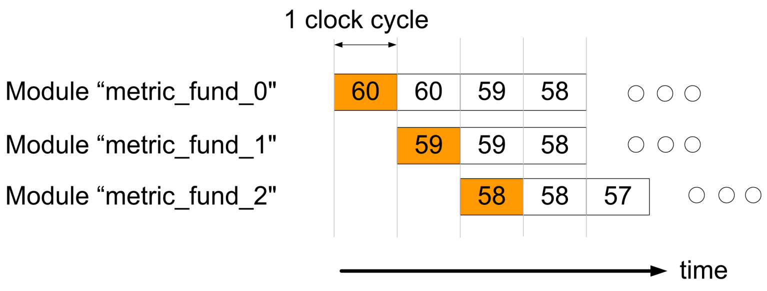


Figure 4.24: If the reference window pointer also runs backwards, the same candidate window can be used by all “metric_fund” modules implementing this sweep.

given in order. Therefore, these results are stored in a buffer of 20 bytes, and thereafter read in the correct order.

One might question if it would not be possible to perform an independent LR sweep structure, instead of re-using the RL structure. In this case, the sweep would be made entirely from right to left, instead of “emulating” that behaviour in the RL structure (which runs rightwise). It would be possible, but considering that the following steps, in particular Matching, are to be done real-time, it is of interest that the results computed in one of the sweeps are physically close to the ones of the other sweep. This matter will be made clearer as soon the matching process is explained.

Close-edge windows

Another matter that was taken into account when developing the Verilog modules must be mentioned: not all the reference windows will have the full 160 candidate windows available. To understand why, consider the following example: taking the right image as the reference one, note that reference windows in positions less than 160 pixels away from the *right* edge of the image will not have the full 160 pixels for disparity calculation.

Recall figure 4.22. One will note that, on the LR sweep operation, for the first 20 reference windows, the candidate window pointer soon finds the image border. It means the full 160 windows are not fully available. This is taken care by the “global_ctrl” by deactivating the “enable” signal of the respective “metric_fund” module when no more candidate windows are available. The module will keep the last values. The same happens in the RL sweep, when reference windows are getting closer to the right edge of the image. Figure 4.25 shows the same process of figure 4.22 but on a temporal basis, and for one single module.

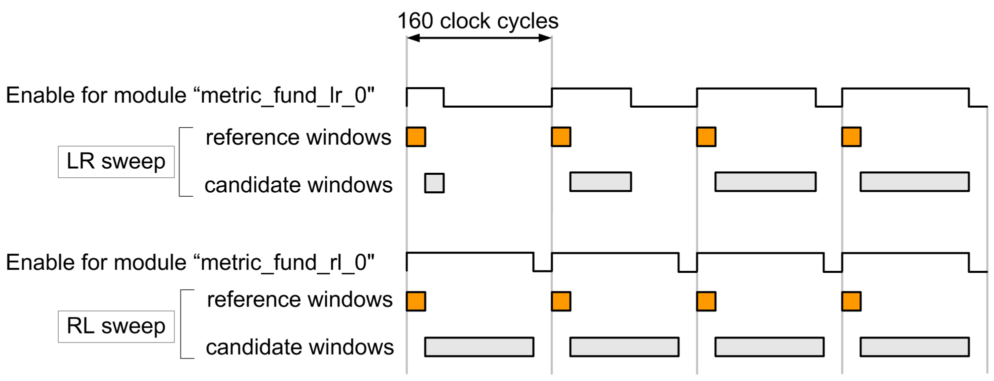


Figure 4.25: Control signals for the “metric_fund” modules near the image edges.

4.3 Applying the threshold

The thresholding stage corresponds to a “binarization” of the disparity results: 1 if disparity is higher than the threshold, or 0 otherwise. In hardware terms, it boils down to the boolean result of a comparison. The threshold level is user-defined: a push button connected to one of the FPGA’s IO ports allows to increment the threshold from 0 to 130, on steps of 10. Recall that the maximum disparity attainable is 135.

Following the ETH Report lead, the two cost-functions results are also merged here: it suffices that one of the metrics, SAD or CENSUS, attributes the pixel a disparity higher than the threshold to be given a 1 in the output bitmap (or in other words, classified as a foreground pixel). It is, therefore, a OR operation, which will yield two bitmaps: one for the LR sweep and other for the RL sweep. Figure 4.26 exhibits the module's hardware struture.

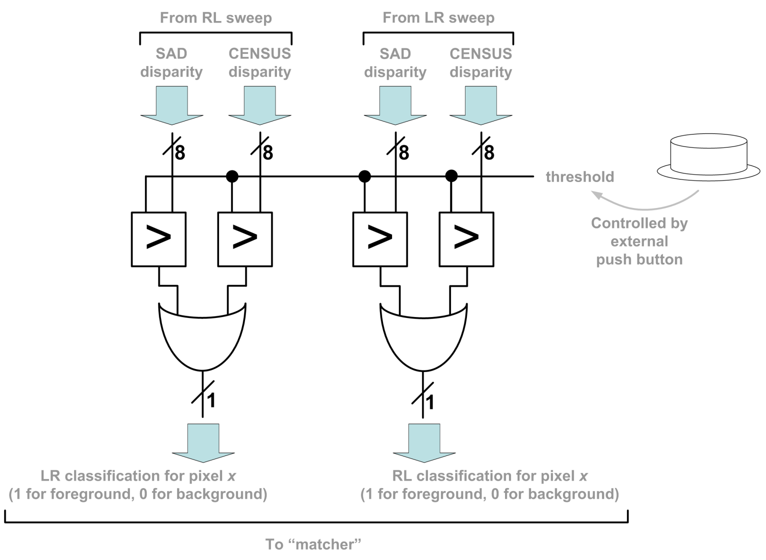

From RL sweep
SAD disparity
CENSUS disparity
From LR sweep
SAD disparity
CENSUS disparity
8
8
8
8
threshold
Controlled by external push button
>
>
>
>
1
1
LR classification for pixel *x* (1 for foreground, 0 for background)
RL classification for pixel *x* (1 for foreground, 0 for background)
To "matcher"

Figure 4.26: The hardware structure dedicated to thresholding and merging the 4 resulting disparity maps.

4.4 Matching the results

The module "Matcher" performs the matching between the two results of the area-match algorithm: one of the computations has the right image as the reference image and the left image as the candidate one (the RL sweep), and the other computation has the left image as the reference one, and the right image as the candidate image (the LR sweep).

Matching these two results helps to overcome the problem of occluded areas (areas visible in one of the images that are not in the other), as discussed in section 2.2.4. The same section also gives a more detailed introduction to the matching operation, but nevertheless a quick remainder follows: pick the pixel in position x from one of the results; it should have the value of either 0 or 1, according to if it was qualified as a foreground or a background pixel. Also consider its associated disparity, disp(x). Check if, in the other result, the pixel in position x+disp(x) was also considered a foreground pixel.

At this point, a few remarks. First of all, bear in mind as always that the final result of this merging is one single image. Also, either the SAD or the CENSUS disparity values can be used in this process. Finally, it should be pointed out that either the LR sweep as well as the RL sweep can serve as the "pivot" image; consistency is checked on the other result. However, considering this process is to be made "on-the-fly" (as results come in), it will be seen next that choosing the first option will considerably reduce hardware resources consumption, as consequence of one particular architecture option.

