GESTURE RECOGNITION SYSTEM BASED ON STEREO VISION AND IMPLEMENTED IN FPGA WITH APPLICATION TO HUMAN-ROBOT INTERACTION

JUAN DAVID GUERRERO BALAGUERA

PEDAGOGICAL AND TECHNOLOGICAL UNIVERSITY OF COLOMBIA
SOGAMOSO SECTIONAL HEADQUARTERS
MASTER'S DEGREE IN ENGINEERING
SOGAMOSO
2017

GESTURE RECOGNITION SYSTEM BASED ON STEREO VISION AND IMPLEMENTED IN FPGA WITH APPLICATION TO HUMAN-ROBOT INTERACTION

JUAN DAVID GUERRERO BALAGUERA

Degree work to opt for the master's degree in engineering with an emphasis in electronic engineering

Thesis Director
Eng. WILSON JAVIER PÉREZ HOLGUÍN, PhD

PEDAGOGICAL AND TECHNOLOGICAL UNIVERSITY OF COLOMBIA
SOGAMOSO SECTIONAL HEADQUARTERS
MASTER'S DEGREE IN ENGINEERING
SOGAMOSO
2017

Acceptance Note

President of the Jury

Jury

Jury

Sogamoso, February 13, 2017

Write your dedication here

Example:

Thanks

Express your thanks here (It can be a paragraph or two but no more than one page).

Summary

Hand gesture recognition is used in a wide variety of human-computer interaction (HCI) applications. Among the most outstanding are: sign language recognition, virtual reality, control of television and video game sets, video surveillance and human-robot interaction (HRI), etc. The interaction of humans with robots is a topic that has been addressed by science fiction and that has transcended reality thanks to the technological advancement of science and technology in recent years. Research in this area seeks to improve communication between humans and robots so that interaction develops more naturally. The recognition of hand gestures based on artificial vision is the tool currently used to establish communication with robots. However, there is a need to communicate increasingly complex information using a greater number of gestures. This causes an increase in the time required to process information and low recognition rates, which is unacceptable in real-time gesture recognition applications.

Field-programmable logic arrays (FPGAs) are a good alternative to conventional computing systems for the development of vision-based hand gesture recognition applications, as they allow the implementation of high-performance parallel processing architectures and their power consumption is significantly lower than other platforms such as GPUs. This paper presents the development of a hand gesture recognition system based on stereo vision and implemented in FPGA, designed to teleoperate a mobile robot through seven hand gestures. The system is composed of four main stages: *i*) capture, *ii*) pre-processing, *iii*) feature extraction, and *iv*) recognition. The capture of gestures is done by a stereoscopic system of cameras. In the pre-processing stage, the image is extracted from the hand using a process of segmentation by skin color and the disparity map. In the feature extraction stage, the orientation of the hand path is used to describe each gesture. In the recognition stage, a classifier based on Hidden Markov Models (HMM) is used.

The capture, pre-processing and feature extraction stages were developed in VHDL and synthesized in an Altera Cyclone 5 SoC FPGA-SoC device, using the DE1-SoC development board. The reconnaissance stage is implemented in software using the HPS of the FPGA-SoC device. The system is capable of processing stereo images at a pixel rate of 100MHz with a processing speed of approximately 307fps. It is worth noting that the designed system has a gesture recognition rate of more than 99%.

Keywords: Computer Vision, Gesture Recognition, Human-Computer Interaction (HCI), Human-Robot Interaction (HRI), VHDL, FPGA, HMM.

Content

List of Tablas

Page

List of Figures

Page

List of Attachments

List of Abbreviations

HCI	Human-Computer Interaction
HRI	Human-Robot Interaction
Great	Time of Flight
FPGA	Field Programmable Gate Array
SVM	Support Vector Machine
HMM	Hidden Markov Model
ANN	Artificial Neural Network
ANMM	Average Neighborhood Margin Maximization
LDCRFs	Latent-Dynamic Conditional Random Fields
KNN	K-Nearest Neighbors
FEMD	Finger-Earth Mover's Distance
KING	Region of Interest
PCA	Principal Component Analisys
DTW	Dynamic Time Warping
P2-DHMMs	Pseudo two dimension hidden Markov models
PSO	Particle Swarm Optimization
DHMM	Discrete Hidden Markov Model
FSM	Finite State Machine
SW	Software
HW	Hardware
FPGA	Field Programmable Gate Array
VHDL	VHSIC (Very High Speed Integrated Circuit) Hardware Description Language

1 Introduction

Gestures are expressive body movements that involve physical movements of the fingers, hands, arms, head, face, or body, with the intention of conveying meaningful information or interacting with the environment (Mitra & Acharya, 2007). Formally a gesture can be defined as a spatio-temporal pattern that can be static, dynamic or both. Hand gestures are a powerful channel of communication between people, which are an important part of the transfer of information in our daily lives (M. O. S. M. Elmezain, 2010).

Since the emergence of computers, there has been a need for people to be able to interact with this type of machine in a more natural way. Until now, the most common systems used to interact with computers are the keyboard and mouse. *Human-Computer Interaction* (HCI) is a field of research that seeks to develop new methodologies and techniques to improve human-computer interaction. In this area of research, the recognition of hand gestures plays an important role, as it allows the development of more comfortable interaction systems between people and computers. Currently, vision-based gesture recognition is a very active field of research in the field of HCI, with great challenges in the development of systems that allow interaction between humans and computers more naturally (M. O. S. M. Elmezain, 2010).

Vision-based hand gesture recognition has quite a few applications in the field of HCI research. These include: sign language recognition, virtual reality, control of television sets and video games, video surveillance and human-robot interaction (HRI). HRI is a fairly broad multidisciplinary field of research that aims to build communication interfaces to interact easily and intuitively with a robot. Vision-based gesture recognition is most often used in HRI applications, surpassing other communication methods such as sound or touch. In gesture recognition, current approaches try to employ more diverse gestures to interact with robots. However, this implies problems of high processing time and low recognition rate (Nguyen-Duc-Thanh, Lee, & Kim, 2012).

Continuous technological advancement has allowed the emergence of technologies to acquire and process gestures in real time. However, the vast majority of processing systems are based on conventional computing architectures, or in the best of cases, parallel processing devices such as GPUs are used. Although these systems adequately solve the processing time problem mentioned above, they have the disadvantage of having a high consumption of electrical energy and requiring a permanent connection to a computer. This greatly restricts their use in HRI applications (as the autonomy of robots depends on the ability of their batteries to store energy), as well as considerably limiting the portability of the system required. On the other hand, *Field Programmable Gate Array* (FPGA) has become very important in recent years in the development of applications that require high

processing capacity and low power consumption, which is why they have become an excellent alternative for the development of HRI-oriented gesture recognition systems.

This paper presents the design and implementation in FPGA of a gesture recognition system based on stereo vision, oriented to HRI. The recognition system is made up of four main stages: *i*) capture of gestures, *ii*) pre-processing, *iii*) extraction of characteristics and *iv*) recognition. The system performs real-time stereo image capture and processing by designing a hardware architecture that enables high-performance parallel processing. The recognition system is designed to teleoperate a robot that can perform seven basic actions: forward, backward, left turn, right turn, 90° left, 90° right rotation and stop. The proposed system was synthesized in Altera's Cyclone V 5CSEMA5F31C6 FPGA-SoC device, which is available on the DE1-SoC development board.

The rest of this document is organized as follows: Chapter 2 presents the literature review, in which each of the stages that make up a typical gesture recognition system is discussed. Chapter 3 presents some fundamental concepts about image processing, stereo vision, parallelism, and algorithm mapping in FPGAs. Chapter 4 provides a detailed description of the recognition system developed. Chapter 5 describes the proposed hardware development for capturing and synchronizing stereoscopic images. Chapter 6 shows the design and implementation in hardware of the pre-processing stage of the proposed recognition system. Chapter 7 discusses the features used to describe a gesture, as well as the hardware implementation of these features. Chapter 8 presents the design of the classifier used in the recognition stage. Chapter 9 analyses the results obtained and Chapter 10 details the conclusions and future work.

2 Literature Review

Human motion analysis has been a very active area of research in computer vision, whose main objective is to segment, capture and recognize human movement automatically and in real time, and even predict subsequent human actions. This research topic can be widely applied to various areas, such as surveillance and security of public spaces, such as shopping malls, parks, public ground transportation stations, and airports. On the other hand, automatic human motion analysis can be used in Human-Computer Interaction and Human-Robot Interaction (HCI/HRI) applications, virtual reality, video games, medicine, among others (L. Chen, Wei, & Ferryman, 2013; Premaratne, 2014).

The recognition of hand gestures is a broad research topic, in which different analysis techniques have been developed to automatically capture, segment and recognize these gestures. There is a considerable amount of work developed on this topic, which presents different techniques and methods for its modeling, analysis and recognition (Mitra & Acharya, 2007; Premaratne, 2014).

In the work on hand gesture recognition presented by (Moni & Shawkat Ali, 2009), (Suarez & Murphy, 2012) and (L. Chen et al., 2013), the stages that make up a gesture recognition system are exposed, such as: image acquisition, pre-processing, feature extraction and recognition (L. Chen et al., 2013; Moni & Shawkat Ali, 2009; Premaratne, 2014; Suarez & Murphy, 2012). Figure 1 shows a block diagram that represents the flow of information through each of the stages.

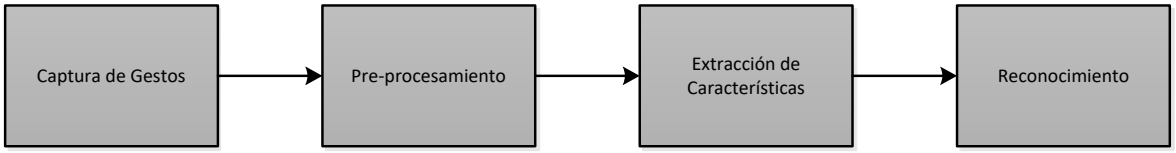


Figure 1. Stages that make up a system of recognition of hand gestures. Source: author.

2.1 Hand Gesture Capture

The first stage of gesture recognition is the capture stage. In this you acquire the gestures that are going to be the object of analysis for a given application. In the literature, there are multiple methods of capturing gestures, among which the use of instrumented gloves stands out (L. Chen et al., 2013). The use of gloves for capturing gestures dates back to the 70s with the development of the first prototype called *Sayre Glove* (Premaratne, 2014). During the last 20 years, technological advances have allowed the development of this type of instrument for capturing gestures whose applications range from video games to 3D movie animation. Some of these developed commercial instruments are: *ZTMGlove*, *MIT Data Glove*, *CyberGlove* II, *CyberGlove* III, 5DT *Data Globe Ultra*, X-IST *Data Glove* and P5 *Glove* (Premaratne, 2014).

Although instrumented gloves are tools with great potential in the development of hand gesture recognition applications, they tend to suffer mechanical wear and tear causing measurements to be affected over time. In addition, the use of gloves does not allow the user to interact naturally with the final application because it is an accessory that must be worn on the body (Mitra & Acharya, 2007). In contrast, vision-based methods allow the user to interact more naturally without the need for additional accessories. However, this method of capture involves other types of demands such as hand detection, trajectory tracking, occlusions of body parts and mainly the need for greater computational calculation (Mitra & Acharya, 2007).

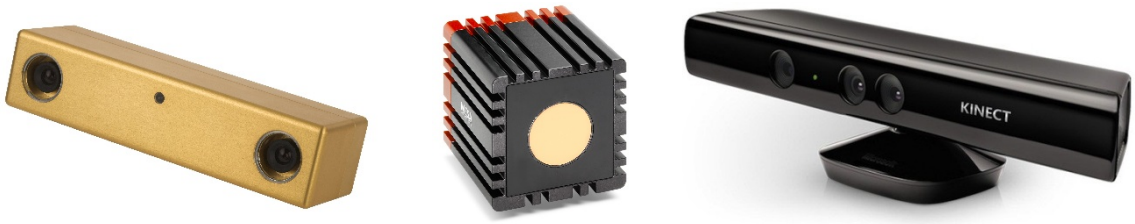
To capture gestures using vision techniques, digital images are used which contain a lot of information related to the environment. However, such images are sensitive to changes in lighting and tasks that include background subtraction are difficult to perform robustly (Aggarwal & Ryoo, 2011; L. Chen et al., 2013; Ji & Liu, 2010; Moeslund, Hilton, & Krüger, 2006; Poppe, 2010; Weinland, Ronfard, & Boyer, 2011). On the other hand, depth images have greater advantages over intensity images. First of all, these images have very good invariability against changes in lighting. And secondly, they provide a three-dimensional structure of the scenes, which facilitates the tasks of background subtraction, segmentation and motion estimation (L. Chen et al., 2013).

Capturing gestures using depth images is done using depth sensors. There are currently three main technologies for performing this type of sensing: stereo cameras, *ToF* (*Time of Flight*) cameras, and structured light (L. Chen et al., 2013; Moni & Shawkat Ali, 2009; Poppe, 2010; Suarez & Murphy, 2012; Weinland et al., 2011). Figure 2 shows the three types of depth sensors. On the left is the *bumblebee* stereo vision system, in the center the *Mesa Imaging* camera based on the *Swiss Ranger* SR4000 ToF sensor, and on the right, the Microsoft Kinect system is presented. Table 1 compares the main characteristics of the three depth sensing methods. Each of these methods of capture presents difficulties. For example, structural light systems only work well in closed environments and are not suitable for mobile app development; ToF-based systems are expensive and the images captured are of low resolutions; stereoscopic cameras rely on ambient lighting requiring effort in pre-processing the images. However, this model can be used in mobile applications using conventional cameras (L. Chen et al., 2013).

Table 1. Comparison of depth sensors (L. Chen et al., 2013)

Sensor Type	**Stereo cameras**	**Time of Flight**	**Structured light**
Resolution	Discharge. 640X480 the mas	Casualty. 64X48 to 200X200	640X480
Velocity	It depends on the speed of the cameras.	Fast	Fast
Rank	Limited by baseline	5m to 10m	0.8m to 3.5m (enclosed spaces)
Depth Resolution	Depends on the baseline and resolution of the cameras	Less than 5mm	Less than 1cm
Field of view	Not limited. Depends on the camera lenses	43º vertical, 69º horizontal	43º vertical, 57º horizontal

Gaps in the Disparity Map	Yes	No	Yes
Price	Cheap	Costly	Cheap



(a) (b) (c)

Figure 2. Depth sensors. (a) Stereo camera (Pointgrey, 2016). (b) Time-of-Flight (ToF) sensor (SwissRanger SR4000 MESA Imaging AG). (c) structured light (Kinect). Source: author.

2.1.1 Stereo cameras

Stereoscopic vision is inspired by human vision. This type of vision obtains the three-dimensional information of the scene from two or more images taken from different points of view, in the same way as human stereo vision does (L. Chen et al., 2013). Obtaining the disparity map from stereo images is a research topic that has been developed since the 60s and has allowed the development of research products such as the *bumblebee system* of the company *PointGrey* (Pointgrey, 2016).

Calculating the disparity map from stereoscopic images is still a challenge due to the complexity involved in stereo geometry. In addition, stereo images are sensitive to changes in lighting which increases the complexity in obtaining stereo correspondence (Hartley & Zisserman, 2004). Due to these types of difficulties, the reconstruction of the disparity map in real time and in real-life applications is still a challenge for researchers in the area (L. Chen et al., 2013). Currently, the use of reconfigurable hardware technologies such as FPGAs provides an alternative solution to the drawbacks of this type of vision systems (Greisen, Heinzle, Gross, & Burg, 2011).

Real-time processing of stereo vision systems is a difficult activity to satisfy when using conventional computer systems. For this reason, the scientific community has tackled this task through the use of reconfigurable logic systems such as FPGAs and ASICs, as well as through graphics processing units such as GPUs (Greisen et al., 2011).

Among the most relevant works in this area are, on the one hand, the development of processing techniques for FPGAS such as the implementation of algorithms based on pipeline architectures and systolic arrays that use matrices of basic processing elements, and on the other hand, the methods for the calculation of disparity maps such as SSDs (*sum of squared differences*). SAD (*sum of Absolute differences*), NCC (*normalized cross correlation*), among others. (Banz, Hesselbarth, Flatt, Blume, & Pirsch, 2010; Bendaoudi &

Khouas, 2013; Ding, Du, Wang, & Liu, 2010; Gudis, Wal, Kuthirummal, & Chai, 2012; Ibarra-Manzano, Almanza-Ojeda, Devy, Boizard, & Fourniols, 2009; Isakova, Başak, & Sönmez, 2012; Ohmura, Mitamura, Takauji, & Kaneko, 2010; Sungchan Park & Jeong, 2007; Woodfill, Gordon, Jurasek, Brown, & Buck, 2006).

In (Banz et al., 2010) the development of a system to generate disparity maps of VGA images of 640X480 pixels, with a range of 128 pixels of disparity, operating under real conditions at a rate of 30 fps (*frames per second*) and a frequency of 39MHz, and implemented in a Virtex 5 FPGA device, is proposed. In (Ding et al., 2010) the implementation of an algorithm based on the NCC method for the calculation of stereo image disparity of 512x512 pixels at 70 fps, implemented in an FPGA device, is presented. In (Ibarra-Manzano et al., 2009) the implementation in reconfigurable hardware of a dense esterero vision algorithm based on the Census transform is presented. In (Shan et al., 2012) the SAD method is presented by which images of 1280x1024 are processed at 23 fps stereo to obtain the disparity map in real time in an FPGA. In the work carried out by (Fife & Archibald, 2013), the implementation in FPGA of an architecture with optimal use of resources is presented, which allows the calculation of the disparity map in real time of stereo images. For the development of this architecture, the dispersed Census transform was used. The implemented system has the ability to process stereo matching algorithms at a speed of 500MHz.

2.1.2 ToF (*Time of Flight*) Cameras

Although stereo vision in humans works very well, making an artificial system work in the same way requires enormous effort in developing techniques and algorithms suitable for that purpose. For this reason, more robust technological advances have been made, such as the development of the so-called *ToF* (*Time of flight*) cameras. Unlike stereo cameras, *ToF* cameras use a single camera and a light source that sends pulses to surfaces in the scene. These cameras measure the time it takes for pulses to reflect off surfaces. To calculate the distance at which objects are in a scene, the time measured by the camera and the speed of light are used (L. Chen et al., 2013).

Compared to other 3D laser scanning devices, TOF cameras are cheaper and smaller. Most commercial devices today use a sinusoidally modulated infrared light signal, and the distance is calculated using the offset of the signal reflected in a standard CMOS or CCD detector (Kolb, Barth, & Koch, 2008). The main advantages of *ToF* cameras are their high speed and the ability to get a dense disparity map that covers every pixel. The main practical drawback is their high price, although they are still cheaper than other 3D scanning devices. The most important technical drawback is its low resolution (L. Chen et al., 2013; Premaratne, 2014).

2.1.3 Structured Light Sensors

In 2010, Microsoft launched Kinect, a vision-based gesture sensing device, aimed at home use only (L. Chen et al., 2013). Kinect is composed of a conventional RGB camera and an infrared depth sensor (L. Chen et al., 2013; Premaratne, 2014). The camera and sensor together have a resolution of 640X480 pixels with a speed of 30 fps, and their measurement range is between 0.8m and 3.5m with a resolution of less than 1cm (L. Chen et al., 2013; Premaratne, 2014). However, depth perception only reaches a maximum resolution of 320X240 pixels (Premaratne, 2014).

Kinect calculates the disparity map using the structured light technique. Its operation is similar to stereo vision, but in this case one of the cameras is replaced by a light source, which generates a known pattern. The advantage of systems that use structured light over the *ToF* system is their low price, making them ideal for the development of everyday applications (L. Chen et al., 2013)

2.2 Pre-processing

Vision-based gesture capture systems allow you to obtain a lot of information about the environment, such as color, texture and dimensions of the objects present in the scene, etc. When a scene is captured using a vision system, it results in both desired and unwanted information. For example, if the vision system is focused on the hands of a person who can perform a gesture, the vision system also observes the other objects that are present in the scene which are unwanted information. Since the vision system is developed to respond to gestures, it must extract only the information it needs and discard information that is not useful. This process is known as pre-processing (Premaratne, 2014).