Look closely to figure 4.27. It presents the spatial relative location of the areas that are compared in both the sweeps. Each orange block corresponds to a 22-pixels wide per 3-pixels high block, equaling 20 3×3 reference matrices (remember each matrix results of a shift of one pixel with respect to the previous matrix). The white blocks refer to the candidate area searched for those particular reference matrices, and the blue arrows represent the direction of the shifts of the candidate windows.

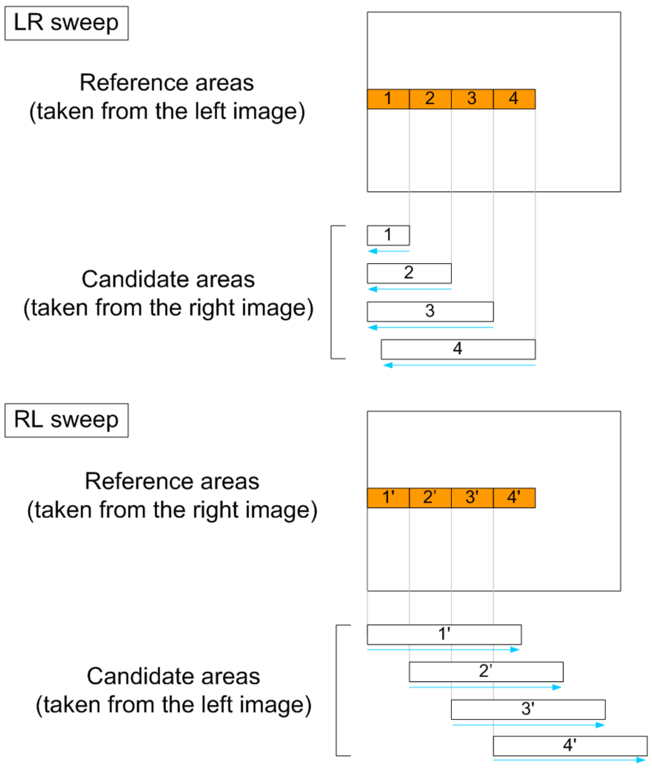

LR sweep
Reference areas
(taken from the left image)
1
2
3
4
Candidate areas
(taken from the right image)
1
2
3
4
RL sweep
Reference areas
(taken from the right image)
1'
2'
3'
4'
Candidate areas
(taken from the left image)
1'
2'
3'
4'

Figure 4.27: Candidate areas swept for best match, according to the location of the reference windows.

Figure 4.28, on its turn, presents the chronological evolution of how results are produced, for the two sweeps. Focusing attention on the LR sweep timing, one sees that the first search area for the first set of 20 reference matrices is considerably small: because the scan in the LR sweep is made from the right to the left, the image border is soon found. This also means the disparity values for these pixels must be smaller than 20 or, in other words, any pixel that could match them in the right image is to be found in the first 20 positions, starting in the edge of the image. Incidentally, at the end of the first 160 cycles, the first 20 reference pixels of the **RL** sweep have been also classified.

This means one can start to perform the consistency check *if* using the LR sweep as the pivot image: pick a pixel of those first 20 pixels from the LR area-match output, pick also its disparity, and check on the RL sweep if the pixel in the location given by the position of the LR pixel plus

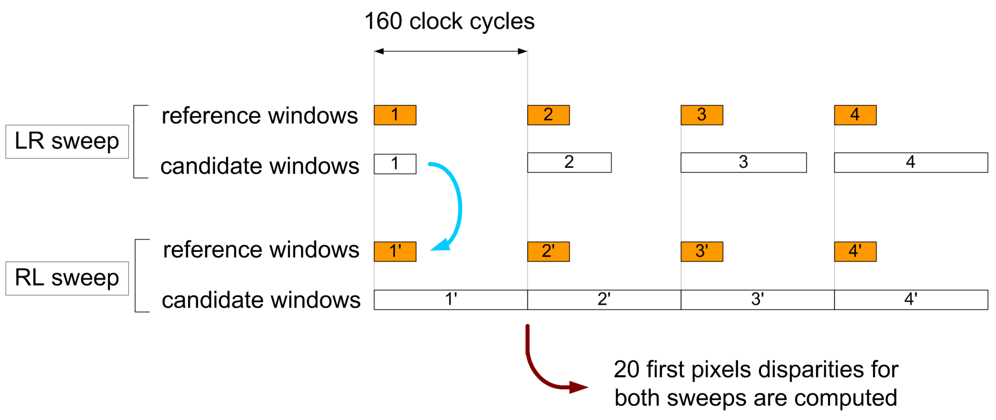


Figure 4.28: Temporal version of the previous figure 4.27. This figure shows how the candidate areas are swept in temporal terms, according to how the algorithm was implemented.

its disparity was also considered a foreground pixel. The blue arrow of figure 4.28 indicates that: because the candidate area for the LR sweep matches the same locations of the reference pixels of the RL sweep, one can start verifying if any of those pixels were considered foreground by both sweeps.

If one was to use the RL sweep as a pivot, computations would only start much later. The results for the first 20 reference pixels comprehend disparities that can go up to 160 pixels to right; however, at the same instant, only 20 reference pixels of the LR sweep have been classified. Therefore, no match can be made yet. The full first 160 pixels of the LR sweep will have to be classified and stored before any matching can begin, which will take at least eight 160-cycles blocks. Similarly, also the RL area-match results would have to be stored. Figure 4.29 depicts the idea.

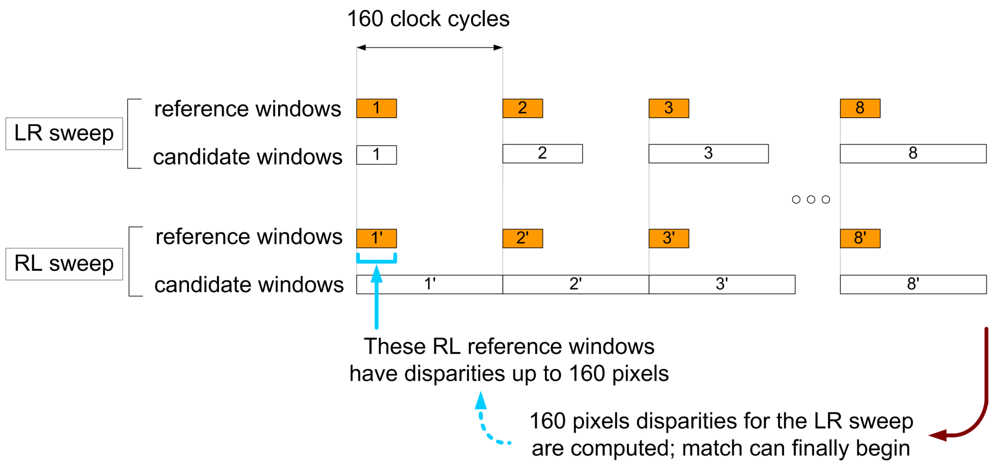

160 clock cycles
LR sweep
reference windows
candidate windows
RL sweep
reference windows
candidate windows
1
2
3
8
1'
2'
3'
8'
○ ○ ○
These RL reference windows have disparities up to 160 pixels
160 pixels disparities for the LR sweep are computed; match can finally begin

Figure 4.29: Alternative temporal analysis, where the right reference pixels are searched for on the left reference image.