Focusing only on the desired information is quite a challenge for vision systems. In the case of hand gestures, identifying the different positions of the hand, in different gestures, with varied skin tones and with different lighting conditions, represents a rather difficult problem (Premaratne, 2014).

One of the biggest challenges in gesture recognition is the extraction of gesture alone in a complex scene. One of the widely used gesture extraction methods is skin color segmentation. The difficulty of recognition increases when the background contains regions with similar skin tones. To solve this problem, the disparity map is used together with skin color segmentation, in this way those objects that are far from the hands can be removed more easily (Premaratne, 2014).

One of the most widely used pre-processing methods in the literature is background removal in depth images. This method consists of segmenting the area of interest from the rest of the scene by restricting the distance. For example, for hand gesture recognition, a simple threshold allows you to segment your hands. Several systems use depth information to extract the foreground of the scene (Bellmore, Ptucha, & Savakis, 2011; Benko & Wilson, 2009; Bergh et al., 2011; Biswas & Basu, 2011; Breuer, Eckes, & Müller, 2007; C. P. Chen,

Chen, Lee, Tsai, & Lei, 2011; M Elmezain, Al-Hamadi, Appenrodt, & Michaelis, 2008; Fujimura & Liu, 2006; Grammalidis, Goussis, Troufakos, & Strintzis, 2001; Guomundsson, Larsen, Aanaes, Pardas, & Casas, 2008; Kollorz, Penne, Hornegger, & Barke, 2008; Kurakin, Zhang, & Liu, 2012; X. L. X. Liu & Fujimura, 2004; Mo & Neumann, 2006; Muñoz-Salinas, Medina-Carnicer, Madrid-Cuevas, & Carmona-Poyato, 2008; Nickel, Scemann, & Stiefelhagen, 2004; Oikonomidis, Kyriazis, & Argyros, 2011; C. B. Park & Lee, 2011; Ren, Meng, & Yuan, 2011; L A Schwarz, Mkhitaryan, Mateus, & Navab, 2011; Loren Arthur Schwarz, Mkhitaryan, Mateus, & Navab, 2012; Suryanarayan, Subramanian, & Mandalapu, 2010; Zhang & Tian, 2013).

Benko & Wilson, 2009 describes the construction of an interactive freehand surface system called *DepthTouch* to track hand gestures using a *ZSense* depth camera in which a 3D region of interest was chosen to segment the user's hands. In the works carried out by (Bergh & Gool, 2011; M Elmezain, Al-Hamadi, Appenrodt, et al., 2008; M Elmezain, Al-Hamadi, Pathan, & Michaelis, 2009; Mahmoud Elmezain & Al-Hamadi, 2012; Nickel et al., 2004; Nickel & Stiefelhagen, 2007; Oikonomidis et al., 2011; Yin & Xie, 2003) the hands are extracted from the images by combining skin color information and depth information.

In other works, the construction of a background model is described using only the images of the background, which are used to extract the shape of the human body. (Loren Arthur Schwarz et al., 2012) performs the simple subtraction of a static background from the scene. (Jansen, Temmermans, & Deklerck, 2007) model the background by averaging several *consecutive frames.* (Guomundsson et al., 2008) model the background using a Gaussian probabilistic model.

Because color images and disparity map information can be very noisy, the use of noise reduction techniques is necessary. Some methods used are the median filter and morphological operations (Breuer et al., 2007; Nickel et al., 2004; Nickel & Stiefelhagen, 2007; S Park, Yu, Kim, Kim, & Lee, 2012).

2.3 Feature Extraction

The purpose of the feature extraction stage is to extract information that allows the hand gestures to be classified properly. For example, if you have a portion of the image in which a human observer is performing the "Stop" gesture, then it is important to extract the information that allows the gesture to be unique compared to other gestures. The success of any sorting system lies in the development of unique and robust features. However, hand gestures can have some degree of variability. This makes the development of artificial systems that allow such variations to be properly interpreted not a trivial task. Therefore, a robust set of characteristics that uniquely describes a gesture must be developed in order to gain reliable recognition (Premaratne, 2014).

There are different approaches to extracting features that allow for proper classification, decreasing the number of false positives and negatives. Among the methods of feature extraction we can find: feature extraction based on gradient orientation histograms, feature extraction using invariant moments, feature extraction based on principal component analysis, among others (Premaratne, 2014).

The trajectory of a hand gesture is a spatio-temporal pattern formed with the centroids of the hand in a consecutive sequence of images. The trajectory of the gesture can be modeled based on characteristics such as the position, speed and orientation of the hand (C. P. Chen et al., 2011; M Elmezain, Al-Hamadi, Appenrodt, et al., 2008; M Elmezain et al., 2009; Mahmoud Elmezain & Al-Hamadi, 2012). In (M Elmezain, Al-Hamadi, Appenrodt, et al., 2008) a method of feature extraction is presented that consists of extracting the orientation of the gesture by calculating this parameter from two consecutive points belonging to the trajectory of said gesture. (M Elmezain et al., 2009; Mahmoud Elmezain & Al-Hamadi, 2012) present feature extraction based on the location, orientation, and velocity of gesture path points. In the aforementioned work, two types of location characteristics are extracted; the first measures the distance between the center of the gesture and each of the points that make it up; the second calculates the distance between the first point of the gesture and each of the points of the gesture. The orientation is calculated by means of the displacement vectors between each of the points of the gesture and the center of the gesture. The velocity is calculated as the Euclidean distance between two consecutive points. These characteristics are calculated in rectangular coordinates and polar coordinates.

A well-known feature extraction method is the Fourier Descriptors (FD)-based method. This method has been used mainly to describe forms; however, it also offers the possibility of describing hand gestures (Premaratne, 2014). Some works present feature extraction methods based on FDs. (Lin & Hwang, 1987) present an alternative representation of the Fourier series using elliptical Fourier features. In his approach, a form can be represented as a set of ellipses which are invariant to rotation and translation. (Lin & Jungthirapanich, 1990) show the development of 3D elliptical Fourier descriptors from 2D elliptical Fourier descriptors.

Other methods for feature extraction are described in the literature. (C. B. Park & Lee, 2011) obtain the temporal information of the gesture along with a probability distribution function (PDF) from the mapping of the hand position. When a person points in one direction, the positions of the hand in three-dimensional space are modeled as a finite mixture of Gaussian distributions. (Yang, Jang, Beh, Han, & Ko, 2012) used a vector of three-dimensional features containing consecutive coordinates using the spherical coordinate system in order to obtain the trajectory of the hand composed of curved lines and strokes. (Bellmore et al., 2011) used a bounding box for the facial area in RGB, which is converted to a luminance image and transformed into a canonical form of 50X60 pixels. The normalization parameters used are $\mu=128$ and $\sigma=100$. (Zhang & Tian, 2013) perform the extraction of features using

the gradient orientation histogram (HOG) method, subsequently applying the dynamic time pyramid method to the extracted features. (Muñoz-Salinas et al., 2008) present principal component analysis applied to the silhouettes of a gesture.

2.4 Recognition

The goal of hand gesture recognition is met when effective feature extraction is coupled with highly efficient sorting. In hand gesture recognition, separating gestures into pre-assigned classes turns the sorting problem into a multi-class sorting problem. For the classification process to be effective, the extraction of characteristics must be carried out under appropriate conditions so that the classes can be differentiated from one another (Premaratne, 2014).

Differentiating one gesture from another is not a serious problem when the gestures differ quite a bit in their execution. However, when the user performs gestures that are very similar, but their meaning is different, there are difficulties in their correct recognition, so that the number of gestures that the system can recognize accurately is very limited. To address this problem, several approaches have been developed to solve the classification problem efficiently. These recognition methods can be classified into two groups: Linear Classifiers and Nonlinear Classifiers (Premaratne, 2014).

Linear classifiers allow you to differentiate two or more classes using lines that are used to establish a separation between the characteristics of each class. The most common linear classifiers are: the Perceptron, linear vector support machines (SVM), the Fisher linear discriminant (LDA) and the Naive Bayes classifier (Premaratne, 2014). Nonlinear classifiers can perform better generalization when the input data is noisy. These classifiers are ideal in the presence of multidimensional data or features. Some nonlinear classification methods are: Multilayer Artificial Neural Networks (ANN), Nonlinear Vector Support Machines (SVM), Hidden Markov Models (HMM) and the K-Nearest Neighbors (KNN) method (Mitra & Acharya, 2007; Premaratne, 2014; Suarez & Murphy, 2012).

One of the most widely used methods in gesture recognition is that of Hidden Markov Models (HMMs). Although this method was initially used in speech recognition (Rabiner, 1989), due to the similarity between speech recognition and gesture recognition, this method has also been used effectively to solve gesture recognition problems. An HMM can be defined as a collection of finite states connected by transitions, in which each state is characterized by having two sets of probabilities: the transition probabilities and the probabilities (continuous or discrete) of emitting an output symbol of a predefined finite alphabet (Premaratne, 2014; Rabiner, 1989). Among the works that present HMMs as a method of recognizing hand gestures are: The work presented by (M Elmezain, Al-Hamadi, Appenrodt, et al., 2008) exposes the use of HMMs for the recognition of Arabic numerals from 0 to 9. The authors used 10 HMMs, one for each of the gestures to be recognized. The extracted characteristics are entered into the 10 models used to evaluate the recognition system. The model that

obtains the highest probability is the one that corresponds to the gesture performed. The developed system has a recognition rate of 98.94%. Likewise, (M Elmezain et al., 2009) presents the use of HMM for the recognition of letters A-Z and numbers 0-9. The procedure is similar to that of the previous description. The developed system has a recognition rate of 98.33%. Other works with similar developments are (Al-Rousan, Assaleh, & Tala'a, 2009; Bansal, Saxena, Desale, & Jadhav, 2011; F.-S. Chen, Fu, & Huang, 2003; M. O. S. M. Elmezain, 2010; Fahn & Chu, 2011; H. K. Lee & Kim, 1999; X. L. X. Liu & Fujimura, 2004; Muñoz-Salinas et al., 2008; Nickel & Stiefelhagen, 2007; C. B. Park & Lee, 2011; Wilson & Bobick, 1999; Yang et al., 2012).

Gesture recognition involves, in some cases, two or more recognition methods to provide greater system reliability. (Nickel & Stiefelhagen, 2007) present the development of a gesture recognition system with application to Human-Robot Interaction (HRI), using Artificial Neural Networks (ANN) and Hidden Markov Models (HMM). ANNs perform head orientation recognition to estimate what the person wants to focus on. HMMs are used to model a hand gesture through three phases: start, hold, and end. The developed recognition system has a recognition rate of over 90%. On the other hand , (C. B. Park & Lee, 2011) present the development of an HRI-oriented gesture recognition system using two HMM in cascade. The first stage has an HMM whose function is to estimate the position of the hand, assigning a more precise position by modeling the kinematic characteristics of the action. The resulting 3D coordinates are used as input in the second stage of HMM, which discriminates "fingerpointing" gestures from other gestures. The system developed has a recognition rate of over 89%.

Another classification method used in the recognition of hand gestures is the *Support Vector Machine* (SVM) method. An SVM is a supervised learning method which is also a linear classifier that maximizes the distance between decision lines. The main advantage of SVMs is that they use *kernels* to transform nonlinear data into separable data clusters within a new feature space (Premaratne, 2014). Among the gesture recognition works that use this classifier we can find (Biswas & Basu, 2011; Muñoz-Salinas et al., 2008; Tang, 2011; Zhang & Tian, 2013). (Biswas & Basu, 2011) employs a multi-class SVM classifier, which is able to correctly classify 8 gestures. (Rahman & Afrin, 2013) propose a system of hand gesture recognition with application to HCI using SVM. The system uses a feature vector obtained by means of the Biortogonal Wavelet Transform. These features enter a multiclass SVM which is able to classify hand gestures into 10 different classes with a recognition rate of 92%.

Additionally, other recognition methods such as those described below are reported in the literature. Mahmoud Elmezain & Al-Hamadi, 2012 presents the *Latent-Dynamic Conditional Random Fields* (LDCRFs) method. LDCRFs are non-directed graphical models that have been developed for the labeling of sequential data, allowing them to be used in dynamic hand gesture recognition. In (Reyes, Domínguez, & Escalera, 2011) a model of the human

body is represented by a vector of characteristics defined by 15 joints corresponding to a 3D human skeleton model using Kinect. Authors use the *Dynamic Time Warping* (DTW) method with the auto-weighting function on each set to achieve real-time recognition of the action. Similarly, (Sempena, Maulidevi, & Aryan, 2011) they also use Kinect to obtain the skeleton model of the human body, and DTW as a recognition method. In this case, the skeletal joints are represented using quaternions to form a feature vector of 60 elements for the 15 joints. In (Kollorz et al., 2008) the development of a static hand gesture recognition system using the K-close neighbor method is presented. In (Fujimura & Liu, 2006; Mo & Neumann, 2006) presents the development of hand gesture recognition systems based on fuzzy rules.

2.5 Hand Gesture Recognition and Hardware Implementation.

Most of the hand gesture recognition systems reported in the literature are mainly developed in software and run on conventional computers, which restricts their use essentially to the field of HCI applications. Conventional computers are not the most suitable platform for intensive high-resolution image processing and high *framerate* (Shi & Tsui, 2007). On the other hand, there are some works in the literature that describe the development of hand gesture recognition systems using reconfigurable hardware such as FPGAs. An image processing algorithm implemented in an FPGA works better than a similar one executed in a conventional computing architecture mainly due to the parallelism offered by the FPGA architecture, which even makes real-time image processing possible (S. H. Lee, SNC, & Siew, 2011).

In addition to the parallelism offered by FPGA devices for real-time image processing applications, these devices are ideal for the development of embedded image processing applications mainly due to their low power consumption, ease of use, low cost, and size compared to conventional computing systems (GPU, CPU, DSP) (Mattoccia, 2013). This makes FPGA devices ideal for the development of portable vision-based gesture recognition systems; autonomous robot navigation; support applications for blind people; applications for the hearing impaired, etc.

Despite the advantages of FPGAs, the process of implementing algorithms, especially those of computer vision, is not an easy task as it can be in a CPU with traditional high-level languages (Mattoccia, 2013). This is because algorithms are developed to be executed sequentially on the computer, so a series of stages must be developed that allow these algorithms to be transformed into their parallel form of execution, to convert them into digital circuits composed of functional blocks, such as memories, flip-flops, registers, etc. (Berkeley Design Technology, 2005). For this reason, the development of vision-based hand gesture recognition applications implemented in FPGAs is a great challenge, especially when there are real-time requirements.

Tables 2 to 6 present a summary of the most representative works on hand gesture recognition. These describe the techniques used by the authors in each of the stages that make up a gesture recognition system, along with their applications and implementation. Within the hardware implementations, it can be seen that most are oriented to the recognition of static gestures using only a digital camera.

Among the works of recognition of static hand gestures implemented in hardware are:(Bonato et al., 2004; Guerrero-Balaguera & Pérez-Holguín, 2015; Jena, Majhi, Gupta, Sehrawat, & Khosla, 2012; S. H. Lee et al., 2011; Shi, Taib, & Lichman, 2006). Bonato et al., 2004 presents the development of a system of recognition of gestures applied to HRI. The system uses a digital camera to capture gestures. In the pre-processing stage, skin color segmentation is applied to extract the shape of the hand. In the feature extraction stage, the image is cropped and compressed to describe the shape of the hand using a vector composed of 768 bits. An ANN is used in the recognition stage. The system works in real-time with a *framerate* of 30fps. In (Guerrero-Balaguera & Perez-Holguin, 2015) a system of gesture recognition applied to the recognition of Colombian Sign Language is presented. The system uses a digital camera to capture gestures. In the pre-processing stage, segmentation is applied based on the threshold of the red component and the morphological operation of opening. The features used consist of a vector that contains the projection of the shape of the hand on the vertical and horizontal axes of the image. An ANN was used in the recognition stage. In (Jena et al., 2012) the development of a hand gesture recognition system with application to HCI is presented. The system employs skin color segmentation to extract the shape of the hand from the background. The characteristics used to describe the gestures are: area of the shape of the hand, perimeter of the shape of the hand, and the direction of the thumb. As a method of recognition, he uses Euclidean distance.

Regarding the recognition of dynamic hand gestures implemented in hardware, we can find the works of (Cho, Li, & Chen, 2012; Shi & Tsui, 2007). In (Cho et al., 2012) the development of a gesture recognition system with application to HCI is presented. This recognition system uses segmentation by skin color to extract the shape of the hand from the background. The characteristics used are based on 16 possible positions to describe the trajectory of a gesture. In the recognition stage, HMMs are used, one for each gesture. The system was implemented on FPGAs with real-time recognition capability with a *framerate* of 30fps. (Shi & Tsui, 2007) present the development of a gesture recognition system with application to HCI. In this work, segmentation by skin color was used to extract the shape of the hand from the background. In the feature extraction stage, the trajectory of the gesture from the center of gravity of the hand was used. An ANN was used as a method of recognition. The system was implemented in FPGAs and its operation is carried out in real time with a *framerate* of 15fps

According to the literature review, it can be observed that a hand gesture recognition system based on artificial vision consists of 4 main stages: capture, pre-processing, and recognition. To effectively extract the gesture from the background, depth information is used in conjunction with skin color segmentation. This makes it easier to determine which gesture is being performed even in scenes with complex backgrounds. In the stages of feature extraction and recognition, a large number of algorithms and methodologies are used to improve the classification of a gesture performed. The most representative characteristics are the Fourier descriptors, Hu invariant moments; the orientation, location and speed of a gesture's trajectory; PCA; and the descriptors of the shape of the hand. As recognition methods, HMM, SVM and ANN predominate.

In the present work we propose the design and hardware implementation of a system with the ability to recognize 6 dynamic hand gestures with application to HRI, using the disparity map, skin color segmentation and the morphological operation of opening in the pre-processing stage. For the feature extraction stage, the orientation of the gesture trajectory is used. In the recognition stage, the use of HMMs is proposed. To obtain the disparity map, a stereoscopic vision system is used, ideal for implementation in hardware due to the computational cost that it requires. The system implementation is done on FPGA SoC with real-time processing capability with a *framerate* greater than 30fps.

Table 2. Summary of the most representative works in gesture recognition.

Paper	Application	Type of Gesture	No of Gestures	Capture Method	Pre-processing	Feature Extraction	Recognition Method	Implementation	Real time	Fps
(Bergh et al., 2011)	HRI	Static	-	Kinect	Depth Segm	Orientation	ANMM	SW	YES	-
(Bergh & Gool, 2011)	HCI	Dynamic	6	Great	Skin Segm Depth Segm	Appearance of the hand	ANMM	SW	YES	-
(Biswas & Basu, 2011)	HCI	Dynamic	8	Kinect	Subtraction of the fund	KING	SVM	SW	N	-
(Breuer et al., 2007)	HCI	Dynamic	-	Great	Noise Reduction Depth Segm	PCA	Hausdorff distance	SW	YES	3
(Elmezain, Al-Hamadi, Appenrodt, et al., 2008)	-	Dynamic	10	Stereo	Skin Segm Depth Segm	Hand tracking Orientation	HMM	SW	YES	15
(M Elmezain et al., 2009)	HCI	Dynamic	36	Stereo	Skin Segm Depth Segm	Hand tracking Orientation Localization Velocity	HMM	SW	YES	15
(Mahmoud Elmezain & Al-Hamadi, 2012)	HCI	Dynamic	36	Stereo	Skin Segm Depth Segm	Hand tracking Orientation Localization Velocity	LDCRFs	SW	YES	15
(Kollorz et al., 2008)	HCI	Static	12	Great	Growth of regions	Hand projection on the x and y axes	KNN	SW	N	-
(Kurakin et al., 2012)	HCI	Dynamic	26	Kinect	Depth Segm	Cell Occupancy Features Silhouette Features	HMM	SW	YES	10
(X. L. X. Liu & Fujimura, 2004)	HRI	Dynamic	10	Great	Depth Segm	Shape, Location, Trajectory, Orientation, Speed	HMM	SW	N	-
(Muñoz-Salinas et al., 2008)	HCI	Dynamic	10	Stereo	Depth Segm	Depth silhouettes features	HMM SVM	SW	N	-

Table 3. Summary of the most representative works in gesture recognition (Continued).

Paper	Application	Type of Gesture	No of Gestures	Capture Method	Pre-processing	Feature Extraction	Recognition Method	Implementation	Real time	fps
(Nickel et al., 2004)	HRI	Dynamic	8	Stereo	Skin Segm Morphology	Vector of hand positions	HMM	SW	NO	-
(Nickel & Stiefelhagen, 2007)	HRI	Dynamic	-	Stereo	Skin Segm Morphology	Vector of hand positions Head Orientation	ANN HMM	SW	NO	-
(Oikonomidis et al., 2011)	HCI	Dynamic	-	Kinect	Skin Segm Depth Segm	Hand Model Parameters	PSO	SW	YES	15
(C. B. Park & Lee, 2011)	HRI	Dynamic	-	Stereo	Connected component labeling	Speed and position	HMM	SW	YES	10
(Rahman & Afrin, 2013)	HCI	Static	10	Mono	Noise Reduction Skin color segmentation Edge detection	Biorthogonal Wavelet Transform	SVM	SW	NO	-
(Ren et al., 2011)	HCI	Static	10	Kinect	Skin Segm Depth Segm	Hand shape	FEMD	SW	NO	-
(Reyes et al., 2011)	HCI	Dynamic	5	Kinect	Skeleton model of the human body	14-joint descriptor	DTW	SW	NO	-
(Sempena et al., 2011)	HCI	Dynamic	7	Kinect	Skeleton model of the human body	Joint Orientation Descriptor	DTW	SW	NO	-
(Suryanarayan et al., 2010)	HCI	Static	6	Great	Otsu Thresholding	PCA	SVM	SW	YES	16
(Yang et al., 2012)	HCI	Dynamic	8	Kinect	CAMSHIFT	Vector of three-dimensional coordinates using the spherical coordinate system	HMM	SW	YES	14

Table 4. Summary of the most representative works in gesture recognition (Continued).

Paper	Application	Type of Gesture	No of Gestures	Capture Method	Pre-processing	Feature Extraction	Recognition Method	Implementation	Real time	fps
(Zhang & Tian, 2013)	HCI	Dynamic	12	Kinect	Enhanced Depth Motion Map	Dynamic Time Pyramid Method	SVM	SW	NO	-
(Al-Rousan et al., 2009)	ASL	Static	30	Mono	background removal,	Cosine dicryte transform (DCT)	HMM	SW	NO	-
(Althoff, Lindl, & Walchshäusl, 2005)	HCI	Dynamic	17	Mono	Adaptive threshold segmentation Morphological operations	Gesture trajectory	HMM	SW	YES	-
(Bansal et al., 2011)	HCI	Dynamic	8	Mono	Skin Segm	adjacency matrix	HMM	SW	NO	-
(Binh, Shuichi, & Ejima, 2005)	ASL	Dynamic	36	Mono	Skin Segm	Kalman filter Descriptors of movement and shape of the hand	P2-DHMMs	SW	YES	25
(Bonato et al., 2004)	HRI	Static	7	Mono	Skin Segm Image compression Image Centering	A 768-bit vector describes the shape of the hand	ANN	HW	YES	30
(F.-S. Chen et al., 2003)	TSL	Dynamic	20	Mono	Background subtraction Otsu threshold Skin Segm Edge detection	Fourier descriptor (FD) Motion analisys	HMM	SW	YES	30
(Z.-H. Chen, Kim, Liang, Zhang, & Yuan, 2014)	HCI	Static	13	Mono	background subtraction	Finger detection	Rule classifier	SW	NO	-
(Cho et al., 2012)	HCI	Dynamic	6	Mono	Median Filter Skin Segm	Features based on 16 gesture positions	HMM	HW	YES	30

Table 5. Summary of the most representative works in gesture recognition (Continued).

Paper	Application	Type of Gesture	No of Gestures	Capture Method	Pre-processing	Feature Extraction	Recognition Method	Implementation	Real time	fps
(Coogan, Awad, Han, & Sutherland, 2006)	HCI	Static/Dynamic	28 / 17	Mono	Skin Segm	color, motion and position	DHMM	SW	YES	10
(Fahn & Chu, 2011)	HRI	Dynamic	8	Mono	Skin Segm Hand tracking	Location, Orientation and Speed of the gesture path	HMM	SW	YES	7
(Ghotkar & Kharate, 2012)	HCI	Static	7	Mono	Hand segmentation Morphological operations	Center of the hand	-	SW	NO	-
(Guerrero-Balaguera & Pérez-Holguín, 2015)	CSL	Static	23	Mono	Hand segmentation using simple treshold Morphological operations	Projection of the shape to the x and y axes	ANN	HW	NO	-
(Jena et al., 2012)	HCI	Static	10	Mono	Skin Color Segmentation	Area Perimeter Thumb direction	Euclidean distance	HW	NO	-
(Keskin, Erkan, & Akarun, 2003)	HCI	Dynamic	8	Stereo	Hand segmentation	Velocity vector sequences	HMM	SW	YES	-
(H. K. Lee & Kim, 1999)	HCI	Dynamic	10	Mono	Hand segmentation	Vector Features with Hand Direction	HMM	SW	NO	5
(S. H. Lee et al., 2011)	HRI	Static	4	Mono	Hand segmentation using the red component	Aspect ratio Position size	Euclidean distance	HW	YES	-
(Malima, Özgür, & Çetin, 2006)	HRI	Static	5	Mono	Skin Color Segmentation	Binary vector counting the number of extended fingers	-	SW	NO	-

Board 6. Summary of the most representative works in gesture recognition (Continued).

Paper	Application	Type of Gesture	No of Gestures	Capture Method	Pre-processing	Feature Extraction	Recognition Method	Implementation	Real time	fps

(Manresa, Varona, Mas, & Perales, 2005)	HCI	Static	8	Mono	Skin Color Segmentation	Ellipse-based features	FSM	SW	YES	30
(Nguyen-Duc-Thanh et al., 2012)	HRI	Dynamic	10	Kinect	Human skeleton detection	Features based on the relative difference between the current position and the start of the gesture	HMM	SW	YES	-
(Shi et al., 2006)	HCI	Static	4	Mono	Skin Color Segmentation	Moment-based features	HMM	HW	YES	15
(Shi & Tsui, 2007)	HCI	Dynamic	8	Mono	Skin Color Segmentation Filtering Contour detection	Calculating the Path of the Hand's Center of Gravity	ANN	HW	YES	15
(Wang, Xia, Cai, Gao, & Cattani, 2012)	HCI	Dynamic	7	Mono	Hand detection using HoG gradient histogram	Hand path orientation	HMM	SW	NO	-
(Xu & Lee, 2012)	HCI	Dynamic	6	Kinect	Camshift algorithm using the disparity map	Hand path orientation	HMM	SW	YES	15
This work	HRI	Dynamic	7	Stereo	Medina Filter Skin Segm Depth Segm Oper Morphologist	Hand tracking Orientation	HMM	HW	YES	>30

3 Fundamental Concepts

3.1 Stereo Vision

Stereo vision is intended to reconstruct the 3D information of a scene from two different images. These images are captured by two cameras that are separated by a previously established distance. Stereo vision works on the same principle as human eyes and allows us to see objects with perception of the depth at which they are (Brown, Burschka, & Hager, 2003; Colodro, Toledo, Martínez, Garrigós, & Ferrández, 2012; Greisen et al., 2011). Figure 3 shows a simplified stereo imaging system.

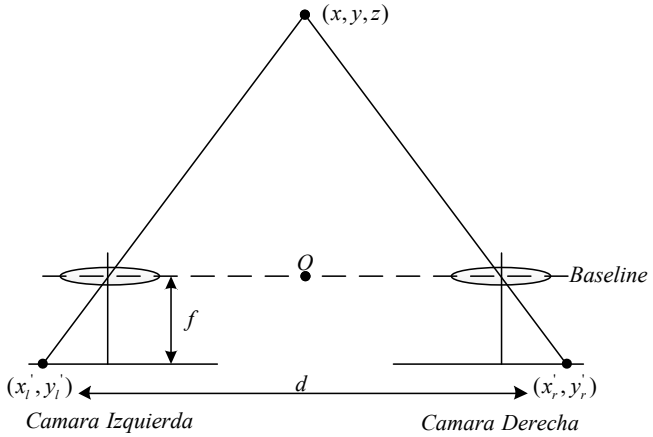


Figure 3. Simplified stereo imaging system. Figure adapted by the author from (Mattoccia, 2013).

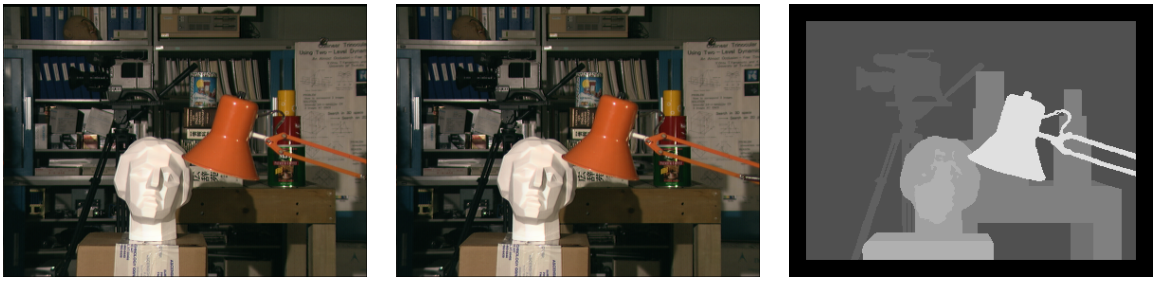
The stereo imaging system in Figure 3 has the following features:

- 2 cameras with their optical axes in parallel and separated by a distance *d*.
- The focal length of both cameras is *f*.
- The line that connects the centers of the camera lenses is called the *baseline*.
- The O origin is in the middle of the baseline.
- The x-axis of the 3D coordinate system is parallel to the baseline.
- Any object in the three-dimensional space that is located in the coordinates (x, y, z) is projected onto image coordinates (x_l', y_l') and (x_r', y_r') in the planes of the left and right images of the respective cameras.

Figure 4 presents an example of stereo imaging, showing a slight shift of the left image (left camera) relative to the right (right camera). The distance between one pixel and its corresponding pixel in the other image (when the two images overlap) is called pixel disparity. To find the correspondence of the left and right images of the stereoscopic image,

a stereo matching algorithm is used (M.O.S.M. Elmezain, 2010). The result is a disparity map that is the same size as the stereo imaging pair with the disparity for each of the pixels. See Figure 4 (c).

Stereo matching algorithms fall into two categories: area-based methods and feature-based methods. For area-based methods, the elements to be matched are fixed-size image windows, and the similarity criterion is a measured amount of the correlation between the windows in the two images. For feature-based methods, matching is aimed at a set of scattered features rather than windows. The first category allows for the calculation of denser depth maps, while the second requires a priori information to determine the optimal features to be used (Colodro et al., 2012; M. O. S. M. Elmezain, 2010).



(a) (b) (c)

Figure 4. Stereo image "Tsukuba". (a) Left image. (b) Right image. (c) Ground *Truth*. Source (Scharstein & Szeliski, 2002).

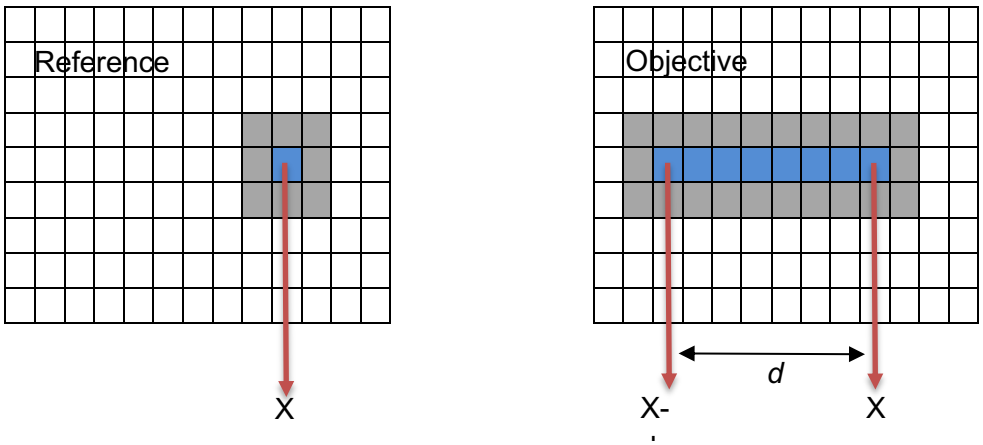


Figure 5. Area-based stereo matching method. In the reference image the window is centered at the x-point, while in the target image there are multiple windows centered at $[x,x-d_{max}]$. Figure adapted by the author from (Mattoccia, 2013).

Figure 5 shows graphically the area-based stereo matching method. In this method, a window is taken in the reference image and compared to *d* windows in the target image. To do this, a cost function is used to determine the similarity between the reference window and each of the target windows. The disparity is calculated by taking into account the target window with the highest degree of similarity (Mattoccia, 2013).

Area-based matching methods determine the best disparity for a pixel and its vicinity by applying a strategy called WTA (*winner-takes-all*). This strategy selects the candidate window with the best similarity value (Humenberger, Zinner, Weber, Kubinger, & Vincze, 2010; Mattoccia, 2013).

3.1.1 Census Transform

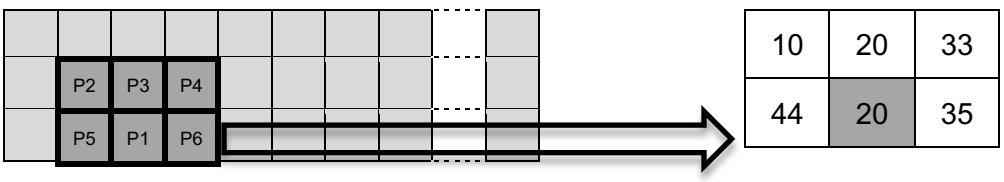
The Census transform is an area-based stereo matching algorithm that is robust to changes in lighting. This algorithm transforms the images before calculating the correspondence for each disparity level. The transform is based on a comparison function ξ, which is used to compare the value of two pixels P_1 and P_2. Equation (1) presents the comparison function ξ (Ambrosch, Zinner, & Kubinger, 2009; Colodro et al., 2012; Zabih & Woodfill, 1994).

$$\xi(P_1, P_2) = \begin{cases} 0, P_1 \leq P_2 \\ 1, P_1 > P_2 \end{cases} \quad (1)$$

The Census Transform considers a window $M\ X\ N$ centered on a pixel $\mathrm{I}(u,v)$ in the image and encodes the window information into a string of $MN-1$ bits in which it assigns value '1' to the bit associated with each pixel with a value greater than the central one and value '0' in another case (Ambrosch, Kubinger, Humenberger, & Steininger, 2009; Ambrosch, Zinner, et al., 2009; Colodro et al., 2012; Zabih & Woodfill, 1994). Equation (2) shows a mathematical representation of this transform, where it $\otimes$ represents the concatenation operator and $m = \frac{M-1}{2}$ and . $n = \frac{N-1}{2}$

$$I_{Census}(u,v) = \bigotimes_{i=-m}^{m} \bigotimes_{j=-n}^{n} \xi(I(u,v), I(\mathrm{u+i}, \mathrm{v+j})) \quad (2)$$

Figure 6 shows an example for calculating the Census transform for a pixel of an image whose vicinity was defined by a 3X3 pixel window. Taking into account the arbitrary values selected, the comparison of the central pixel, whose value is 20, with each of the pixels defined in its vicinity is made. This process results in a string of bits as shown in the bottom right of Figure 6. This procedure must be repeated throughout the image, always maintaining the same selected neighborhood size. The final result of the Census transform will be an array the size of the original image, where each of the positions of the array will correspond to a string of bits.



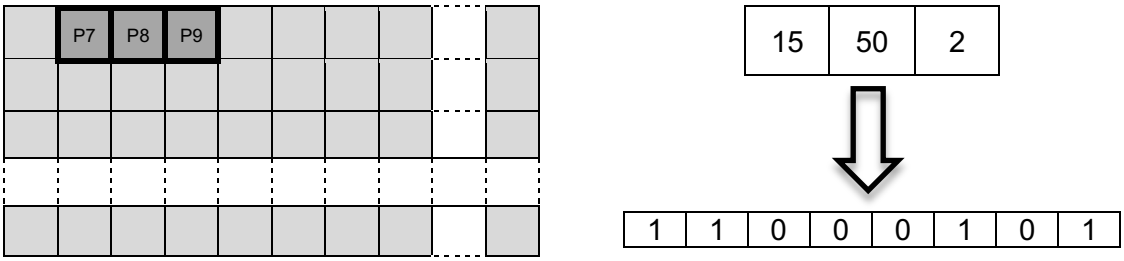


Figure 6. Calculation of the Census transform for a pixel with a vicinity of 3X3 pixels. Source: Author.

3.1.2 Transform Census Scattered

The main disadvantage of the Census transform is the large size of the resulting vector. This vector corresponds to the number of pixel comparisons that are made, and has a great impact on the hardware resources required for its implementation, especially in stereo vision systems. For this reason, different methods have been proposed to mitigate this drawback (Fife, 2011). One of them is to apply a mask with a uniform sub-muetreum pattern, which determines the pixels within the window that are taken into account to apply the transformation. This method reduces the number of comparisons needed without deteriorating the result of the Census transform (Fife, 2011; Humenberger et al., 2010).

Figure 7 shows two windows, each with a different pattern. The gray areas represent the pixels that are compared to the central pixel. In the window on the left (a) you can see that all pixels are taken into account. In this case, the resulting string of bits will have a total of 24 bits. In the window on the right (b) only 50% of the pixels in the neighborhood are taken into account, so the bit string of the Census transform will have only 12 bits. This reduction in the number of bits allows optimizing the use of hardware resources (memory) needed to calculate the disparity in stereo images, without affecting the final result (Fife, 2011).

Figure 7. Masks for calculating the Census transform. (a). Conventional 5X5 mask. (b) Evenly distributed 5X5 mask, Census transform neighborhood with 50% dispersion. Source: Author.

3.1.3 Sum of Hamming Distances (SHD)

Census transformation is only the first step in the Census matching method for stereo imaging. The next step is to add the Hamming distances (SHD) of the Census transformation vectors of each of the candidate windows. The calculation of the Hamming distance is done by comparing two strings of bits, and the result is the number of bits by which these strings differ (Ambrosch, Kubinger, et al., 2009; Ambrosch, Zinner, et al., 2009; Fife, 2011;

Humenberger et al., 2010; Zabih & Woodfill, 1994). The similarity SHD can be measured as shown in equation (3), where $I_{LCensus}$ y $I_{RCensus}$ are the Left and Right images in their Census transformation, and *d* corresponds to the disparity. See Figure 5.

$$SHD: \sum_{i=-m}^{m} \sum_{j=-n}^{n} \text{Hamming}(\text{I}_{LCensus}(\text{u}+\text{i},\text{v}+\text{j}),\text{I}_{RCensus}(\text{u}+\text{i},\text{v}+\text{j}+\text{d})) \quad \forall \text{d} \in \{0,1,2,....,\text{d}_{\max}\} \qquad (3)$$

3.2 Digital Image Processing

Digital image processing is a set of mathematical operations applied to digital images. The goal of image processing is to transform one image into another with different characteristics, such as noise removal, contour enhancement, object or pattern detection, etc. (Gonzalez & Woods, 2002).