From this discussion, the conclusion is drawn that using the LR sweep as the pivot image saves hardware resources. Recall that, in subsection 4.2.2, it was noted that a reason existed for re-using the RL sweep structure for the LR sweep operation (instead of performing a totally independent LR sweep control structure). The Matching stage, and the specific operation it features, explains why was there advantage in making such option. Also, the need for the 20 byte buffer (for temporary storage of 20 pixel disparities of the LR sweep) is hereby justified.

In terms of hardware, the matching itself is made by a structure similar to the one of figure 4.30. It should be noted that the disparity values that are used can be either the ones from CENSUS or from SAD. A external button allows to chose which one is to be used; normally, those which provide better pratical results are chosen.

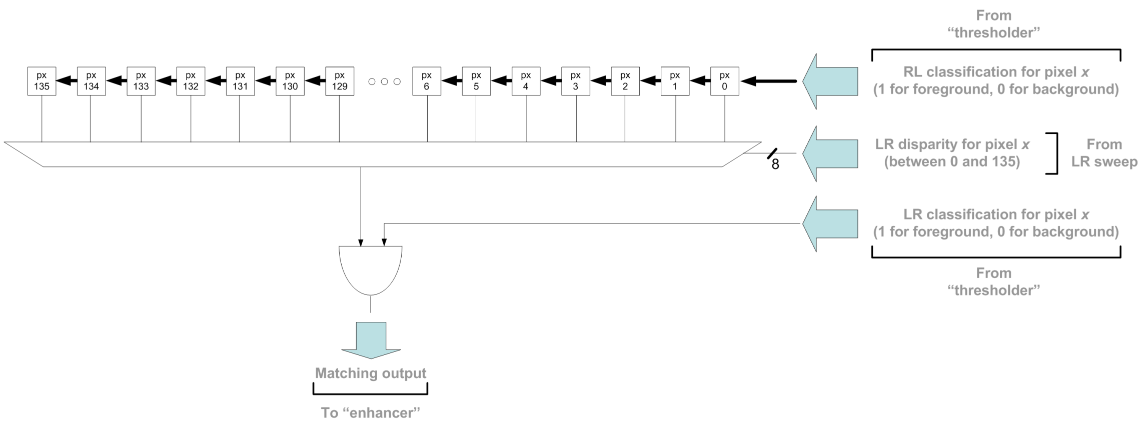

From "thresholder"
RL classification for pixel x (1 for foreground, 0 for background)
px 135
px 134
px 133
px 132
px 131
px 130
px 129
px 6
px 5
px 4
px 3
px 2
px 1
px 0
LR disparity for pixel x (between 0 and 135)
8
From LR sweep
LR classification for pixel x (1 for foreground, 0 for background)
From "thresholder"
Matching output
To "enhancer"

Figure 4.30: An AND operation is performed on both the results to validate which pixels truly belong to the foreground.

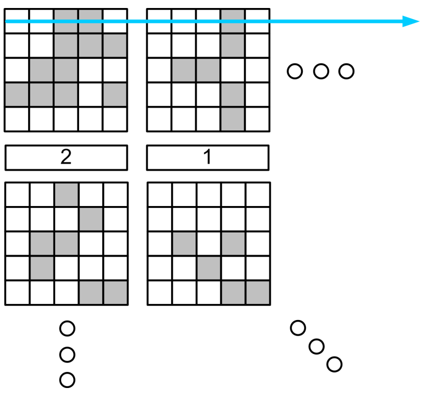
4.5 Post-processing

The post-processing stage has been introduced in the previous chapter 3. It consists of computing a more coarse-grained version of the 640×480 bitmap outputted by the matching stage. In this case, by considering blocks of 20×20 pixels, the resulting bitmap is reduced to a 32×24 matrix. The module associated with this operation is called "enhancer".

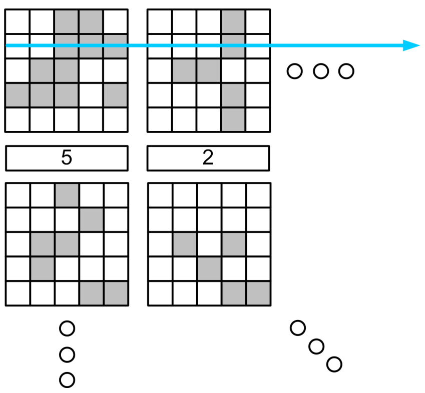
The hardware implementation is now described. First of all, bear in mind that the pixels come into the module one by one. 640 pixels account for a full line. Each incoming pixel is represented by one single bit: it either is a 1, meaning that pixel is a foreground pixel, either it is a 0, and thus it is a background pixel.

In order to count how many foreground pixels exist in one 20×20 pixels block, 32 registers of 9 bits are used ($640 \div 20 = 32$). Suppose a line of 640 pixels starts arriving, pixel by pixel, to the module. The first register stores the number of foreground pixels that exist between the 1st and the 20th pixel. Afterwards, from the 21st pixel up to the 40th incoming pixel, the second register stores the number of foreground pixels that were detected in that interval. The procedure continues until the 640 pixels have entered the module.

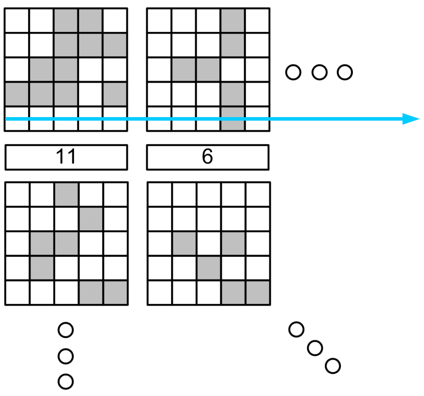
A new line of 640 pixels then arrives. The same operation takes place: the number of foreground pixels in the first 20 pixels is summed to the first register, the ones in the second 20-pixel interval summed to the second register, and so forth. The operation continues until the 20th line comes in. Note that, at this moment, each register contains how many foreground pixels exist in the first line of 20×20 pixels blocks. Therefore, the 32 registers are read sequentially, and compared with a threshold. This threshold is user-defined by a external push button, and sets how many foreground pixels must a block contain in order to be considered foreground. After being read, the 32 registers are reset, and the classification of the next line of 20×20 pixels blocks begins. Figure 4.31 depicts the full operation.



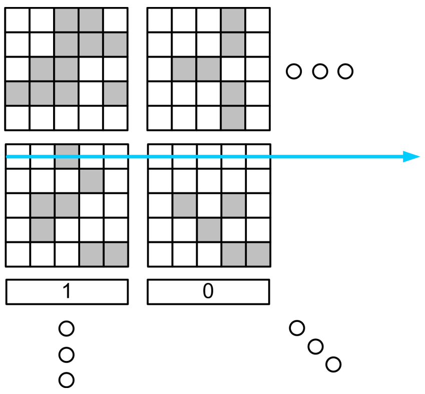
(a) Each foreground pixel increments the counter of the respective interval.