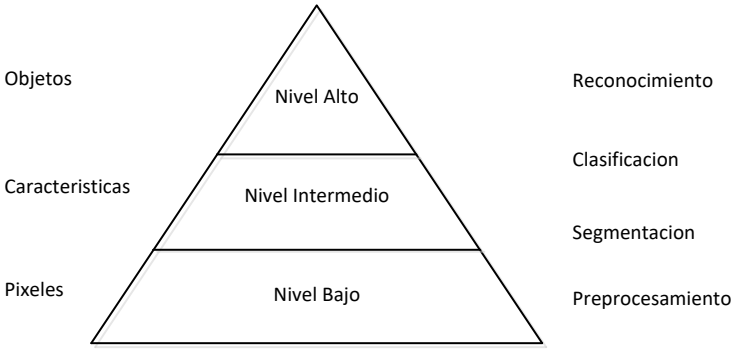


Figure 8. Image processing pyramid. Figure adapted by the author from (Bailey, 2011).

Image processing operations are grouped into categories according to the type of data they are working with. This grouping is commonly referred to as the image processing pyramid, see Figure 8. At the lowest level of the pyramid are pre-processing operations. These are low-level operations oriented to pixels or pixel neighborhoods, whose objective is to highlight the important information within the image and eliminate that information that does not represent usefulness. Examples of pre-processing operations are: edge detection, filtering for noise reduction, and brightness and contrast enhancement. At the intermediate level are segmentation and classification operations. Targeting operations transform an image into regions. Examples of segmentation operations are: Thresholding, color detection, region growth, and labeling of related components. Classification operations transform information from regions to features, and are used to identify or classify objects within various categories. At the highest level are recognition operations, which allow a description or interpretation of the scene to be made (Bailey, 2011).

There are two categories of low-level image processing operations: pixel-level operations and neighborhood-oriented operations. The first category is the simplest type of operations since the output pixel is obtained from a function f applied to the corresponding input pixel

(Bailey, 2011). These types of operations are defined as shown in equation (4), where $I[x,y]$ it represents a pixel located in the coordinates (x,y) of the image I and $Q[\mathrm{x},\mathrm{y}]$ represents the value of a pixel obtained by the function f applied to the input pixel $I[x,y]$. The most common pixel-oriented operations are: brightness and contrast adjustment, image negative, global thresholding, multilevel thresholding, color segmentation, and transformations between color spaces.

$$Q[x,y] = f(\mathrm{I}[\mathrm{x},\mathrm{y}]) \tag{4}$$

Neighborhood-oriented operations are an extension of pixel-oriented operations, with the difference that the output value depends on a function applied to a local neighborhood (or window) (Bailey, 2011). These operations can be defined mathematically by equation (5), where W is a local neighborhood centered on $I[x,y]$, as shown in Figure 9. Examples of neighbourhood-oriented operations are: Sobel filter, Prewit filter, Median filter, Media filter, erosion, expansion, etc.

$$Q[x,y] = f(I[x,y],...,I[x+\Delta x, y+\Delta y]), \qquad (\Delta x, \Delta y) \in W \tag{5}$$

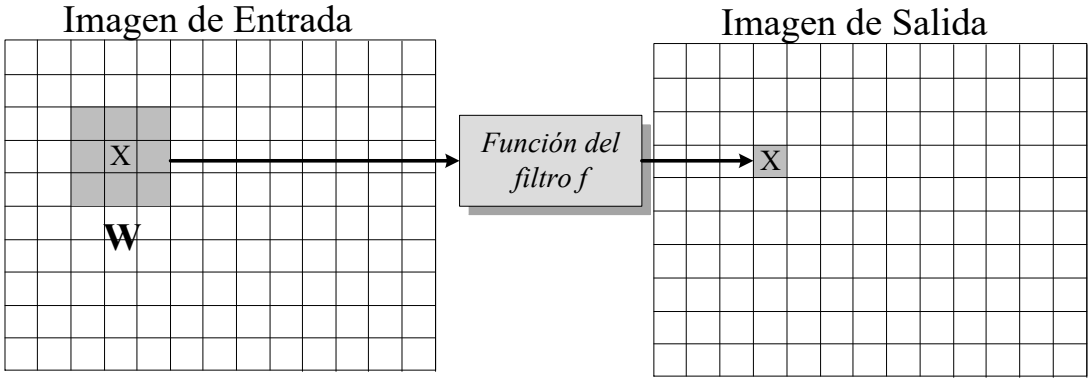


Figure 9. Local Filter. The shaded area represents a neighborhood of pixels entering a filter, which generates a new pixel in the center of that neighborhood. Figure adapted by the author from (Bailey, 2011).

3.3 Embedded Image Processing

An embedded system is a computer-based system that is embedded or embedded within a product or component. An embedded system is usually designed to accomplish a specific task, or a small number of specific tasks, usually with real-time constraints. A basic example of an embedded image processing system is a digital camera. These systems allow digital images to be captured, stored, retouched, visualized, compressed, and decompressed (Bailey, 2011).

Embedded vision systems are extremely useful in "smart" camera applications, as they are not only used to take images, but also analyze and extract important information for specific applications, such as surveillance, inspection and quality control in industry, vision in robots, etc.

3.3.1 Real-time image processing

A real-time system is one in which the response to an event must occur within a specific time, otherwise the system is considered to have failed. A real-time image processing system is one that regularly captures images, analyzes those images for information, and uses the resulting information to monitor some activity. All image processing must occur within a set time, usually (but not always) this time is set by the frame rate (Bailey, 2011).

3.3.2 Parallel image processing

Traditionally, image processing platforms are based on sequential computer architectures. The execution of tasks is carried out as a sequence of instructions ordered through a Logical Arithmetic Unit (ALU). However, an algorithm, especially in image processing, can be implemented by separate processing elements resulting in a completely parallel execution. This is especially useful for low- and intermediate-level algorithms in the image processing pyramid (Bailey, 2011).

All image processing algorithms are made up of a sequence of image processing operations. This translates as a form of *temporal parallelism*. This structure suggests using a separate processor for each operation, as shown in Figure 10. This is known as a pipeline architecture. The operation of this architecture is very similar to a production line, in which data passes through each of the stages as it is processed. Each processor applies its operation and the result is sent to the next stage.
In image processing, it is important to consider the time it takes for each of the processing stages to receive data, process it and generate its respective response. This time is known as latency. Latency can be small if the operation only depends on an input pixel, or a small vicinity of pixels. When an operation requires having the entire image complete to obtain an output, in this case the latency will be very high. Pipeline operations can provide a significant increase in performance when all operations have low latency, since the processor of the next operation can start working even though the previous processor has not yet finished doing so (Bailey, 2011).

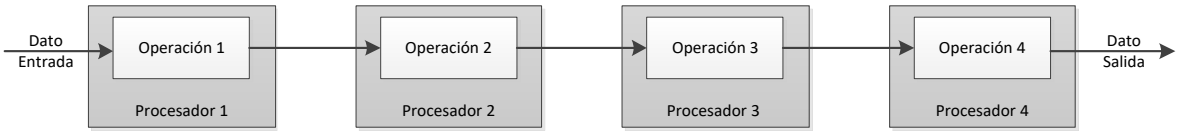


Figure 10. Temporal parallelism using pipeline processors. Adapted from (Bailey, 2011).

Another type of parallelism that can be used in image processing is spatial parallelism. This type of parallelism involves partitioning the image into blocks and using a separate

processor to implement the operation on each partition. The most common partitioning schemes are to divide the image into row blocks, column blocks, or rectangular blocks as shown in Figure 11. An important consideration when partitioning the image is to minimize communication between processors, which involves minimizing communication between different partitions. Therefore, each processor must have local memory to reduce any delays caused by global memory accesses (Bailey, 2011).

Logical parallelism reuses a functional block many times within an operation. This type of parallelism commonly corresponds to loops within an algorithm, for which the same operation is always repeated. In this case, logical parallelism is implemented by splitting the loop present in the algorithm by executing the operation in parallel (Bailey, 2011).

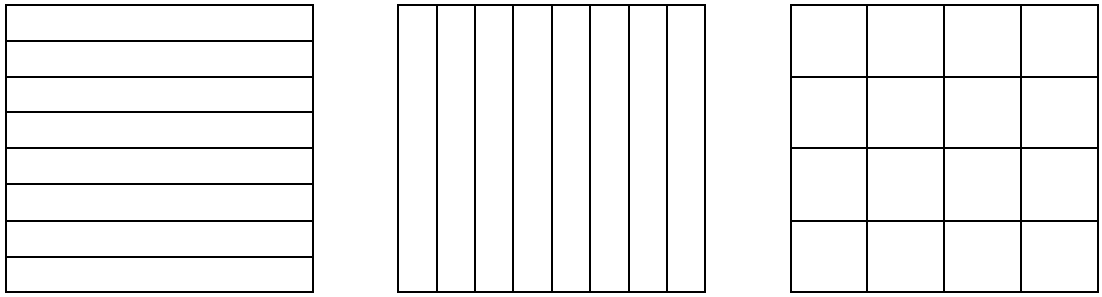


Figure 11. Spatial parallelism based on image partitioning. (a) Partitioning by rows. (b) Partition by Columns. (c) Partitioning by blocks. Figure adapted by the author from (Bailey, 2011).

One of the most common bottlenecks in image processing is the time and bandwidth required to read the image from memory and write the result back to memory. *Stream processing* allows us to address this problem by converting spatial parallelism into temporal parallelism. The image is read and written at a sweep equivalent to one cup of one pixel per clock cycle. This type of processing is most effective if pixel neighborhoods are used as inputs, otherwise cache requirements may be excessive. For most operations, latency is low compared to loading and processing an entire image. In this case, the image processing time is usually governed by the *frame* rate. Figure 12 shows graphically how this type of processing works (Bailey, 2011).

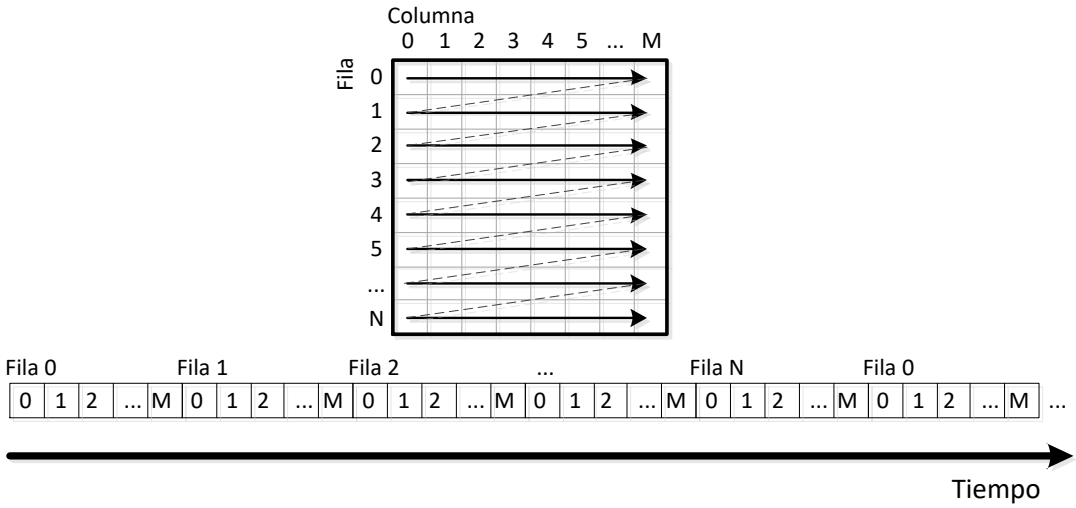


Figure 12. Data flow processing. Conversion of spatial parallelism into temporal parallelism. Figure adapted by the author from (Bailey, 2011).

3.3.3 Algorithm Mapping Techniques in FPGAs

Unlike software implementation, FPGA implementation of image processing algorithms, using *stream processing,* leads to an increase in the performance of these algorithms. However, more effort is required in hardware design, which involves developing or applying algorithm mapping techniques to FPGA resources in order to obtain the desired results. According to (Gribbon, Bailey, & Johnston, 2005, 2006) there are three types of constraints in the mapping process: time constraint, memory bandwidth constraint, and available resource constraint. The use of resources depends on the efficiency of the design. Care must be taken at the level of logical design. The following sections describe the standard techniques used to address the first two constraints.

3.3.3.1 Time restriction

Time restriction is common in real-time applications, where data rate requirements impose strict limitation. At video speeds, for example, all the processing needed for each pixel must be done at the speed of the pixel clock (or faster). To comply with this restriction, low-level *pipelining* is generally required (Gribbon et al., 2005).

Low-level segmentation or *pipelining* divides an operation into small stages so that the entire operation is executed in segments. As a result, the propagation delay is reduced to a single stage. Because only a portion of the operation is executed on a clock cycle, the speed can be increased. To execute a complete operation requires more than one clock cycle. If the hardware is duplicated, multiple pixels can be processed at once. The number of pixels that can be processed simultaneously depends on the total pipeline stages (Bailey, 2011).

To illustrate this procedure, consider the following equation.

$$y = ax^2 + bx + c$$
$$y = (ax + b)x + c \tag{6}$$

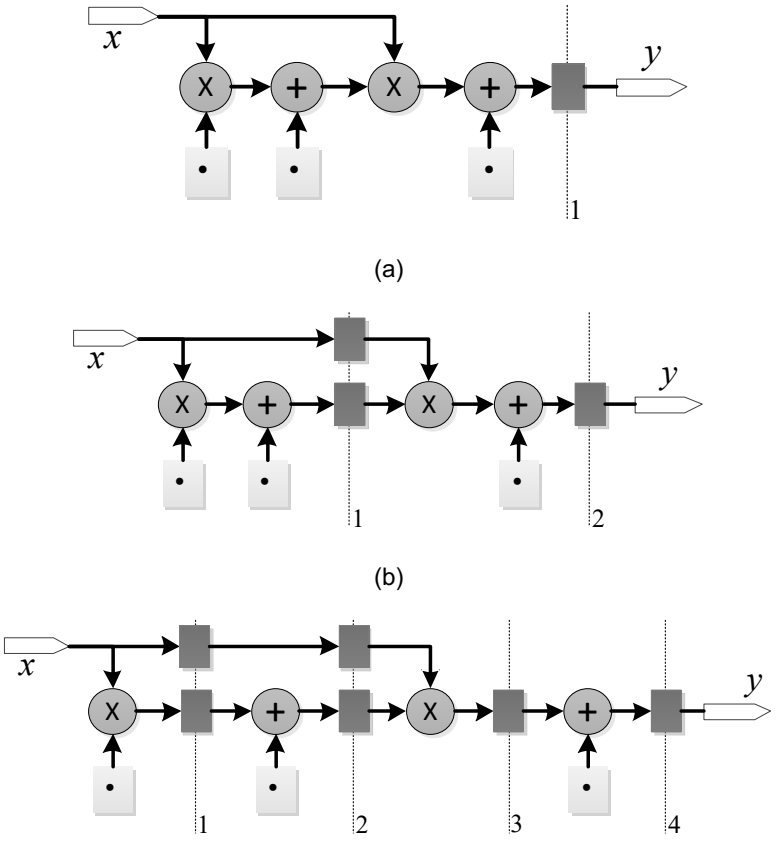


Figure 13. Pipeline execution example. (a) Perform the calculation in a single clock cycle. (b) Perform the calculation in two clock cycles (two pipeline stages). (c) Perform the calculation in four clock cycles (four Pipeline stages). Figure adapted by the author from (Bailey, 2011).

The above equation can be implemented using one, two, or four pipeline states as shown in Figure 13. Optimal latency can be obtained by implementing 2 pipeline states as the *throughput* of the system is improved. This is because the addition and multiplication operations are evenly distributed between the two stages. In the case of four pipeline stages, stages 1 and 3 have multiplication operations, while stages 2 and 4 have addition operations. A multiplication operation requires more time than an addition operation. In other words, the distribution of time in the stages is not uniform. This causes imbalance, and increases system latency. This effect can be attenuated by a method known as *retiming*. This method consists of executing a part of the multiplication in the first stage and the remaining part in the next stage, which allows obtaining an equivalent time distribution between the stages of the *Pipeline*. Therefore, the selection of the number of pipeline stages is extremely important in designs with FPGAs (Bailey, 2011).

3.3.3.2 Memory bandwidth restriction

Memory bandwidth constraint occurs when memory is required to be accessed too frequently, creating a processing bottleneck. This is the case with the standard software approach in which each pixel required by an operation is read from memory, the operation is applied, and the results are written back to memory. This process is repeated for each pixel within the image. A similar process is repeated for each operation that is required to be applied to the image. Continuous reading and writing to memory in many situations can be avoided by using a frequently reused data caching architecture (Bailey, 2011; Gribbon et al., 2005).

One method in which chache storage is implemented in image processing operations is *row buffering*. If you consider a spatial operation with a 3X3 window, where each output pixel is a function of 8 neighboring pixels in the window. In a normal implementation, nine pixels must be read at each position of the window (for each clock cycle for *Stream* Processing) and in addition, each pixel must be read nine times while the window is scrolled through the image. Typically, pixels that are horizontally adjacent are required in consecutive clock cycles, so they can be buffered and delayed by registers (Bailey, 2011; Gribbon et al., 2005). Figure 13 graphically shows the operation of two *Row Buffers*. As you can see, every time you enter a new pixel in the window's scan direction, the pixel values of the previous rows are available to be processed, thus avoiding the repetitive memory reading of those pixels. The FPGA implementation of this caching method is shown in Figure 15. In this figure you can see the row storage buffers, which are typically FIFO type memories with a storage capacity of pixels $M = N - w$, where N is the width of the image in pixels and w the width of the window. The window is made up $W = w^2$ of horizontally interconnected registers as can be seen in the figure. These logs can be read simultaneously in a way that allows the processing of the operation linked to that window.

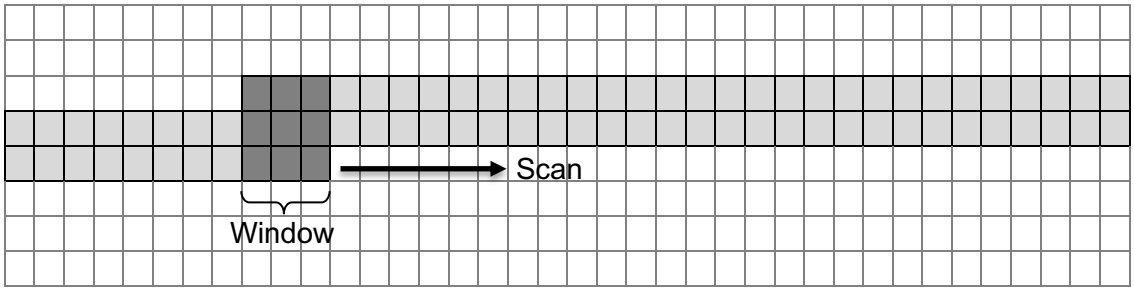


Figure 14. Scanning of the image using a 3X3 window using the *Row Buffering* method. Figure adapted by the author from (Bailey, 2011).

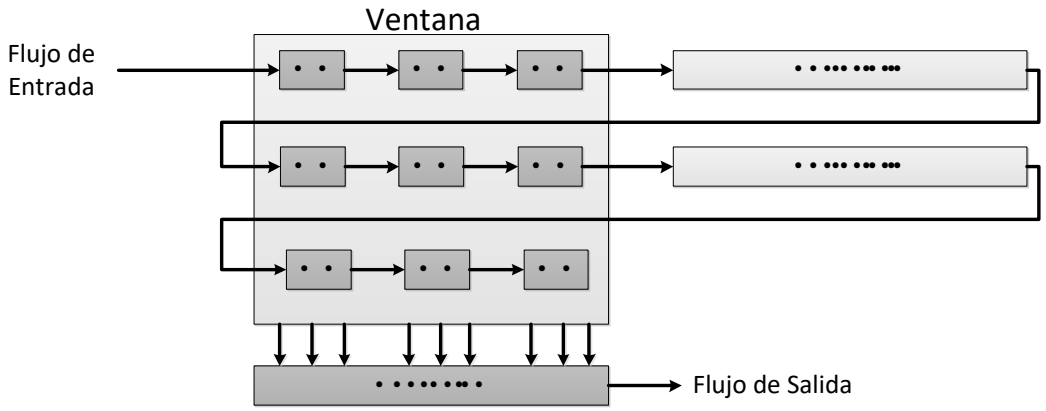


Figure 15. *Row Buffering* Architecture for FPGAs. Adaptation from (Bailey, 2011).

There are other methods that allow you to avoid memory bandwidth restriction, some of them are: double buffer storage, *Frame Buffer*, cache memory, among others.

3.4 Hidden Markov Models

A Hidden Markov Model (HMM) is a statistical model that describes a stochastic sequence $O = O_1, O_2, ..., O_T$, as an indirect observation of a hidden random sequence $Q = Q_1, Q_2, ..., Q_T$, (Rabiner, 1989).

In a normal Markov model, the state is directly visible to the observer, so the probabilities of transition between states are the only parameters. In a hidden Markov model, the state is not directly visible, but only the variables influenced by the state are. Each state has a probability distribution over the possible exit symbols. Consequently, the sequence of symbols generated by an HMM provides some information about the sequence of states.

A discrete first-order HMM is formally defined by the following elements:

- A set $S = \{S_1, S_2, ..., S_k\}$ of hidden states. Although these states are hidden, there are some practical applications, where states can have physical meaning. For example, in handwriting recognition or speech recognition. A state in time t is denoted as Q_t (Rabiner, 1989).

- A matrix of transitions $A = \{a_{ij}\}$, of dimensions k x k, where the element $a_{ij} \geq 0$ represents the probability of going from state S_i to state S_j. In addition, it must be complied with that $\sum_{j=1}^{k} a_{ij} = 1$. This matrix is known as the stochastic matrix (Rabiner, 1989).

- A set $V = \{v_1, v_2, ..., v_m\}$ of observation symbols. This is the alphabet, and it corresponds to the physical outputs of the process being modeled (Rabiner, 1989).

- A dimension emission matrix $B = \{b(j \mid S_i)\}$ (k x m), which indicates the probability of emitting a symbol v_j in the state S_i. Donde $b(j \mid S_i) = P[O_t = v_j \mid O_t = S_i]$, con $b(j \mid S_i) \geq 0$ y $\sum_{j=1}^{m} b(j \mid S_i) = 1$ (Rabiner, 1989).

- An initial probability distribution $\pi = \{\pi_i\}$, where: $\pi_i = \pi(S_i) = P[q_1 = S_i]$ with $\pi_i \geq 0$ and $\sum_{i=1}^{k} \pi_i = 1$.

A Hidden Markov model (HMM) is usually denoted as a tuple as follows: $\lambda = (\mathrm{S}, \mathrm{V}, \mathrm{A}, \mathrm{B}, \pi)$ (Rabiner, 1989).

3.4.1 Basic problems of an HMM

There are three basic problems that must be solved for the model to be useful in real-world applications:

Problem 1: Given the sequence of observations $O = O_1, O_2, ..., O_T$ and a model $\lambda = (\mathrm{A}, \mathrm{B}, \pi)$, how do you efficiently calculate the probability $P(\mathrm{O} \mid \lambda)$ of observing the sequence given the model? The evaluation problem is of particular interest when one wants to choose between several possible models, since one can choose the one that best explains a sequence of observations (Rabiner, 1989).

The solution to the model evaluation problem is an algorithm known as the forward and backward procedure. This algorithm is based on the use of two intermediate variables. The forward variable α and the reverse variable β . The advance variable is defined in equation (7) and represents the probability of observing the sequence $O = O_1, O_2, ..., O_t$ up to time t starting in the state S_i (Rabiner, 1989). Equations (8), (9) and (10) present the procedure to be followed to calculate this variable.

$$\alpha(i) = P(\mathrm{O}_1, \mathrm{O}_2, ..., \mathrm{O}_t, \mathrm{q}_t = \mathrm{S}_i \mid \lambda) \tag{7}$$

1. Initialization

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \le \mathrm{i} \le \mathrm{k} \tag{8}$$

2. Induction

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{k} \alpha_t(i) a_{ij} \right] b_j(O_{t+1}), \qquad 1 \le \mathrm{t} \le \mathrm{T\text{-}1} \quad 1 \le j \le k \tag{9}$$

3. Termination

$$P(O \mid \lambda) = \sum_{i=1}^{k} \alpha_T(i) \tag{10}$$

The receding variable is defined as $\beta_t(\mathrm{i}) = \mathrm{P}(\mathrm{O}_{t+1}, \ldots \mathrm{O}_T \mid \mathrm{Q}_t = \mathrm{S}_i, \lambda)$ and represents the probability of observing the symbols $\mathrm{O}_{t+1}, \ldots \mathrm{O}_T$, starting at the state S_i at time t (Rabiner, 1989). This variable is calculated recursively as shown in equations (11) and (12).

$$\beta_T(i) = 1 \qquad 1 \le \mathrm{i} \le \mathrm{k} \tag{11}$$

$$\beta_t(i) = \sum_{j=1}^{k} a_{ij} b(O_{t+1} \mid S_j) \beta_{t+1}(j) \qquad t = T-1, \ldots, 1 \qquad 1 \le \mathrm{i} \le \mathrm{k} \tag{12}$$

Problem 2: Given the sequence of observations $O = O_1, O_2, \ldots, O_T$ and a model $\lambda = (\mathrm{A}, \mathrm{B}, \pi)$, how do you choose a corresponding sequence of states $\hat{Q} = \hat{Q}_1, \hat{Q}_2, \ldots, \hat{Q}_T$, so that it best explains the observations? In this case the goal is to find the sequence of states $\hat{Q} = \hat{Q}_1, \hat{Q}_2, \ldots, \hat{Q}_T$ in such a way that $\hat{Q} = \arg\max_{Q} P(\mathrm{O}, \mathrm{Q} \mid \lambda)$. This problem is solved by Viterbi's algorithm (Rabiner, 1989). This procedure begins by defining the following variable.

$$\delta_t(i) = \max_{Q_1, \ldots, Q_{t-1}} P(\mathrm{Q}_1, \mathrm{Q}_2, \ldots, \mathrm{Q}_t = \mathrm{S}_i, \mathrm{O}_1, \mathrm{O}_2, \ldots, \mathrm{O}_t \mid \lambda) \tag{13}$$

This variable represents the best score or highest probability along a simple trajectory, at time *t*. To obtain the sequence of states of the argument of δ_i must be stored for each *t* and each *i* obtaining as a result a vector $\psi_t(i)$. Viterbi's algorithm is defined by the following recursive steps:

1. Initialization

$$\delta_1(i) = \pi_i b(O_1 \mid \mathrm{S}_i) \qquad 1 \le \mathrm{i} \le \mathrm{k}$$
$$\psi_1(i) = \phi \qquad 1 \le \mathrm{i} \le \mathrm{k}$$

2. Recursion

$$\delta_t(i) = \max_{1 \le i \le k} \left[\delta_{t-1}(i) a_{ij} \right] b(\mathrm{O}_t \mid \mathrm{S}_j) \qquad 1 \le \mathrm{j} \le \mathrm{k} \qquad 2 \le \mathrm{t} \le \mathrm{T}$$

$$\psi_t(i) = \arg\max_{1 \le i \le k} \left[\delta_{t-1}(i) a_{ij} \right] \qquad 1 \le \mathrm{j} \le \mathrm{k} \qquad 2 \le \mathrm{t} \le \mathrm{T}$$

3. Termination

$$\hat{P} = \max_{1 \le i \le k} \left[\delta_T(i) \right]$$

$$\hat{Q}_T = \arg\max_{1 \le i \le k} \left[\delta_T(i) \right]$$

4. Backtracking of the Sequence of States

$$\hat{Q}_t = \psi_{t+1}(\hat{Q}_{t+1}) \qquad \mathrm{t=T\text{-}1,T\text{-}2,...,1}$$

Problem 3: How to adjust the parameters of the model $\lambda = (\mathrm{A}, \mathrm{B}, \pi)$ to maximize the $P(\mathrm{O} \mid \lambda)$? This is the most difficult problem to solve and is usually solved by employing the criterion of maximum probability ML. The algorithm that allows this criterion to be implemented is the algorithm known as *Baum-Welch* (Rabiner, 1989). In order to describe the procedure for re-estimating the parameters of the HMM model, two variables are used.

- $\xi_t(i, j)$: represents the probability of moving from the state S_i at time *t* to S_j the state at time *t+1*, given a sequence of observations and the model. This variable is calculated using the forward and backward variables.

$$\xi_t(i, j) = P(\mathrm{Q}_t = \mathrm{S}_i, \mathrm{Q}_{t+1} = \mathrm{S}_j \mid \mathrm{O}, \lambda)$$

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} b(\mathrm{O}_{t+1} \mid \mathrm{S}_j) \beta_{t+1}(\mathrm{j})}{\sum_i \sum_j \alpha_t(i) a_{ij} b(\mathrm{O}_{t+1} \mid \mathrm{S}_j) \beta_{t+1}(\mathrm{j})} \qquad (14)$$

- $\gamma_t(i)$: Represents the probability of starting in the state S_i at time *t*, given a sequence of observations and a model. The variable is calculated as shown below.

$$\gamma_t(i) = P(Q_t = S_i \mid O, \lambda)$$
$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_i \alpha_t(i)\beta_t(i)} \tag{15}$$
$$\gamma_t(i) = \sum_{j=1}^{k} \xi_t(i,j)$$

Once the above variables have been calculated, the parameters of model A, B and the following π are estimated:

$$\bar{\pi}_i = \gamma_t(i) \tag{16}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{17}$$

$$\bar{b}(\mathrm{v}_j \mid \mathrm{S}_i) = \frac{\sum_{t=1}^{T-1} \gamma_t(i)}{\sum_{t=1}^{T-1} \gamma_t(i)} \tag{18}$$

4 Description of the Gesture Recognition System

This paper presents the design and implementation of a vision-based gesture recognition system in FPGA. The system is designed to capture, process and recognize 7 dynamic hand gestures in real time. The selected gestures correspond to the basic control commands of a mobile robot taken as an example to demonstrate the application possibilities of the gesture recognition system designed. The user must perform the gestures using only the right hand without the need to use any additional accessories. The gesture is interpreted according to the trajectory obtained as a result of the movement of the hand. Figure 16 presents the dynamic gestures used in this work and their meaning for the control of the mobile robot.

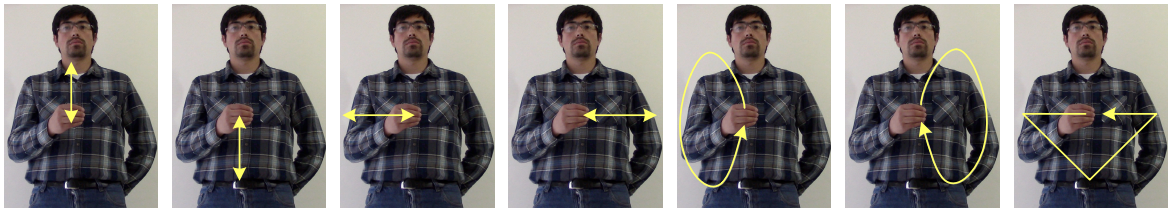


Figure 16. Gestures used in this work to remotely control a robot. Source: author.

4.1 Stages of the Gesture Recognition System

The hand gesture recognition system developed in this work is composed of five main stages: gesture capture, pre-processing, feature extraction, recognition and communication, and robot control (see Figure 17). Gestures are captured in a uniformly lit and backdrop environment using a stereoscopic vision system consisting of two ArduCam cameras featuring Omnivision's OV5642 sensor.

In the Pre-processing stage, the median filter is applied to the captured images in order to eliminate the noise produced in the acquisition. Once the filtering process has been carried out, the disparity map is calculated using the stereo correspondence algorithms: Scattered Census Transform and Hamming Sum of Distances. To detect hands, skin color segmentation is used in conjunction with disparity map segmentation. Additionally, the morphological opening operation is used to eliminate the noise produced in the segmentation.

In the Feature Extraction stage, the gesture is obtained by calculating the centroid of the hand shape from a consecutive sequence of images. The gesture may contain noise, which is why a moving average filter is used to soften the gesture performed. The resulting

trajectory of the gesture is a spatio-temporal pattern composed of points of the centroid of the hand. One of the challenges in gesture recognition is to determine those movements that have a meaning (gesture) of other types of movements (non-gesture); this process is known as "G*esture Spotting*" (M. O. S. M. Elmezain, 2010). The process of "G*esture Spotting*" is carried out by detecting the movement of the hand, taking into account static starting and ending positions. Once this process is done, the characteristics that describe the executed gesture are extracted. In this work, orientation between two consecutive points of the hand's trajectory is used to describe any gesture. Subsequently, these characteristics are transformed through a quantization process in which 8 possible discrete values are calculated that are necessary for the use of HMM-based classifiers.

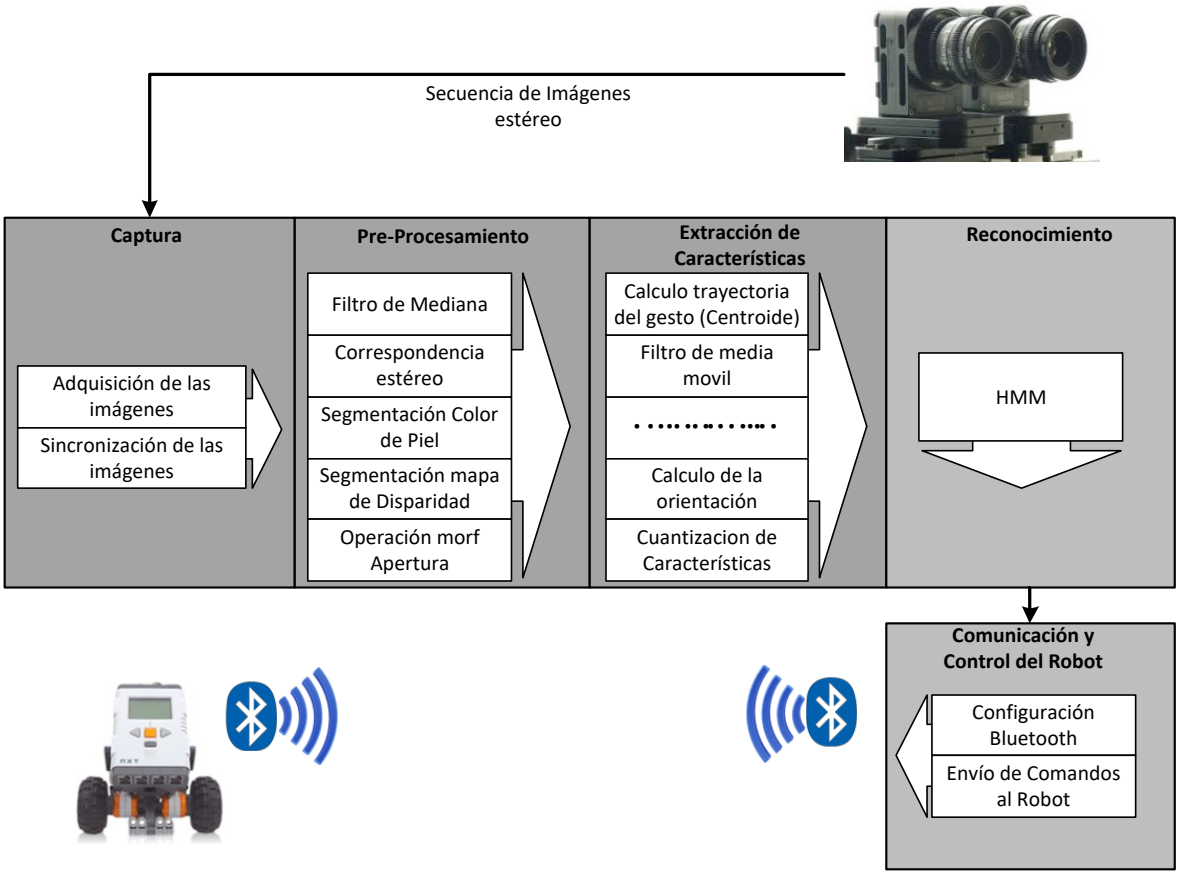


Figure 17. Conceptual schematic of the hand gesture recognition system. Source: author.

In the Reconnaissance stage, HMMs are used as classifiers. HMMs are ideal probabilistic models for the modeling of spatio-temporal patterns, as is the case of the gestures used in this project. In this work, 7 HMMs with LRB architecture were used, one for each gesture. The training of the HMMs was carried out using the cross-validation methodology called "*K-Fold Cross-Validation*". This type of validation allows you to select a model with the best performance from several optional models, each with different parameters.

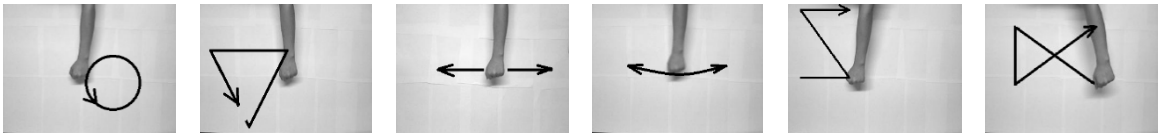
In the communication and control stage of the robot, a LEGO Mindstorms NXT mobile robot was used. The recognition system detects the gesture made and sends the corresponding command to the robot via Bluetooth. The robot receives the prompt and executes the programmed action according to the action performed by the user.

4.2 Design Process of the Gesture Recognition System.

The hand gesture recognition system presented in this work was developed by executing three fundamental steps: *i*) algorithm development, in this stage the sequence of image processing operations required to transform the input image or images into the desired result is determined. In addition, the additional algorithms necessary to process the information extracted from the images are developed. *ii*) determine which algorithms or portions of them are implemented through a custom hardware architecture (FPGA) or, failing that, in a conventional processor (software). This selection is made taking into account the portion of the algorithm that can be parallelized and the computational capacity required by it. For the selection of the hardware architecture, the algorithm must be transformed into operations, which have an equivalent of physical processing. *iii*) implementation of the algorithms on the chosen architecture in accordance with the previously established selection parameters.

4.2.1 Algorithm development

In this work, the MATLAB® development software was used for the design and selection of the algorithms required in this case. Likewise, the SKIG database (L. Liu & Shao, 2013) was used, from which six dynamic gestures were selected to perform preliminary software tests (see Figure 18). The result of this first phase was the image processing algorithms that allow the gesture to be extracted from the scene, and the algorithms for processing the trajectory of the gesture that allow its description to be carried out appropriately. The selected algorithms are: median filter, segmentation by skin color, segmentation of the disparity map, morphological operation of aperture, calculation of the trajectory of the gesture from the centroid of the shape of the hand, moving average filter applied to the trajectory of the gesture, detection of the gesture from the movement and calculation of the characteristics based on orientation. The stereo correspondence computation was developed independently taking into account that the SKIG base does not have stereoscopic information. However, this includes the depth information that was used for the selection of the aforementioned algorithms. Therefore, the stereo matching algorithm was tested using stereo images from the database presented in (Scharstein, Ugbabe, & Szeliski, 2011). For the training of the HMMs, the Toolbox for MATLAB® developed by Kevin Murphy (Murphy, 1998) was used. The evaluation of the HMMs from a sequence of observations was carried out preliminarily using the advance algorithm using a MATLAB® script in order to verify their operation and facilitate their subsequent implementation.



Circle *Triangle* *Right-Left* *Wave* *"Z"* *Cross*

Figure 18. SKIG database gestures used for the first phase of the recognition system design. Source (L. Liu & Shao, 2013).

The software implementation of the algorithms used in this work is taken as a reference model for the verification of results obtained in hardware. The result of these comparisons indicates whether the FPGA implementation is working correctly.

4.2.2 Architecture selection

The selection of the architecture is carried out by evaluating the algorithms used, taking into account their computational cost and their possibility of being parallelized. The algorithms implemented in hardware in this work are: median filter, stereo correspondence, segmentation by skin color, segmentation of the disparity map, the morphological operation of opening and the extraction of the trajectory of the gesture from the centroid of the shape of the hand. Additionally, the moving average filter and gesture detection from movement were implemented in hardware. The architecture chosen in this work for the implementation of the algorithms in hardware is a full pipeline architecture for online data processing or also called "*Stream Processing*" (Bailey, 2011), which is a form of temporal parallelism and allows the images to be processed at the same capture speed.