(b) Second line is being scanned. The same registers are updated.



(c) The 20th line arrives. The registers now contain the number of foreground pixels in the corresponding 20×20 block.



(d) The second line of 20×20 blocks starts being treated.

Figure 4.31: The coarsing operation. For simplification, blocks have 5×5 pixels instead of 20×20. The larger matrix is 32 blocks wide and 24 blocks tall; only 4 are represented.

If no further processing is done, as one of those presented in section 6.2, the inverse procedure must be performed to produce the module's results. The 32×24 bitmap experiences up-resolution to 640×480 by outputting 20 foreground pixels (1 bit each, high if it is foreground), over 20 lines, if the corresponding 20×20 pixels block was considered foreground. Again, recall that visual representation of the hand in the monitor will be severely distorted. Nevertheless, areas without enough foreground pixels were shutdown, where areas with high concentration of those were enhanced. This, ideally, will better the coordinate computation.

4.6 Computing the coordinates

The system must now compute of the center of gravity of the foreground pixels, and send them to the computer or any other receiving device. The procedure was explained in subsection 3.2.5. Each line or row has its foreground pixels summed, and the total is multiplied by the line \ row index. The sum of all weighted line sums, divided by the total number of foregroud pixels, yields the Y coordinate; if it is the sum of all weighted row sums, the X coordinate is obtained.

As to the Y coordinate, a running total can be used: as pixels arrive, a line register is incremented if the incoming pixel is foreground. When a full line of pixels is received, the total is multiplied by the line index, and the result added to a global line sum. However, the X coordinate has an important difference in terms of implementation: considering the pixels arrive line by line, the total number of foreground pixels in one row can only be known at the end of the image. Registers for storing the temporary row sums are therefore needed, while the end of the image is not reached. When that happens, all row values are multiplied by the corresponding row index, summed up, and divided by the number of foreground pixels. Figure 4.32 depicts the idea. The divisions are then made by a Coregen IPCore module. A set of coordinates is produces on every frame.

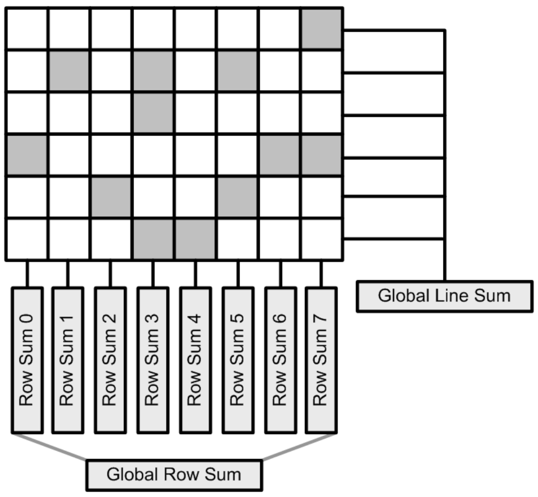


Figure 4.32: Required hardware registers for coordinate computation.

Afterwards, the data is sent to the computer through the serial port, by RS232 protocol. It is a serial byte-oriented digital transmission protocol, with simple procedures. The line is kept at a logical 1 while not being used; a start bit with the value 0 is sent to indicate a byte will follow it, and afterwards the line returns to the logical one. The used baudrate is 115200.

On the receiving computer, a “JAR” program is being run. Java was the chosen language for the simplicity of implementation it provided on the two ends: for one side, the RS232 communication port could be easily used, thanks to already-made implementations [19]. On the other side, Java is highly oriented for visual programming, and thus it was easy to find a function which would place the mouse’s cursor anywhere on screen. Any suplementary code is simple data-processing.

This being, one has a program that simply sets a thread. The later is waiting for an incoming-message signal and, should this happen, the thread starts to receive the data. Talking about the code specifics, one simply instantiates an InputStream object, and associates it with COMPort. The thread will be checking the stream of bytes given by that InputStream object, and will start the data-treatment routines, if any, as soon a byte different than 255 arrives. The program is expecting 4 bytes: the MSB and LSB of Y coordinates, and the MSB and LSB of the X coordinates. Afterwards, the mouse cursor is moved accordingly. The update rate is of one set of coordinates per frame (25 times a second).

Chapter 5

Project Outcomes

In this chapter, the PCB for supporting the cameras is initially presented. Afterwards, it is discussed the intermediate outcomes of the project, with help of videos. Finally, the FPGA's resource usage is analyzed.

5.1 Physical Outcomes

The first outcome of the project was a physical piece of hardware: the PCB board for the holding the cameras and interconnecting them with the XUP board. Other PCBs dedicated to stereoscopic setups had pin assignments incompatible with the Virtex 2-Pro board. On top of that, previous work wih the XUP had shown that some load problems were to be expected, which led to the decision of incorporating buffers in the PCB to prevent this problem.

The schematic, placement and routing of the support PCB were made in the Orcad designing environment, at FEUP's laboratories, as well as the physical implementation. When ready, the board did have some functioning problems at first. The main one was that the cameras' "vsync" signal were not synchronized. Should they have a fixed interval of time between them, and the problem could easily be fixed with a buffer. But the fact was that the signals had relative velocity, which meant the time interval between the two was continuosly changing. Eventually, it was found out that the clock provided to the cameras by the FPGA should **not** go through the buffers. This did solve the problem, but not before a relatively big amount of time was spent in analyzing the PCB.

This was a crucial stage of the project, as without the PCB the cameras and the FPGA could not communicate, and it was stressful for a time to watch the board not operating correctly. The custom-made PCB is presented figure 5.1: it is the one holding the cameras. Figure 5.1 also shows the global physical setup and the components disposition that is used in the attached videos.

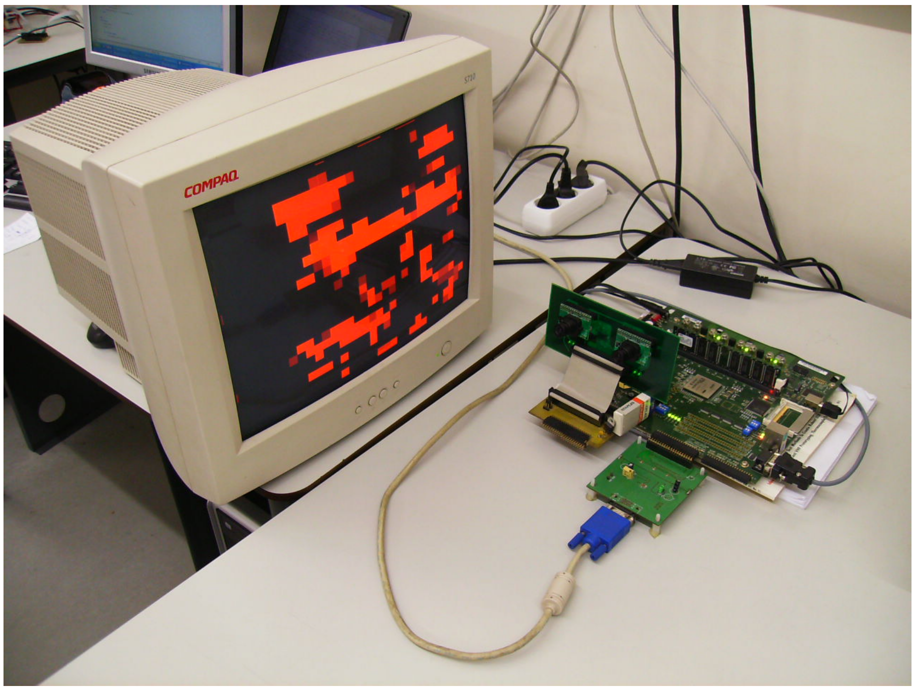


Figure 5.1: The set of physical components that make up the system.