High-level algorithms such as feature extraction, HMM, and application control are implemented in software on a conventional processor since the amount of information they must process is low and corresponds only to one point with x,y,z coordinates for each image captured.

4.2.3 Implementation

The final stage in the design process is the implementation of the system. In this, algorithms are projected onto the hardware resources chosen according to the architecture definition. FPGA-based implementations require the design of the specific hardware tasked with carrying out the necessary processing operations (Bailey, 2011). Figure 19 presents the design flow for the FPGA implementation of the recognition system used in this work. For the design and implementation of the system in hardware, the EDA development tool of Altera Quartus II v 14.1 was used.

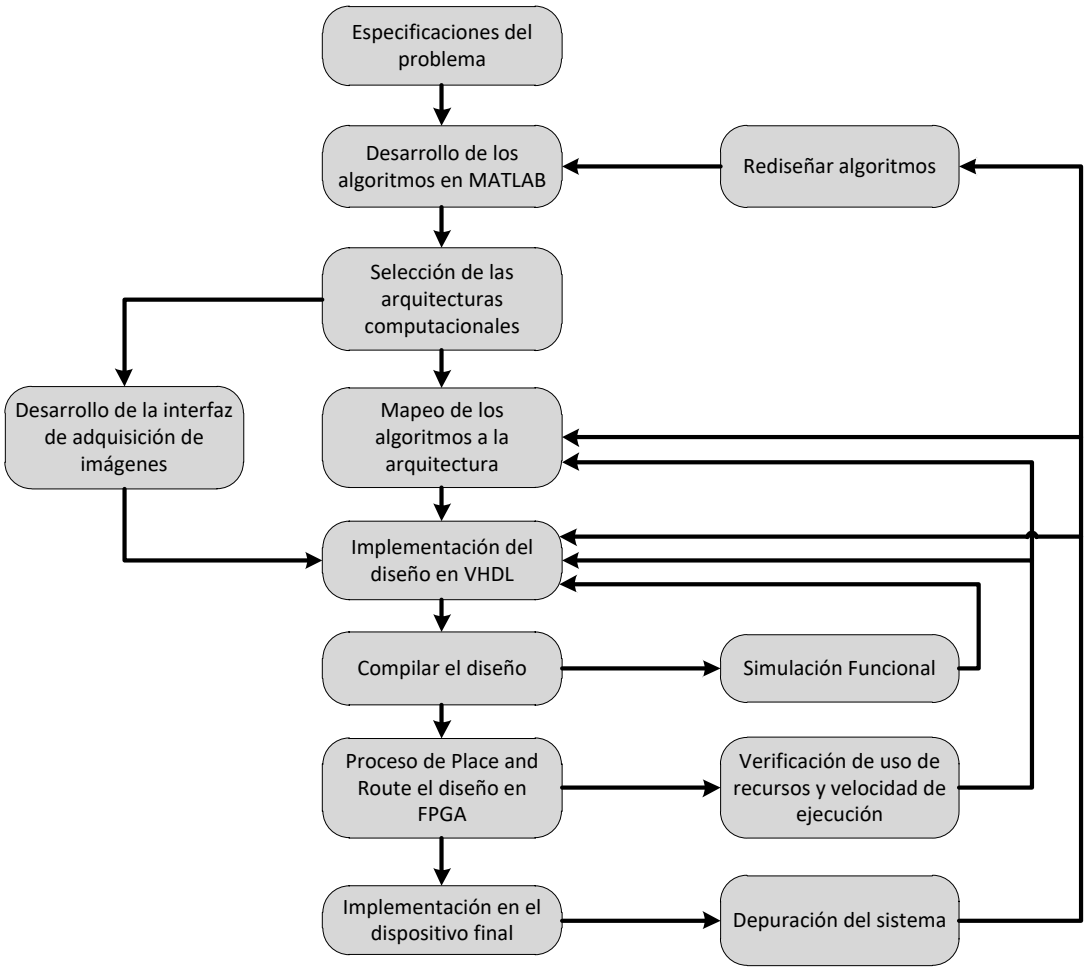


Figure 19. Design flow used for the implementation of the recognition system in FPGA. Figure adapted by the author from (Bailey, 2011)

The recognition system was implemented in Terasic's DE1-SoC development board (see annex A), which has a FPGA CYCLONE V SoC 5CSEMA5F31C6 (see annex A) from the company Altera®, as well as the hardware necessary for the development of computer vision applications. The description of each of the stages of the recognition system was made using the VHDL hardware description language.

The design and implementation of the recognition system was carried out using the *Top-Down* design methodology following a hierarchical structure. Figure 20 presents a general schematic of the architecture developed for the gesture recognition system described in this work. The capture and pre-processing stages are implemented in hardware. The feature extraction stage is implemented in hardware and software. The robot's recognition and control stages are implemented in software. The software-developed applications were implemented using the Hard *Processor System* (HPS) available within the SoC device. In addition, hardware modules were implemented to perform the debugging of the system and to provide feedback to the user on the performance of the gestures. These modules are: a

video multiplexer, a VGA controller to display images on a screen and a RAM controller to synchronize the capture and display speed.

The HPS processor system is a hardware resource offered by SoC systems that allows software applications to be deployed that do not necessarily need to be implemented in hardware directly. This type of processors are already physically deployed and are part of the hardware resources available in the FPGA SoC, so their inclusion in an application does not make use of additional resources and allows all the reconfigurable hardware to be available for the implementation of high-performance operations. In conventional FPGA devices, logic resources must be used to synthesize a processor (such as Altera's NIOS II, Xilinx's MicroBlase, among others) in order to run software applications. This decreases the amount of logical resources for custom hardware design and implementation.

The following chapters present in detail the design and implementation, both in hardware and software, of the gesture recognition system. Each of the designed hardware modules are described from the design of their architecture to the simulation and comparison of the results with the implementations made in MATLAB.®

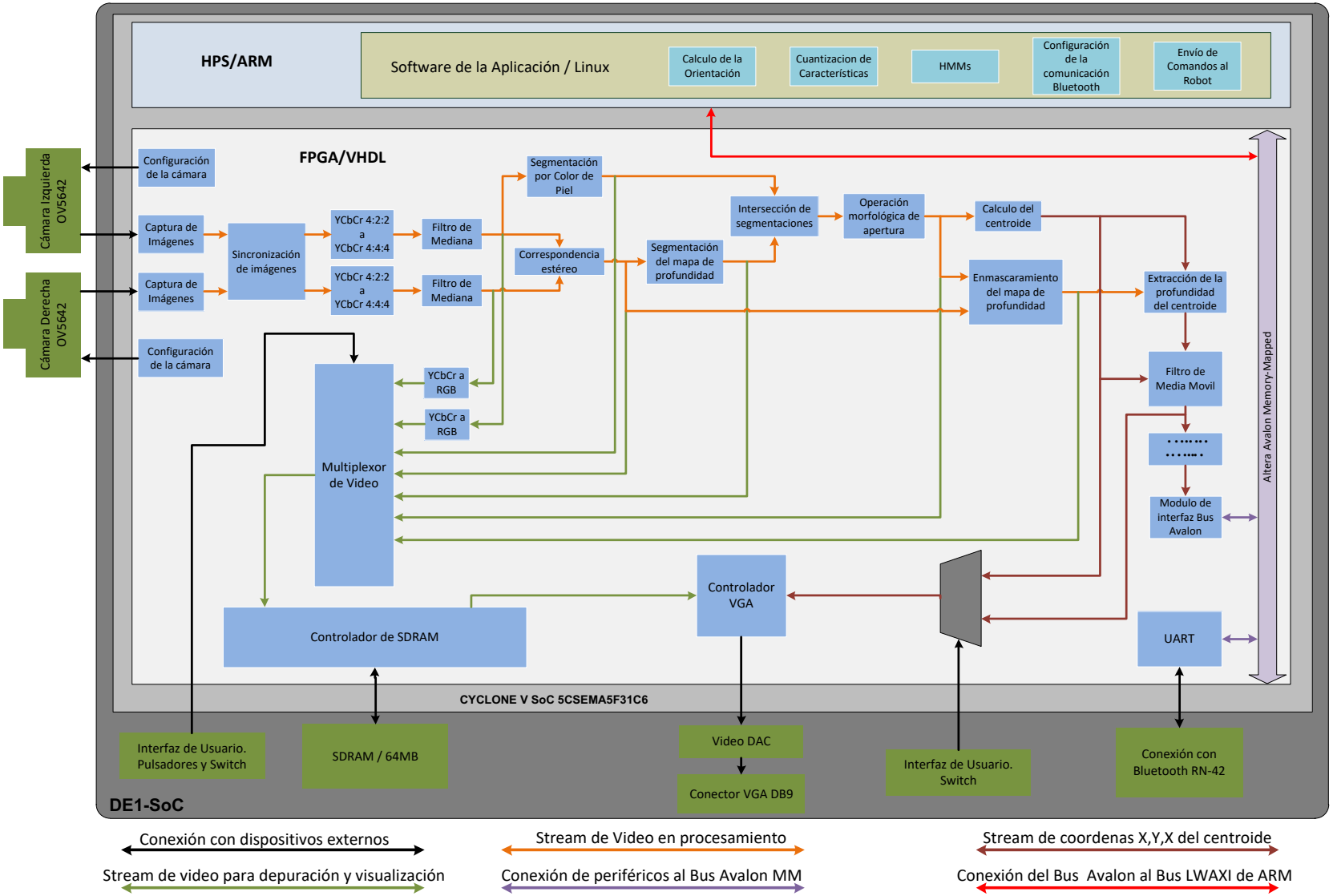


Figure 20. General diagram that exposes the architecture of the developed recognition system. Source: author.

5 Gesture capture

The acquisition of the gestures used in this work is carried out by means of a stereoscopic vision system which consists of two ArduCam digital cameras separated by a distance between their focal axes of 30.48mm. The cameras used consist of an OV5642 CMOS sensor and a 3MP objective lens for CCTV with CS mount. The lens of the cameras has a focal length of 4mm, an aperture of 1:1.4 and an image format of 1/2.5". Figure 21 presents the camera array developed in this work for the capture of stereoscopic images.

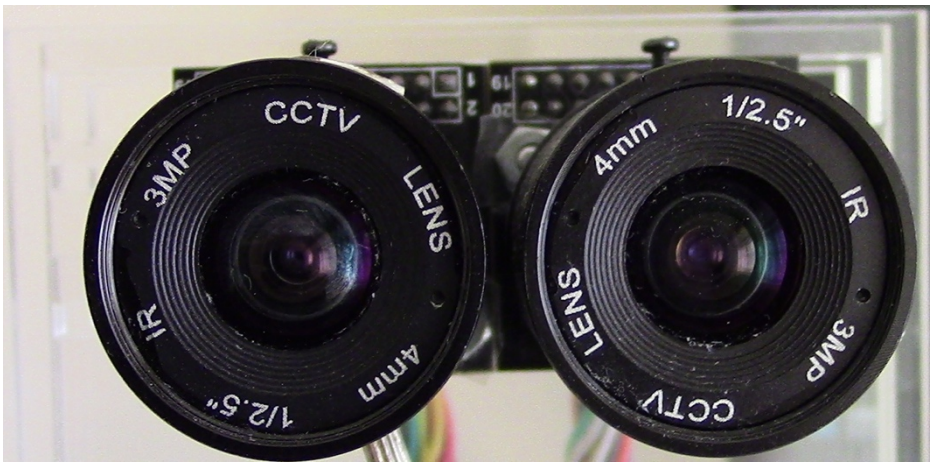


Figure 21. Stereoscopic camera system developed for the application. Source: author.

The camera system connects to the DE1-SOC development board via its GPIO expansion ports. The system's right camera connects to the GPIO_0 port and the left camera to the GPIO_1 port. Figure 22 shows the required interconnection between a camera and the FPGA device. As can be seen in this figure, the connection between the camera and the FPGA requires seven connection signals. The SDA and SCL signals belong to the I2C bus through which the camera initialization is performed. The HREF, VSYNC, DATA[9:0], and PCLK signals are the signals that contain the information in the captured images. The XCLK signal is the external clock source required by the camera to function properly. The sensor is polarized at a voltage of 3.3V (see Appendix A).

To capture the images and carry out their respective processing within the FPGA, it is necessary to design and implement four hardware modules. The camera configuration module is responsible for initializing the sensor by selecting the image resolution and *frame-rate*, among other parameters. The capture module is responsible for receiving an image pixel by pixel by panning from left to right and from top to bottom starting from the upper left part of the sensor. Because the capture of the right and left images is done by separate hardware modules, there may be a lag between these modules, because during the start of the capture either camera may be delayed by a few clock cycles with respect to the other.

To correct this, a hardware module is used to synchronize the two images, since stereo matching and other pre-processing algorithms require it. Finally, a module was implemented that converts the YCbCr 4:2:2 sampling, used by the cameras, to the 4:4:4 format required to obtain a pixel in its three color components, which is necessary to perform the subsequent pre-processing. Below is the design of each of the modules used in this stage of the gesture recognition system.

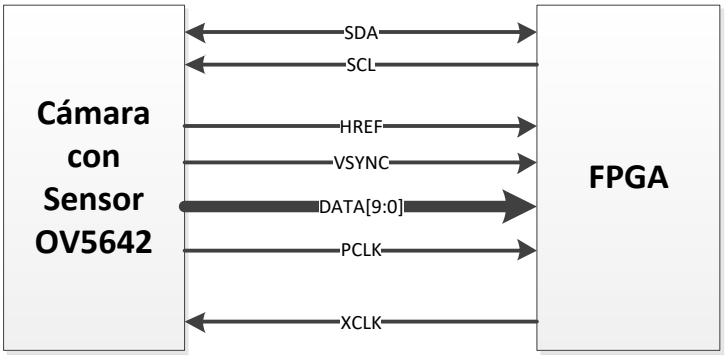


Figure 22. Connection interface between the OV5642 sensor and the CYCLONE V SOC FPGA. Source: author.

5.1 Camera Configuration Module

The digital cameras employed in this work require an initialization process before beginning image capture. Initialization of the cameras consists of configuring the internal registers of the OV5642 sensor in order to establish the following operating parameters: image capture resolution, *frame-rate*, image subsampling (*Binning*), color model, and chrominance subsampling. Table 7 presents the operating parameters of the camera used in this work. Annex B presents the configuration values of each of the internal registers of the sensor, which allow the aforementioned camera operating parameters to be established. The camera configuration values were taken from (Freescale Semiconductor, 2013). (For more information on the OV5642 sensor configuration logs, please refer to annex A).

Table 7. OV5642 Sensor Configuration Parameters

House Resolution	Frame-rate	Image subsampling (*binning*)	Color Model	Color Subsampling
640X480	30 fps	2X2	YCbCr	4:2:2

To carry out the configuration of the camera from the FPGA device, a hardware module was designed that allows each of the camera configuration values to be sent using the I2C synchronous serial communication protocol. To write the information to each of the sensor configuration registers, it is necessary to perform the sequence shown in Figure 23. First, the control word containing the address of the camera is sent, then the upper and lower part of the address of the register to be configured is sent, and finally the data is sent. The camera responds with an ACK='0' signal each time it correctly receives a byte.

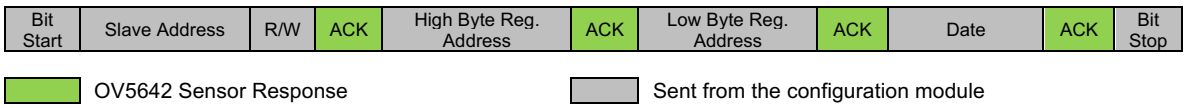


Figure 23. Data sending sequence via I2C bus for configuration of OV5642 sensor logs.

In this work, the TRDB_D5M camera configuration module developed by Terasic was used, which was modified for use with the OV5642 sensor. This module is made up of four main hardware blocks: frequency divider, I2C communication module, LUT (*Lookup-Table*) lookup table and a finite state machine. Figure 24 shows the diagram of the configuration module showing the aforementioned functional blocks and their interconnection. Table 8 describes the I/O ports of the configuration module.

As shown in Figure 24, a crossover module was used to obtain a 400KHz clock signal (from a 50 MHz input signal), which is used in I2C communication. The I2C communication module is responsible for transmitting the information following the sequence established in Figure 23. For this, the module has a 32-bit data input that contains the 4 bytes that are required for transmission, a GO input that enables communication, and two END and ACK outputs that report the status of the transmission. The configuration module stores in a lookup table (LUT) both the addresses of the logs and the data to be sent to the OV5642 sensor. The LUT was configured for the required application taking into account that a total of 988 positions of 24 bits each (16 bits for the register address and 8 bits for the corresponding data) are needed.

To send the configuration data available in the LUT, an FSM (*Finite State Machine*) of 3 Mealy states is used, which works as follows: in the S0 state, the position of the LUT is read as indicated by an index and in turn it is concatenated with the control word 0X78. This control word corresponds to the direction of the OV5642 sensor as a slave for I2C writing. In this same state, transmission begins by GO='1' and the transition to the *S1 state is made*. In the S1 state , NDT and ACK signals are verified. END='1' indicates the completion of a transmission and ACK='0' indicates that the communication was successful. If these two conditions are met, then transition to the *S2* state, otherwise the FSM is restarted. In the *S2* state , the LUT index is increased and the S0 state is transitioned. This procedure is repeated until all the positions of the LUT are read. Annex C presents the state diagram showing the behavior described here. Annex D presents the hardware description files in VHDL that model the behavior of the camera configuration module.

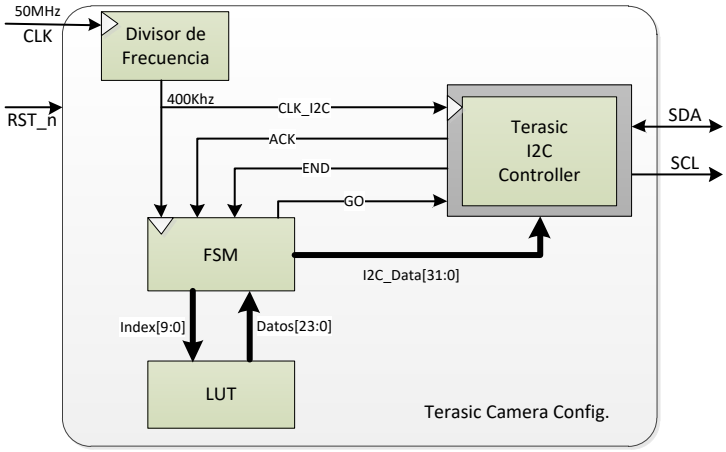


Figure 24. OV5642 sensor initialization module architecture. Source: author.

Table 8. Description of the I/O ports of the Camera Configuration Module.

Port of entry	Function	No. of Bits
CLK	Module clock connected at 50MHz	1
RST_n	Active asynchronous reset under the configuration module.	1
Departure port	**Function**	**No. of Bits**
SCL	I2C Communication Watch	1
Input/output port	**Function**	**No. of Bits**
SDA	Data for I2C communication	1

5.2 Image Capture Module

To capture the images coming from the camera, the behavior of the video signals used by the sensor to transmit the information to the FPGA must be taken into account. Figure 25 shows the timing diagram for the OV5642 sensor. As can be seen, the sensor sends one pixel (10 bits) on each ascending edge of the clock signal (PCLK). To determine whether the pixel sent by the sensor is valid, the behavior of the VSYNC and HREF signals must be taken into account. When VSYNC is '0' and HREF is '1' the pixels sent are valid, otherwise the information that does not correspond to the captured image is considered. When the VSYNC signal has a downward edge, it means that a new image is being transmitted, while an upward edge of this signal indicates that a complete image has been sent. When the HREF signal is '1' the sensor is sending a row of the image pixel by pixel.

The OV5642 sensor sends an image by panning the pixels from top to bottom and from left to right, so the first pixel sent corresponds to the upper left corner and the last pixel sent corresponds to the lower right corner of the image (see Figure 12). This feature allows you to implement parallel image processing by applying the concepts of temporal parallelism.