5.2 Stage Outcomes

In order to discuss the intermediate results of the system and to replicate the real-time operation of the system, support videos were made. The following discussion comments those videos. The scene that is shown is the monitor in front of the video camera. The system's cameras are on the right side of the scene, close to the FPGA.

5.2.1 "Video 1"

In the first video, "video1", the four disparity maps are shown in the following order:

1. RL sweep SAD disparity map
2. LR sweep SAD disparity map
3. LR sweep CENSUS disparity map
4. RL sweep CENSUS disparity map

It can be noted, upon the first disparity map, that as the hand moves further away from the cameras, the hand gets smaller intensity values. This is the objective of the disparity map. It can also be noted the differences in the two cost-functions outputs: SAD tends to create more large coherent areas, whereas CENSUS tends to individualize edges and produce a "salt" effect (a large number of incorrectly classified pixels appears randomly).

It can be seen too a large quantity of objects that have an incorrectly assigned disparity: these are algorithms errors, and are associated with the cameras' small decalibration. The PCB that holds the cameras does so incorrectly, and consequently an incorrect disparity for certain areas of the scene is computed. It has been seen before that the algorithm operates horizontally, so if one of the images is shifted upwards or downwards, or even rotated, invalid matchs will happen. The solution to this problem is either to physically adjust the PCB, or implement a rectification algorithm prior to the system's operation.

Also, in the video it is not very visible, but there is also considerable flicker in some zones. These are normally low textured areas, where very slight variations in the pixels intensity values captured by the cameras often implies some temporal incoherence. Work done in [15] seems to confirm this idea.

Finally, as a secondary remark, it is mentioned these results have little brightness. Recall that a disparity map presents the disparity that was assigned to a pixel of the reference image (when that pixel was searched in candidate image) in a visual manner. The intensity of a pixel in the disparity map is proportional to the disparity it was assigned. Considering the maximum disparity range is only 135 pixel positions, the maximum luminance value is also 135, and thus the overall image luminance is low. Doubling the value would exceed the pixels' maximum intensity value: 255 (8 bits per pixel).

5.2.2 "Video 2"

On the following video, "video2", the output of the thresholding operation for two of the four disparity maps is shown. The order is as follows:

1. Thresholded RL sweep SAD disparity map

2. Thresholded RL sweep CENSUS disparity map

One will notice, in the first thresholded disparity map, that it is much more vivid, as now luminance values do not have to be limited to 135. In fact, what one has now is a bitmap. Also, the full monitor is lit up. This is because the disparity threshold is set very low: initially at 0. Almost all pixels have a disparity higher than 0, so most of them apper. When the push button is pressed, the threshold goes up by 10 units at a time; one can see the number of lighted pixels gradually becoming smaller, until only relevant foreground areas are present. Naturally, a compromise must be struck: having a very high threshold will discard noisy areas, but also the hand will be eroded.

On these results, note that, when the hand gets in front of the cameras, it is clear that it is the object with the larger amount of active pixels, thus confirming the idea that focal planes can be differentiated. But it can be seen too that considerable noise remains, due to some of the algorithm's misjudgements. Paralelly, the flicker is much more noticeable here than in the first set of results.

Afterwards, the CENSUS and SAD results of each sweep are merged: each pixel of the bitmap is ORed with the other metric's bitmap. The four bitmaps turn in this way into only two. They will be later matched into one single bitmap, by the module "matcher". Such is shown in "video3".

5.2.3 "Video 3"

In "video3", the following sequence is shown:

1. Matched bitmaps using SAD disparities
2. Matched bitmaps using CENSUS disparities
3. The output of the post-processing stage

The difference between using the SAD and the CENSUS disparity values is clear: SAD allows much more noise, thus making it more unreliable. The disparity threshold could be adjusted to disregard some of these areas, but recall that the hand will also lose sharpness. CENSUS, however, does not consider wide coherent areas; mainly edges can be seen. Although the foreground areas lose a considerable amount of pixels, it is more robust.

In the third and last output, the post-processing stage is presented. It is performed over the matching output that uses CENSUS, and therefore it should be compared with it, in order to assess its contribution. Initially, the whole screen is lit up. This is due to the second threshold: the number of foreground pixels that a 20×20 block must have to be fully lit up. The threshold is inititally at zero, and due to the "salt" effect of CENSUS, almost every area of the screen has a foreground pixel in it. As the threshold goes higher, one can see clearly that only hand-related zones are presented. The coarsing operation did effectively ehnanced the hand areas and discarded other areas with noisy pixels; but obviously, the hand shape was severely altered.

Finally, the mouse cursor moving around the computer screen was to be shown, but due to the video's low resolution it would not be seen, and therefore it was not presented. Nevertheless, the current situation is that the cursor does follow the hand correctly, although with a slightly jumpy behaviour. To better this, a low-pass filter could be implemented on the computer. It simply stores the last frames coordinates and compares it with the most recent; if the difference on some of the coordinates is too high, they will be atenuated, in order to place the cursor closer to the last position.

5.3 FPGA occupation

The following table 5.1 shows the system's resource consumption, according to component. These values concern the minimum for system operation. The system's operating frequency is of 100MHz.

The number of used flip-flops does seem reasonable. Many of them are consumed by the 40 "metric_fund" modules. Consider, from this module's internal structure, only the registers needed for holding the reference window, and also the registers for holding the nine absolute differences

Element	**Total available**	**Used**	**Ratio**
Slices	13,696	9565	69%
LUTs	27,392	16,847	39%
Flip-flops	27,392	10,936	61%
BRAMs	136	18	13%

Table 5.1: The resource usage by element.

(one for each pixel of the 3×3 matrix): 40 modules $\times$ 9 pixels $\times$ 2 sets of registers $\times$ 8 bits yields 5760, which is approximately half of the consumed flip-flops, as can be seen in the table 5.1. This simple math confirms the 40 “metric_fund” modules as the largest resource users, as a part of their architecture acounts for half of the used flip-flops. Considering the rest of the resources used by “metric_fund” modules and the remaining modules, the flip-flop consumption value is rather justified.