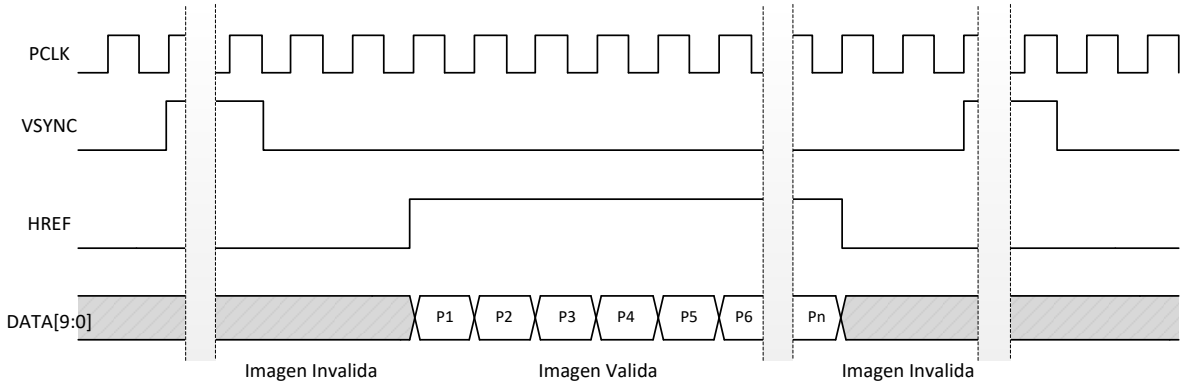


Figure 25. Time diagram of the signals from the OV5642 sensor used in image capture. Source: author.

In this work, the camera was configured with chrominance subsampling YCbCr422. This subsampling format configures the sensor to send the luminance component Y in one clock cycle, and in the next cycle the chrominance component of a pixel Cb or Cr, as appropriate. In order to facilitate the conversion to the YCbCr444 format, the capture module was designed. In this, the input sequence (8 bits) is transformed by concatenation of the Y component with the next component (Cb or Cr), to obtain a pixel with two components (16 bits) in a single clock cycle. Figure 26 shows the input sequence to the capture module and the output sequence generated. As can be seen, the Y1 component enters as a pixel independent of the Cb component. However, the pixel is actually made up of these two components, therefore, when grouped together, a single piece of data is obtained that is interpreted as a pixel at the output Po_1.

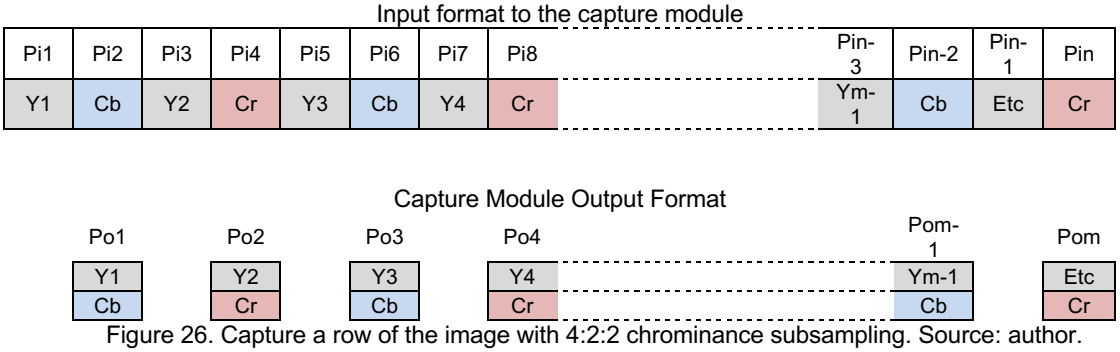


Figure 26. Capture a row of the image with 4:2:2 chrominance subsampling. Source: author.

Figure 27 presents the internal structure of the capture module and describes the process of concatenation of the captured data. This is done by cascading logs, sending the information from the last two logs through a port. To determine whether a captured pixel is valid, two hardware blocks were developed. The first determines the behavior of ports i_fval (connected to VSYNC) and i_lval (connected to HREF) according to the time diagram in Figure 25. The second is a counter that determines the position of the captured pixel within the row. The validity signal of the captured pixel is generated using the least significant bit of the counter, since the output pixels are obtained at half the input frequency (see Figure 26). Additionally, another counter is used to determine the row in which the pixel is located.

The count values of the rows and columns are sent abroad through two independent ports. Ports i_clk and i_rst are the clock and reset inputs for the module and are connected to all internal blocks. Table 9 presents a detailed description of the input-output ports of the designed module. Annex E presents the simulation results of the capture module obtained using the Modelsim-Altera software.

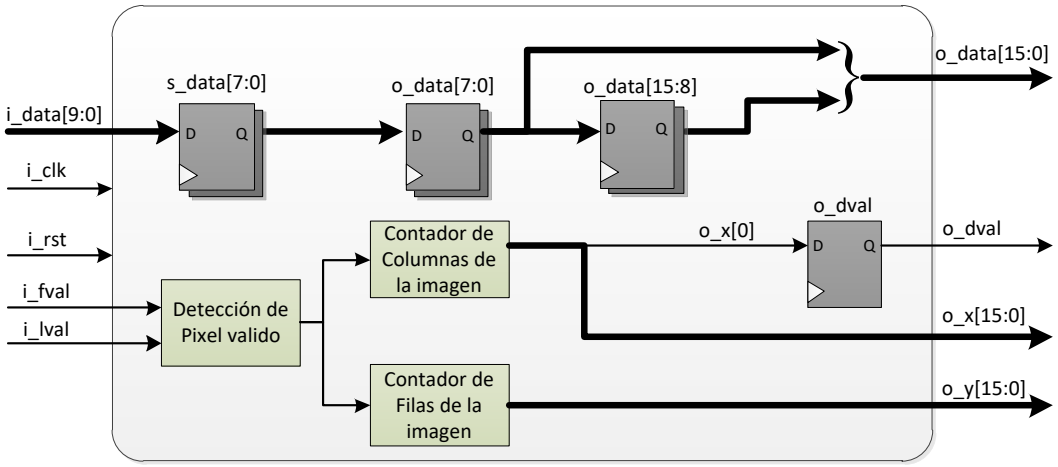


Figure 27. Internal architecture of the image capture module. Source: author.

Table 9. Description of the I/O ports of the Image Capture Module.

Port of entry	Function	No. of Bits
i_rst	Asynchronous module reset, its performance is active under	1
i_clk	Clock for capturing the image coming from the camera. Connects to PCLK	1
i_fval	Valid frame from the camera. Connects to VSYNC	1
i_lval	Valid line coming from the camera. Connects to HREF	1
i_data	Sensor data from the camera. Connects to DATA[9:2]	8
i_start	Start Image Capture User Interface	1
I_end	Pause Image Capture User Interface	1
Departure port	**Function**	**No. of Bits**
o_dval	Determine the validity of the captured pixel for the following steps	1
o_data	Pixel captured for the next stages	16
o_x	x-coordinate within the image to which the captured pixel belongs	16
o_y	Coordinate and within the image to which the captured pixel belongs	16

5.3 Stereo Image Synchronization Module

Stereo images obtained directly from the cameras are not synchronized as separate hardware modules are used for configuration and capture. Each camera has its own clock signal, so it is not possible to guarantee the simultaneous capture of the pixels of the two images. In addition, in the subsequent processing steps, especially in stereo correspondence, it is required that the pixel corresponding to the same coordinate for the left and right image be available in a clock cycle. For this, a hardware module was designed that allows correcting or synchronizing the captured images. This module independently captures the two images and then sends them to the next processing modules using a single clock signal. Figure 28 presents the architecture of the synchronization module developed in this work and Table 10 presents a description of its input-output ports.

The synchronization module is composed of three hardware blocks: one FSM and two FIFO memories, one for the left image and one for the right image. FIFO memories store the pixels coming from the capture modules of the left and right cameras, respectively. When these memories have stored an amount greater than N pixels, the FSM starts the process of reading N pixels in both memories. The FSM also generates an output that indicates the validity of the pixels sent to the outside of the synchronization module. In the process of reading FIFO memories, a clock is used that is independent of the clock signals used in writing. This allows the following processes to have the left and right images at their disposal, as well as the data flow required to apply parallel processing.

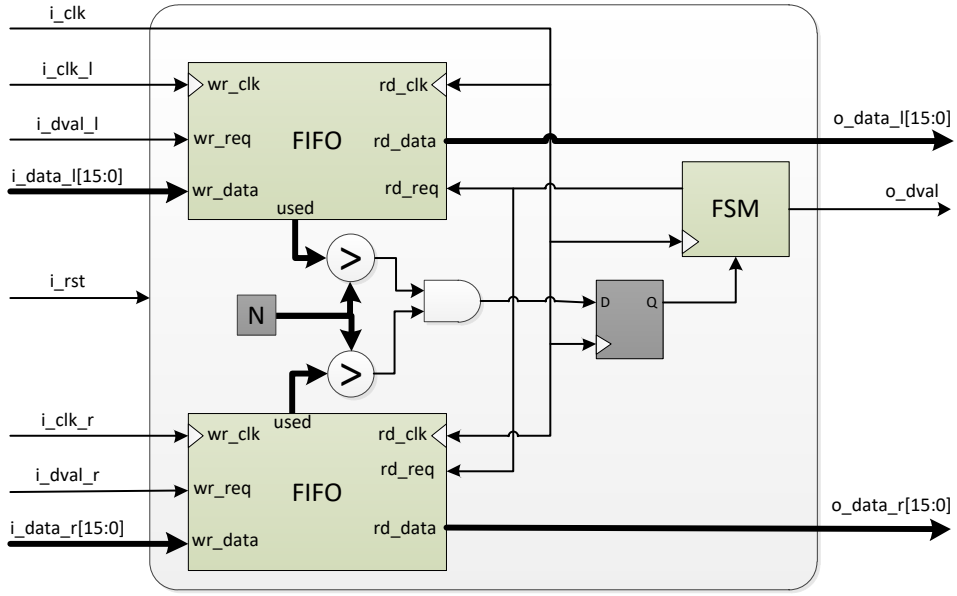


Figure 28. Internal architecture of the stereo image synchronization module. Source: author.

Table 10. Description of the I/O ports of the Synchronization Module.

Parameters	Function	
N	This parameter allows you to choose the minimum number of pixels that images require for synchronization	
Port of entry	**Function**	**No. of Bits**
i_clk	Module clock for stereo image synchronization	1
i_rst	Reset the active module at a low logical level	1
i_clk_l	Clock from the left camera	1
i_dval_l	Input that determines the validity of a pixel coming from the left camera	1
i_data_l	Input data containing one pixel from the left camera	16
i_clk_r	Clock from the right camera	1
i_dval_r	Input that determines the validity of a pixel coming from the right camera	1
i_data_r	Input data containing one pixel from the right camera.	16
Departure port	**Function**	**No. of Bits**
o_dval	Output indicating the validity of the sync for a pixel from the left and right cameras	1
o_data_l	Output data containing one pixel from the left camera.	16
o_data_r	Output data containing one pixel from the right camera.	16

5.4 YCbCr422 to YCbCr444 Conversion Module

As mentioned above, the sensor used in this work was configured to capture images by subsampling YCbCr422 chrominance. This type of subsampling allows images to be

obtained from the cameras at a lower pixel rate compared to using full sampling. This is because the amount of information for each pixel is limited to only two color components: the luminance component Y and a single chrominance component (Cb or Cr). However, the subsequent processing stages require images with pixels represented in all three color components. For this reason, it is necessary to do a conversion process that consists of transforming 1 pixel of two components into 1 pixel of three components or YCbCr444. The hardware module designed to perform this conversion is presented in Figure 29. Table 11 presents a detailed description of the input-output ports that make up the designed module.

For the design of the hardware module that performs this conversion, the information sent from the capture module must be considered, as explained in section 5.2. The captured pixels corresponding to the even-numbered columns of the image contain the Y and Cb components, while the pixels of the odd-numbered columns contain the Y and Cr components respectively (see Figure 26). Taking into account this capture order for the pixels in a row of the image, the process of converting YCbCr422 to YCbCr444 is developed using the simple replication method of the Cb and Cr components presented in (Gregg Hawkes, 2001). This method of replication consists of taking two consecutive pixels (P1={Y1,Cb1} and P2={Y2,Cr2}) and from these pixels obtaining two new pixels by replicating the components Cb and Cr for which their luminance components Y are maintained, in such a way that the result is two pixels in three components ($P_{1=\{Y1}$, Cb1,Cr2} and P2={Y2,Cb1,Cr2}).

In the present work, the design of a hardware module was carried out to perform the conversion of YCbCr422 to YCbCr444. As shown in Figure 29, the hardware module developed for this conversion consists of three fundamental elements: an FSM, a multiplexer, and registers. FSM determines the validity of an input pixel and evaluates the column to which that pixel belongs within the image. In case the column is even, the input data is stored in the Y1 and Cb1 registers, and if the column is odd, the input data is stored in the Y2, Cr1 and Cr2 registers, and the Cb2 register loads the information contained in Cb1. To send the correct data to the outside, the FSM controls a multiplexer by sending Y1, Cb1 and Cr1 to the output if the pixel belongs to an even column, or Y2, Cb2 and Cr if it belongs to an odd column. The developed hardware module has a fixed latency of three clock cycles.

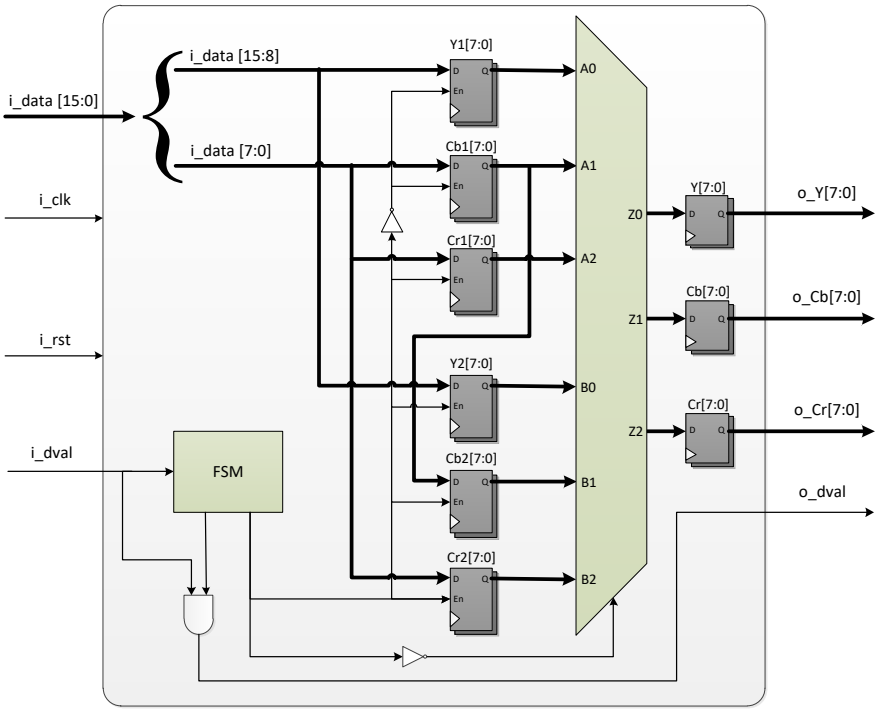


Figure 29. Internal architecture of the YCbCr422 to YCbCr444 conversion module. Source: author.

Table 11. Description of the I/O ports of the conversion module is YCbCr 4:2:2 to 4:4:4.

Port of entry	Function	No. of Bits
i_clk	Module Clock Input	1
i_rst	Active asynchronous reset input at low logic level	1
i_dval	Input that determines the validity of a pixel	1
i_data	Input data containing one pixel in YCbCr422 format	16
Departure port	**Function**	**No. of Bits**
o_y	Luminance Component Output Data Y	8
o_cb	Output data of the Blue Chrominance Cb component	8
o_cr	Output data of the Red Chrominance Cr component	8
o_dval	Output that determines the validity of a pixel in its three components YCbCr	1

Annex D presents the VHDL description of the YCbCr422 to YCbCr444 synchronization and conversion modules. Annex E presents the simulation results obtained using the Modelsim-Altera software for each case.

6 Pre-processing

This chapter describes the algorithms involved in the pre-processing stage of the recognition system. Pre-processing involves the application of algorithms in a certain order in order to extract the shape of the hand from the scene. First, the median filter is applied to the input images in order to eliminate the noise that occurs on the sensor at the time of capture. Once the images have been filtered, the disparity map is obtained using a stereo matching algorithm. The result of this operation is filtered using the median filter in order to smooth the resulting result. To extract the shape of the hand from the scene, a segmentation process is applied consisting of skin color segmentation and disparity map segmentation. The segmentation process produces noise that is filtered through the morphological operation of opening.

The pre-processing stage algorithms implemented in hardware have a computational architecture for *Stream Processing*. This architecture allows you to take advantage of the flow of data from the camera as it enters the device through the Raster scan format, which delivers a pixel on each ascending flank of the clock signal. The designed module has a control signal that indicates the validity of the data delivered by each module, which allows the image to be processed at the same time as it is captured. Figure 30 presents a general outline of the pre-processing stage implemented in FPGAs. This figure shows the flow of data and control signals through the modules involved in this stage.

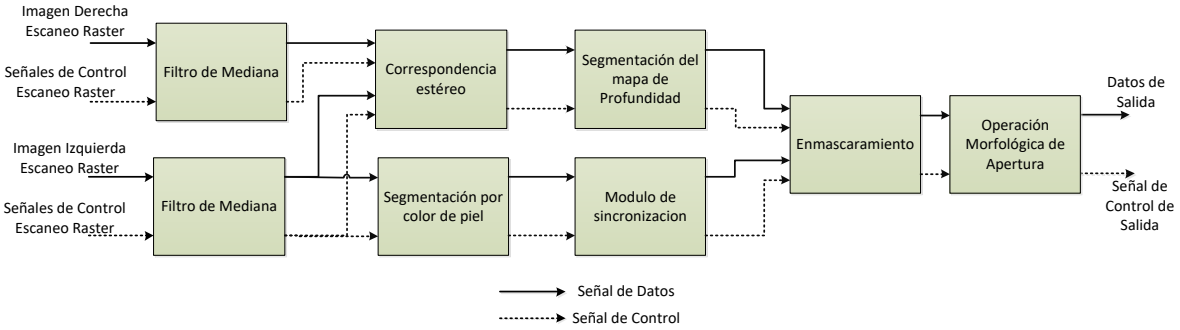


Figure 30. Interconnection of the components that make up the pre-processing stage. Source: author.

6.1 Median Filter

The images obtained from the cameras contain noise that is mainly due to the characteristics of the sensor used and the chrominance subsampling method used in the capture stage. For this reason, the median filter is used to eliminate noise without distorting the information contained in the images. Although this filter efficiently eliminates impulsive noise (known as salt and pepper), it can also be used in the elimination of Gaussian noise, since unlike other filters such as the average one, the resulting image does not present too much blur. The

median filter consists of replacing the value of one pixel in the image with the median of its neighborhood W. This allows you to correct pixels whose intensity is very different from that of their neighbors, obtaining a smoothing or elimination of noise in the image.