As to the LUTs, however, some effort has been put into optimizing their use, and approximate the number of used LUTs to that of the flip-flops. An interesting case of operation optimization was found in the SAD operation: the sum of absolute differences, performed on module “metric_fund”, was computed by an expression similar to:

```
diff <= (px > px_pivot) ? px - px_pivot : px_pivot - px;
```

where ’diff’ is an 8-bit register, ’px’ stands for a pixel (8 bits) of the candidate matrix, and ’px_pivot’ for a pixel of the reference matrix, that is stored in the module. The fact is that this line of code encloses 3 operations: the comparison, and the two subtractions. Such operations are implemented on LUTs, and therefore three sets of LUTs were used. If one considers that 8 more similar lines of code are used in the “metric_fund” module Verilog description, and that 40 such modules are synthesized, one can see many LUTs dedicated to this line of code. In fact, using this coding the ratios of consumption for flip-flops and LUTs were, respectively, 41% and 71%.

An alternative implementation was tried. Only one subtraction is performed, and if a overflow occurs, the 2’s-complement of the result is computed. Although the later operation is composed of inverting the result and adding 1, it only uses one set of LUTs. In consequence, LUT consumption dropped to the current value.

```
reg [7:0] diff;
wire [7:0] diff_aux;
(...)
diff <= px - px_pivot;
(...)
assign diff_aux = (diff[8] == 1) ? (~diff + 9'd1) : diff;
```

BRAMs are also considerably used in the system. Most of them are assigned to the “mem_3x3” modules. There are two of those, and each one has 4 memories, each one capable of storing a line of an image. In order to emulate two-read ports memory, the synthesizer attributes 2 BRAMs to each one of those “mem_3x3” line memories. Therefore, there are 2 modules $\times$ 4 line memories $\times$ 2 BRAMs = 16 BRAMs assigned to those two modules. As to the remaining 2 BRAMs, one is used in the post-processing stage, for storing the temporary number of foreground pixels in a 20×20 pixels block, and the other in the coordinates computation stage, for storing the temporary row totals.

Finally, and to close this discussion on the resource consumption, it should be stressed out the good scalabity capacity of this system. It can be achieved by adding more “metric_fund” modules, but that will only increase the achieved disparity. The main enhancement that could be made to the system operation would be to enlarge the window size. This can be easily done by adding the necessary BRAMs to the “mem_3x3” modules, according to how tall should the matrix be, and adding more registers to the “metric_fund” modules, in order to handle not 9 pixels, but more. In terms of other system characteristics, the only consequence is that the disparity will decrease accordingly to the window size. For example, if a 5×5 window is to be used, the “mem_3x3” modules will need 6 line memories each, instead of the current 4; modules “metric_fund” will need to handle 25 pixels; and disparity will drop from 135 to 133, to ensure the shift-register structure of modules “mem_fund”, discussed in section 4.1, gets fully filled before modules start using it.

As one can see, the system is easily scalable, and therefore could be implemented in other platforms and enlarged to fit their capacity with no significant difficulty.

Chapter 6

Conclusion

The last chapter is dedicated to drawing a final closure to the project, and proposing ideas for enhancing the system.

6.1 Conclusion

The presented system aims at providing a useful functionality, that will provide great ease of interfacing with machines. Being able to transmit information to a computer by simply moving your hand is a tremendous enhancement over traditional Human Computer Interfaces, like the mouse or the keyboard.

In order to do so, the system has to perform a computationally expensive operation. In fact, it computes 4 dense disparity maps, using two different cost-functions: two disparity maps use the Sum of Absolute Differences, and the other two the CENSUS transform. Also, two disparity maps are performed having the right image as the reference one, and the other two having the left image as the reference image. Afterwards, a threshold operation roughly individualizes the hand, and a consistency check is made to overcome the problem of occluded areas.

The system operates at this moment over images of 640×480 pixels, at a frame rate of 40 fps, achieving a disparity of 135 and using windows of 3×3 pixels. However, the architecture has great trade-off capabilities, allowing the same hardware structure for operating, for example, at 160 frames per second, over images 320×240, up to a disparity of 55. If the 40 fps frame rate is desired, then the disparity can reach 295 pixels under that resolution (which is an exagerate number, but serves to state the point). Under the 640×480 image dimension, increasing the maximum disparity simply boils down to use more "metric_fund" modules. In that sense, the architecture is highly scalable. Another important issue is the low memory consumption it implies, which turns the system quite suited for ASIC implementation.

On other level, this project also presents a new approach for the post-processing stage: a piramid method, that computes a smaller version of the 640×480 original output. This approach seems rather powerful, thanks to the low memory usage it implies and the wide range of optimization it allows, as stressed in the previous section 6.2. In fact, this technique can be used by any system that does not have a stringent requirement of identificating the objects of the scene. This is the case, for example, of the automotive industry, where the ability to detect an incoming object is more important than identifying it. In that sense, using a more coarse version of the initial representation helps to overcome some of the algorithm's underperformance.

Finally, it should be emphasized the work done on this project. Extensive study on image algorithms, as well as simulation tests, were performed in order to determine the system's optimal operational conditions. Also, considerable effort was spent in producing and putting to work the PCB board that supports the cameras. The system implementation in Verilog also took significant commitment, as well the variety of tests made to enhance the system's operation.

All thing considered, the project implements an interesting concept, which requires substantial knowledge on image processing algorithms, as well on hardware description languages and synthesis of digital circuits, and posed a reasonable level of difficulty in order to achieve all proposed objectives.

6.2 Future Work

Some ideas in order to better the system are now presented.

On what is concerned to the area-match algorithm, a more powerful cost-function could be used. For example, NCC, the Normalized Cross Correlation, is a more powerful metric than SAD. Although it uses divisions, methods can be used to efficiently perform this operation on a FPGA, namely by using a BRAM as a look-up table. Nevertheless, bear in mind that such operations would have to be implemented in *every* one of the 40 "metric_fund" modules, which can be quite resource-demanding.

Also, some ideas about noise reduction on the CENSUS and SAD core operation could enhance the system to a point where the post-processing stage, discussed in subsection 3.2.4, would be useless. These are presented in [20].

The use of edge detecting filters, like the Robert Cross operator, the Sobel operator, and the zero-crossing detector, could help the egde individualization. In particular, the zero-crossing detector results always in closed curves, which could be used to fill image areas known to belong to the foreground. More information about this can be found in [21].

Color segmentation of scene could be of help in order to individualize areas that have a constant color, and therefore provide a measure of confidence for the system operation.

Yet, if the idea of projecting the full 640×480 pixels image into a smaller 32×24 matrix was to be used, interesting applications could be divised. Several proposals based on the image coarsing method are now discussed in detail. Recall that a block comprises 20×20 pixels.

6.2.1 Coordinate computation simplification

The coordinate computation can be greatly simplified by the use of the lower-resolution version of the scene. One could compute the center of gravity not of the foreground pixels, but of the foreground blocks. This would yield a smaller division hardware module: the maximum argument would decrease from 27 bits (in case all pixels are foreground, the global sums are $480 \times 240 \times 640 = 73728000$) to 14 (all foreground blocks $24 \times 12 \times 32 = 9216$).

Naturally, a multiplication would still be needed in order to make the results compatible with a 640×480 display. However, if the used blocks would have dimensions that are a power of 2 (like 32×32 pixels, or 16×16, instead of 20×20), that multiplication would boil down to a simple shift.

Also, a considerable amount of memory is spent in storing the temporary number of foregroud pixels for each row, while the image end is not reached. In fact, this memory takes 640×9 bits, considering there are 640 rows and each can have up to 480 foreground pixels. It would be of convenience to downsize it; using the coarsing operation, a memory of 32×5 bits would suffice.

6.2.2 Noise reduction

Some noisy areas will erroneously activate the corresponding block; however, noisy areas tend to be rather sparse and normally small, which means noisy blocks will most times be in small groups. The major group of foreground blocks will ideally belong to the hand.

In order to enhance the overall performance, one more criterion could be tested before asserting a block as a foreground one: only if at least a few of its neighbouring blocks has also been considered as foreground, this one would also be so. Figure 6.1 depicts the idea.

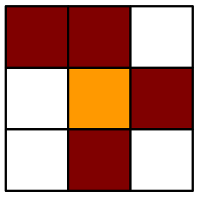


Figure 6.1: If at least 4 neighbours are lit up, so the center one can be.

The consequences of performing such thresholding operation are simple. As said before, noisy blocks normally do not have many neighbours, and therefore incorrectly activated blocks will be shutdown. Similarly, blocks with many “good” neighbours will also be considered foreground, thus enhancing the area of the hand.

Naturally, one exception happens: in the case of a large noisy area. Should it happen, and there will be an area where many blocks will be activated. When applying the “neighbour threshold”, the noisy blocks will remain activated and the noisy area will even be enhanced. It is up to other mechanisms to guarantee that no large noise areas are allowed at the inputs of this thresolding operation. In particular, the disparity threshold and the number of foreground pixels threshold are the best ones.

How is this to be done? The initial image is scanned and progressively its 20×20 pixels blocks will be classified. If one is to verify the 8 blocks surrounding each block (thus it being the center of a 3×3 blocks), one must store at least the result of 3 lines of the more-coarse grained version. Afterwards, one can start applying the “neighbour threshold” for the center block.

6.2.3 Enhancing overlooked areas

Another issue of the system is the tendency to overlook significant areas of the hand where textures are very alike (borders are normally more easily identified). This is due to the specific cost-functions used. A way around would be to use the previous framework, but also activate blocks that did not qualify on the number of foreground pixels, provided enough active neighbours are available. However, there is a question concerning this approach.

Assume the "neighbour threshold" operation is being performed with a 3×3 block matrix. In figure 6.2, the matrix under assessment is framed by the blue square. Consider that the center block does *not* respect the "number of foreground pixels" threshold. If there are at least 4 activated blocks in the matrix, the center one could also be lit up, producing the final result in the right side of the figure.

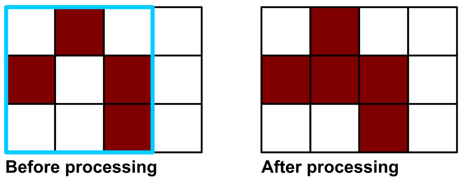


Figure 6.2: The matrix contains enough foreground blocks for the center one to be also considered as one, although it did not meet the "number of foreground pixels". The threshold is 4.

Assume now the verification matrix shifted one block to the right. A new center pixel is now being classified. Assume this block also does not comprehend enough foreground pixels to be classified as foreground block. The question here is that the information that the previous block was activated is not contemplated in this new computation.

This means that, if the block has only 3 activated neighbours, it will not be activated. That is the case of figure 6.3. It would be, however, if the updated information that its left neighbour had become activated had been used, as in figure 6.4.

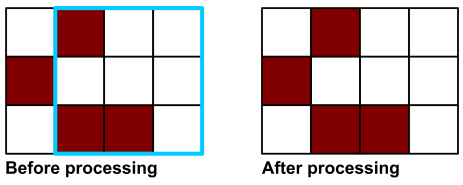


Figure 6.3: The next block is classified, but no updated information is used in its classification.

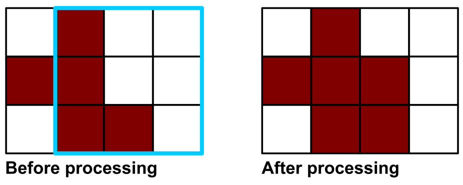


Figure 6.4: The classification is made using the updated information.

This new method, where the information is updated, requires the previous line of block classification to be stored, in order to use that information in the top line of the matrix. Considering one must simply store 32 bits, this is a reasonable requirement. However, the main problem resides in the third line of the matrix. Those blocks have not yet been classified. At this point, the information of the “number of foreground pixels” threshold alone could be used, but an uneveness remains between the first line, which has updated information, and the third line, which has not. A way to minimize this asymmetry is to use the information from the previous frame, that already has updated information, although belonging to the previous chapter. Figure 6.5 depicts the idea, where the blocks with a “s” (for “spatial”) are the recently classified blocks, and the ones with “t” are the ones from the previous frame (“t” for temporal).

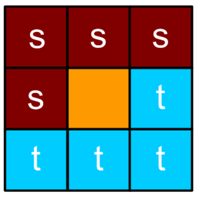


Figure 6.5: The purple blocks refer to pixels that have already been classified, whereas the blue ones refer to blocks from the previous frame.

Naturally, this option does have a stringent requirement: the whole 32x24 bits matrix, where each element represents a block, will have to be stored for each frame, in order to be used in the following frame for the “neighbour threshold” operation.

6.2.4 Temporal low-pass filter

If it would be possible, the storing of the matrix in the system would for more features. If each word of the memory can have more than one bit, then one can think of implementing a temporal low-pass filter.

Why is this necessary? A recorrent problem of the “number of foreground pixels” threshold operation output is that some 20×20 pixels blocks will flicker temporally, according to if, in one frame, that area was considered to be foreground, and in the consecutive frame it was not. This, naturally, happens when there are not enough foreground pixels in the 400 pixels that constitute a 20×20 pixels block, and there are in the following frames. Using the memory, an history for each block can be stored of how many times was that block classified as a foreground one in the last n frames.

This procedure will imply a temporal hystheresis: a block considered as foreground one will not be immediately activated if in the last few classifications it did not the match the “number of foreground pixels” threshold. Similarly, a block may not be immediately shutdown if the history shows that recently that block has been active for quite a few frames. The article [22] presents an analysis to this problem.

6.2.5 Identifying gestures

Being able to determine variations in the hand's distance to the cameras, as well as keeping a record of the hand's position, would allow for some interesting interactions. For example, in the case of a presentation, moving a hand closer and farther to the cameras could indicate the computer that switching between desktop windows was desired. If one would move his hand from left to right or vice-versa, then alternation between slides of the presentation could be implemented.

The presented representation coarsing would simplified this task, as a much lesser amount of information would need to be stored from frame to frame.

6.2.6 Recognizing the hand's posture

To make the system able to determine the hand's posture would be useful for transmitting more information to a computer, and provide a visual feedback. A few examples: if one has one finger pointing, the system will only track your hand; if one closes his hand fully, the window on display might close; if one fully opens his hand, this can be interpreted as a "double-click".

This is a rather more ambitious goal than the one of simply identifying the hand's position. An idea would be to, from the larger 32×24 matrix representation, pick up the small matrix (for example of 8×6 blocks) corresponding to the location where one expected the hand to be. Afterwards, that 8×6 grid could be transformed into a codeword (remember each element of the matrix is a bit). This codeword would serve as an address for a database entry, where the codeword would be matched into a posture. Figure 6.6 shows an example.

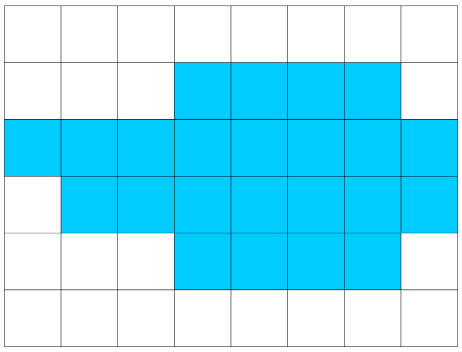


Figure 6.6: This matrix would be turned into a codeword and an associated posture would be searched for in a database. In this case, a open hand can be infered.

The database would, naturally, be made off-line and only once, and then stored in the system. However, 8×6 blocks means one would need $2^{8\times6}$ entries for that database. A more interesting implementation, if the resources would be available, would be to implement a alread trained neural network to identify the hand's postures. The article [23] performs something similar. A proposal is made on figure 6.7.

Another issue is how to locate the hand, in order to know which 8×6 matrix is to be considered. This can be done by implementing also the previous proposal: gesture identification.

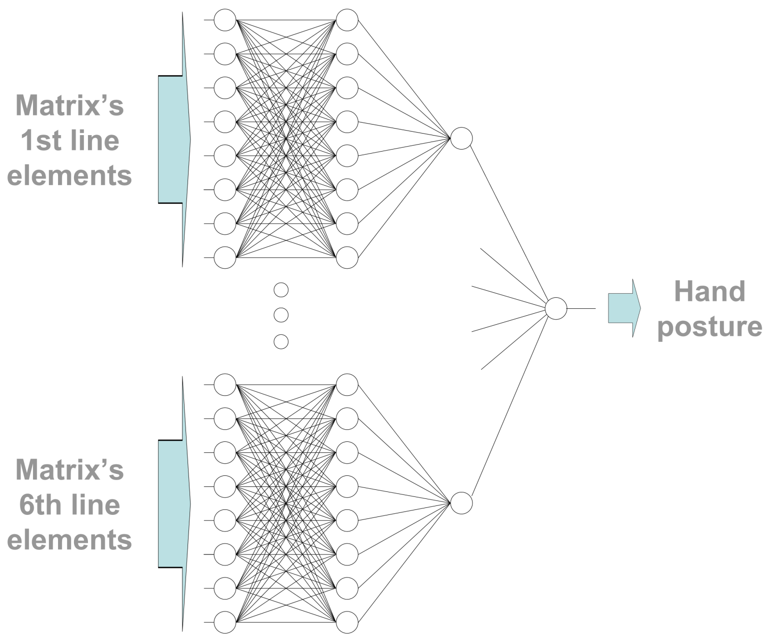


Figure 6.7: A neural network having as inputs the elements of a 8×6 matrix.

References

[1] Patrick de la Hamette and Gerhard Tröster. Architecture and applications of the FingerMouse: a smart stereo camera for wearable computing HCI. *Pers Ubiquit Comput*, 2006.

[2] Gerard de Haan. *Video Processing for Multimedia Systems*. Eindhoven Sep. 2000.

[3] P. de la Hamette, P. Lukowicz, and G. Troster. Fingermouse: A Wearable Hand Tracking System. *UBICOMP conference (proceedings)*, 2002.

[4] Patrick de la Hamette, Gerhard Tröster, and Marc von Waldkirch. Laser Triangulation as a means of robust Visual Input for Wearable Computers. *ISWC'04 Proceedings*, 2004.

[5] Julian Hebb, Thomas Koch, and Sven Kuonen. VLSI Implementation of the FingerMouse Algorithm. Master's thesis, Department of Information Technology and Electrical Engineering, Swiss Federal Institute of Technology Zurich, Winter Term 2004/2005.

[6] Focal lenght. `http://en.wikipedia.org/wiki/Focal_distance`.

[7] Tsukuba images. `http://cat.middlebury.edu/stereo/data.html`.

[8] Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. 2001.

[9] Antonios Gasteratos, Lazaros Nalpantidis, and Georgios Ch. Sirakoulis. Review of Stereo Matching Algortihms for 3D Vision. *16th International Symposium on Measurement and Control in Robotics (ISMCR).*

[10] Stephen T. Barnard and Martin A. Fischler. Computational Stereo. *ACM Comput. Surv.*, 14(4):553–572, 1982.

[11] N. W. Bergmann and R. B. Porter. A generic implementation framework for FPGA based stereo matching. *Proc. IEEE TENCON '97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications'*, 1997.

[12] J. Banks, M. Bennamoun, and P. Corke. Non-parametric techniques for fast and robust stereo matching. *Proc. IEEE TENCON '97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications'*, 1997.

[13] John Woodfill and Brian Von Herzen. Real-Time Stereo Vision on the PARTS Reconfigurable Computer.

[14] Luping An, Yunde Jia, Mingxiang Li, and Xiaoxun Zhang. A Miniature Stereo Vision Machine (MSVM-III) for Dense Disparity Mapping. *Proceedings of the 17th International Conference on Pattern Recognition (ICPR'04).*

[15] Kristian Ambrosch, Martin Humenberger, Wilfried Kubinger, and Andreas Steininger. SAD-based Stereo Matching using FPGAs. *Chapter 6 of Embedded Computer Vision - Advances in Pattern Recognition Series, Springer Verlag, London, 2008*, Oct. 2008.

[16] C. Murphy, D. Lindquist, A.M. Rynning, T. Cecil, S. Leavitt, and M.L. Chang. Low-Cost Stereo Vision on an FPGA. *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*, pages Page(s):333 – 334, 23-25 April 2007.

[17] Richard Carrillo, Javier Díaz, Alberto Prieto, and Eduardo Ros. Real-Time System for High-Image Resolution Disparity Estimation. *IEEE TRANSACTIONS ON IMAGE PROCESSING*, 16, 2007.

[18] OV7620 Single-chip CMOS VGA Color Digital Camera / OV7120 Single-chip CMOS VGA B&W Digital Camera Datasheet. Technical report, OmniVision.

[19] Sebastian Kuligowski. RS232 in Java for Windows. `http://www.kuligowski.pl/java/rs232-in-java-for-windows,1`.

[20] João Rodrigues. Implementação de Rectificação de Imagens Estéreo num Sistema Embutido Baseado em FPGA. Master's thesis, Faculdade de Engenharia da Universidade do Porto, June, 2008/2009.

[21] R. Fisher, S. Perkins, A. Walker, and E. Wolfart. Feature Detectors, 2003. `http://homepages.inf.ed.ac.uk/rbf/HIPR2/featops.htm`.

[22] D. Piccinini. Temporal filtering of disparity measurements. In *ICIAP '01: Proceedings of the 11th International Conference on Image Analysis and Processing*, page 145, Washington, DC, USA, 2001. IEEE Computer Society.

[23] Kouichi Murakami and Hitomi Taguchi. Gesture recognition using recurrent neural networks. *CHI '91: Proceedings of the SIGCHI conference on Human factors in computing systems*, 1991.