By definition, the median filter is a nonlinear spatial filter (Bailey, 2011). This filter takes a W neighborhood for each pixel within the image and sorts the values of the pixels within that neighborhood from lowest to highest. The filter output is the value of the median obtained in the sorting process. Due to the characteristic of the filter, a relatively large computational cost is required if it is implemented in software on a conventional processing architecture. However, such a filter can be implemented in parallel, making it ideal for application in hardware.

```
1:  procedure MedianaIm(I_i)
2:      Input: Una Imagen de entrada I_i
3:      Output: Imagen Filtrada I_o
4:      I_W ← ancho de la imagen I_i en pixeles
5:      I_H ← alto de la imagen I_i en pixeles
6:      N_w ← Tamaño de la ventana del filtro
7:      V[N_w × N_w] ← Vector Temporal para construccion de la ventana del filtro
8:      for j ← 2 , I_W do          ▷ Recorrido de la ventana sobre las columnas de la imagen
9:          for i ← 2 , I_H do          ▷ Recorrido de la ventana sobre las filas de la imagen
10:             w ← 0
11:             for l ← 1 , N_w do          ▷ Extracción de la Ventana a procesar, en un vector V
12:                 for k ← 1 , N_w do
13:                     w ← w + 1
14:                     V[w] ← I_i[i − 2 + l, j − 2 + k]
15:                 end for
16:             end for
17:         end for
18:         I_o[i, j] ← MEDIANA(V)          ▷ Calculo de la mediana sobre la ventana V
19:     end for
20:     return I_o
21: end procedure
```

Figure 31. Algorithm Corresponding to the median filter applied to an input image *Ii*

Taking as a reference the design process proposed in section 4.2, for the design and implementation of image processing algorithms in FPGAs it is preferable to initially implement this algorithm in a software tool. This allows for a better understanding of how it works and the development of the required hardware architecture. In this work, the design of the median filter was carried out in MATLAB® and then the respective mapping in the desired hardware architecture was developed. Figure 31 shows the algorithm for applying the median filter to an image. In this figure you can see the path of the window over the image and the application of the filter for each of the displaced windows. Figure 32 presents the algorithm corresponding to the calculation of the median following the bubble method

for data ordering. Annex F presents the development of the median filter using MATLAB® software.

```
1: procedure MEDIANA(V)
2:     Input: Un vector con el contenido de la ventana a procesar V
3:     Output: Calculo de la mediana M
4:     N_v ← Tamaño del vector V
5:     for j ← 1 to N_v do                ▷ Algoritmo de burbuja para ordenar los elementos de V
6:         for i ← 1 to N_v − j do
7:             if V[i] > V[i+1] then
8:                 aux ← V[i]
9:                 V[i] ← V[i+1]
10:                V[i+1] ← aux
11:            end if
12:        end for
13:    end for
14:    return V[N_v/2 + 1]                ▷ Retorno del valor correspondiente a la mediana
15: end procedure
```

Figure 32. Algorithm Corresponding to the calculation of the median applied to an input vector *V*

6.1.1 Hardware architecture

The design of the median filter in FPGA is made to be compatible with computational architecture (*Stream Processing*). The median filter receives a new pixel each clock cycle, processes it, and sends a processed pixel to the next module for further processing. The hardware module for the median filter is designed using algorithm mapping techniques known as *pipelining* and *row buffering*. A detailed description of these mapping techniques is presented in section 3.3.3. These mapping techniques allow a new pixel to be processed in each clock cycle and leads to executing the operation through temporal parallelism. The designed module is made up of three functional blocks: 3X3 moving window generator, median filter and validity signal generator for the output pixel. The flow of input data and control signals from one functional block to another is shown in Figure 33.

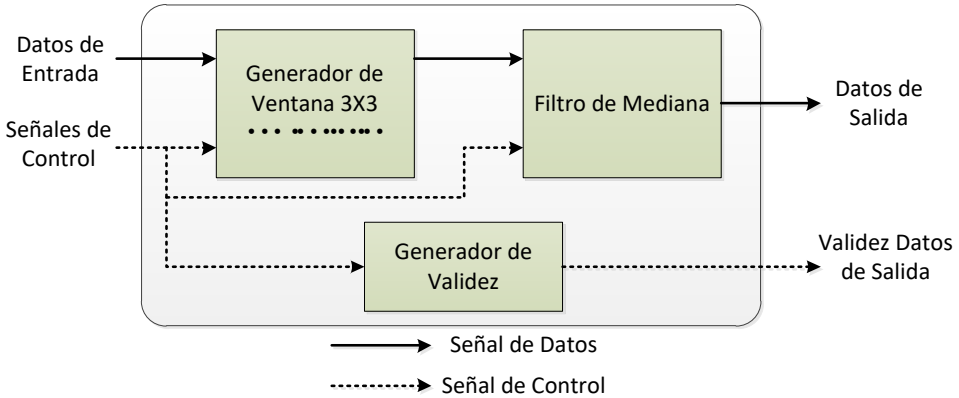


Figure 33. Median filter hardware module. Flow of data signals and control. Source: author.

Figure 34 shows the schematic diagram of the hardware module for the median filter. This diagram shows the module's connection interface. The module has two generic parameters N used M in the component synthesis process. The parameter N allows you to select the number of bits of the module's input and output data. The parameter M sets the width of the image for the construction of the *Row Buffers*. On each ascending edge of the clock signal (i_clk) an input data is captured and in turn a new data is generated at the output. The asynchronous restart signal (i_rst_n) is active at a low logic level and allows the module to be restored at any time. The valid pixel signal (o_valid) is set to high if the output pixel is valid, as long as the input pixel is valid (i_valid at a high logical level). The input signal i_valid acts as an enabler of the module. When i_valid is in low the data within the module does not change and the input data is ignored.

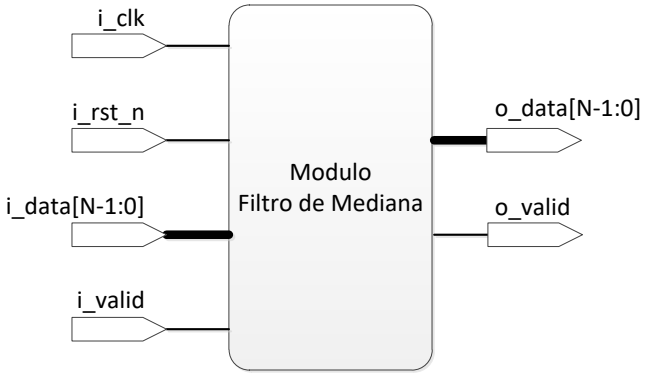


Figure 34. Schematic diagram of the Median filter module. Source: author.

6.1.2 Mobile Window Generator

The mapping method for *Row Buffering* is employed by constructing a mobile window architecture. This type of architecture allows pixels to be accessed in consecutive rows of the image without the need to store the entire image in external memory (see section 3.3.3.2), thus maintaining the continuous data flow required in subsequent processing stages. In this work, a 3X3 window was selected in such a way that two *Row Buffers* are required to cache the values of the pixels of two preceding rows. Nine registers are also required to store the window values before they are processed by the module.

The window architecture reads pixels coming from the outside row by row in a raster scan. An input pixel enters the register W_1 and shifts to the right on each clock cycle (see Figure 35). Pixels continuously move from left to right over registers and buffers. When the two *Row Buffers* are filled and the first pixel of the image reaches the register W_5 , the first 3X3 window is ready to be processed.

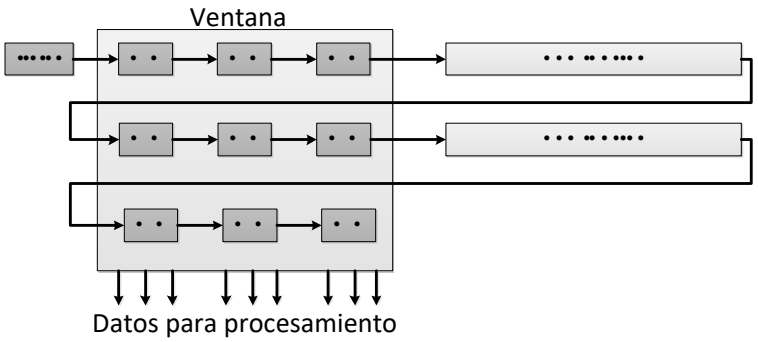


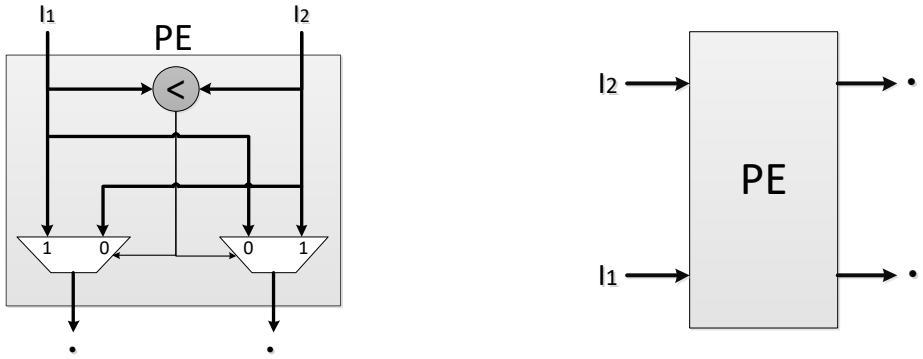
Figure 35. Cache architecture employing *Row Buffering*. Adaptation from (Bailey, 2011).

6.1.3 Ordering network for the calculation of the median

The median filter takes the nine input data coming from the moving window generator and gets its median. As described in the algorithm in Figure 32, the median filter consists of ordering several elements, and then selecting the value of the median resulting from this process. The hardware implementation of this algorithm is carried out through ordering networks. A *sorter network* (SN) is defined as a network of elementary operations denoted as *compare&swap* (CS) (Vasicek & Sekanina, 2008). The number of CS operations in an SN is determined by the number of items to be sorted. SNs are characterized by their implementation in hardware being completely parallel and allow the development of a *pipeline architecture*.

According to (Habermann, 1972; Lakshmivarahan, Dhall, & Miller, 1984) there are several SNs within which the odd-even transposition ordering network is found. This ordering network consists of making comparisons in a similar way to the process carried out by the bubble algorithm. The operation of this method is carried out by comparing pairs of elements with odd/even indexes. If a pair of elements is in the wrong order, the elements are reordered. The next step is to compare adjacent elements with odd/even indexes. The process is repeated alternately until the list of items is completely sorted.

The CS processing element is the fundamental component of the odd-even transposition SR. Figure 36 presents the corresponding architecture, which consists of a magnitude comparator and two multiplexers. The input data I_1 and I_2 enter the comparator which evaluates whether $I_2 > I_1$. If true, the output of the comparator is set to 1 controlling the multiplexers so that the data comes out in the same order of input (without exchange) so and $H = I_2 \quad L = I_1$. In case the comparison is false, the input data is exchanged obtaining $H = I_1$ and $L = I_2$



(a) (b)

Figure 36. CS Comparison and Exchange Processing Element. (a) internal architecture. (b) schematic diagram. Source: author.

The implementation of the SN in FPGAs is carried out through layers of interconnected CS processing elements following the process described by (Habermann, 1972). In the first layer of the network, the values from the moving window generator are taken, and grouped in pairs taking the data with odd-even indices (from top to bottom) for each CS. The second layer has a similar behavior, but this time taking the data with odd-even indexes. To carry out the sorting of 9 data, 9 processing layers are required with 4 comparison elements per layer. To carry out pipeline processing, simply put records between each of the processing layers, obtaining an architecture with 9 pipeline stages. Figure 37 shows the complete construction of the SN.

The values that enter the SN are ordered from highest to lowest, with position 9 being the maximum value and position 1 being the minimum value. Therefore, to obtain the median of the input data, the value obtained at position 5 is taken. To obtain a valid median value, the latency of the SN and the latency of the moving window generator must be taken into account, so an independent hardware module is required to determine this state.

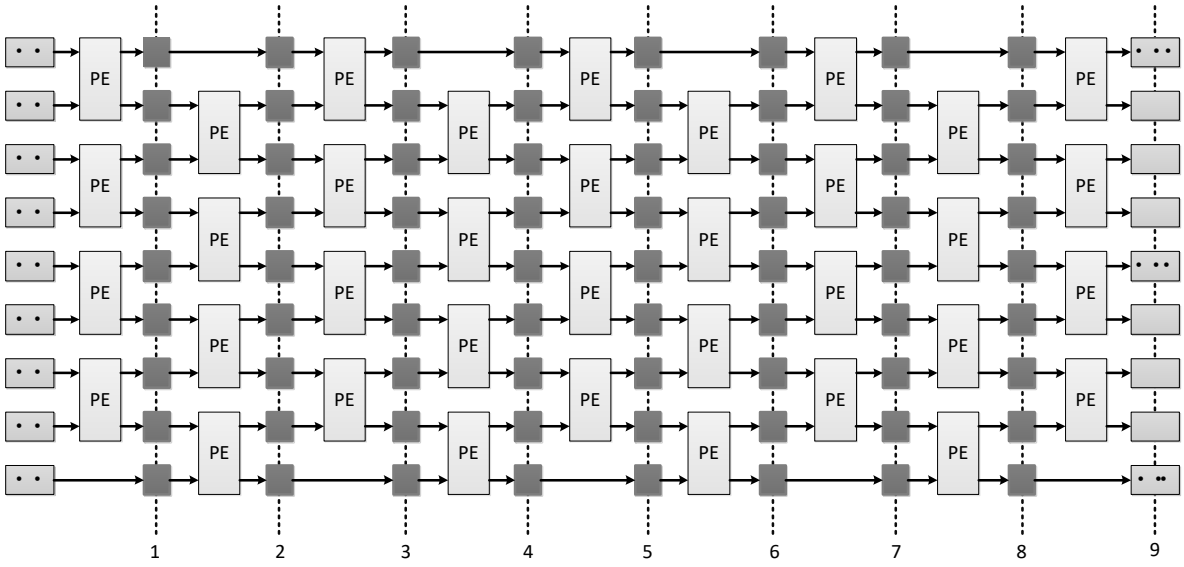


Figure 37. Full-pipeline architecture of the median filter based on an odd-even transposition ordering network. Source: author.

6.1.4 Output Data Validity Signal Generator

In order to know if the filter's output data is valid, a validity signal called *o_valid is required*.

The median filter module starts producing valid data when the first pixels have propagated through the window generator and the SN. According to the time diagram in Figure 38, it can be seen that the generation of valid data (signal *o_valid* high) must begin after the entry of n valid data into the module. From that point on, valid output data is generated as long as the control signal (*i_valid*) is high. This implies that there is a latency n in the processing which is determined by the width of the image to be processed, the size of the window and the number of *pipeline stages* of the SN.

The total latency n of the median filter module is calculated by taking into account the latency produced by the moving window generator and the SN. To obtain the first valid window to be processed $M+2$, valid input data is required, being M the width of the image. Processing a window using the SN requires 9 valid data or clock cycles. Therefore, the total latency of the module is $n = M + 11$ valid clock cycles.

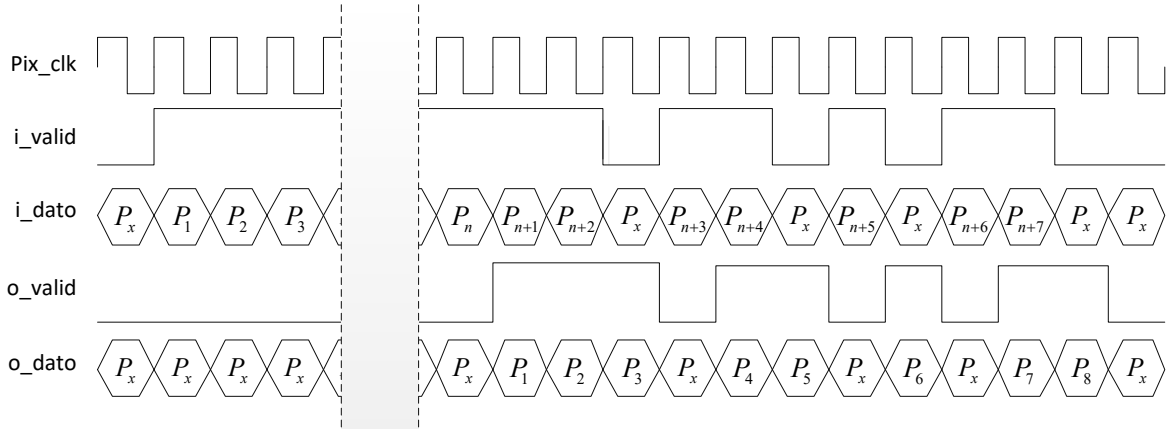


Figure 38. Behavior of the median filter over time. Source: author.

Taking into account the behavior and total modulo latency of the median filter, a functional block is developed that allows the correct generation of the output signal *o_valid*. Figure 39 shows the schematic circuit corresponding to this functional block. To wait for the first valid clock cycles, an ascending counter is used, which is incremented to the value $M+10$. The counts are controlled by the input signal *i_valid* and by a magnitude comparator. The magnitude comparator in turn controls the output signal *o_valid*. Therefore, as long as the counter is less than $M+10$ the validity signal, it remains low regardless of the state of the i_valid input. When the counter is equal to $M+10$ the validity signal it is controlled by the i_valid input.

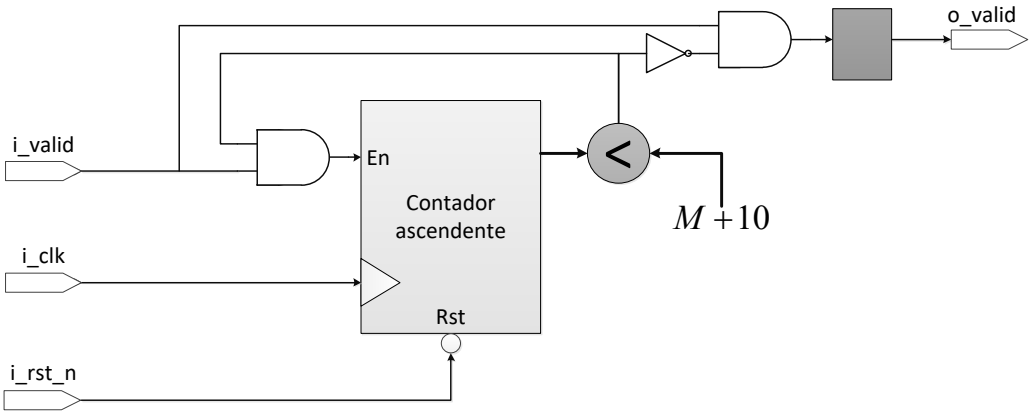


Figure 39. Generator of the validity signal of the processed output pixels. Source: author.

6.2 Stereo Correspondence

The stereo matching algorithm is initially implemented in software using MATLAB.® The disparity map obtained in hardware is compared with the disparity map obtained in software; the result of this comparison indicates whether the FPGA implementation is working

correctly. The development of the stereo matching module is presented in detail below, starting with its implementation in software and the subsequent development of its hardware architecture.

According to (Fife, 2011; Fife & Archibald, 2013) the stereo vision method based on the non-parametric Census transformation provides superior correlation accuracy compared to other area-based stereo vision methods. Using this method, the calculation of the disparity map is carried out by first applying the Census transformation to each of the stereo images. In this work, the 50% dispersed Census transform was used as explained in sections 3.1.1 and 3.1.2., since similar results are obtained to the original transformation, but reducing by half the amount of resources (especially memory) required for implementation in FPGA (Humenberger et al., 2010). Once the transformation has been carried out, the degree of similarity between the images is evaluated using the correlation method called Hamming Sum of Distances (SHD). To improve the result of the disparity map, the Left-Right Consistency Check algorithm (LRCC) is used.

Figure 40 presents the algorithm that describes its implementation in software of the 50% sparse Census transform. The image is traversed from top to bottom and from left to right. For each window centered on, $I_i[i,j]$ a string of bits is obtained $\frac{W_c^2-1}{2}$, the result of comparing the central pixel and its neighbors taking into account the mask in Figure 7.b. The result of the Census transform is an array of dimensions $I_H \times I_W \times \frac{W_c^2-1}{2}$, where I_W e I_H is the width and height of the original image and W_c the size of the Census window.

After applying the Census transform to the left and right image, the disparity map is calculated following the algorithm shown in Figure 41. This algorithm allows the calculation of disparity maps using the left image M_{L2R} and the right image M_{R2L} as a reference. The calculation of the disparity map is carried out by hovering over the images, the comparison windows in the direction from top to bottom and from left to right. Each window in the reference image is compared to d windows in the target image. To get the L2R *left-to-right* map, the window on the left image centered on $[i,j]$ is compared to d windows on the left of the right image. For the *R2L* right-to-left map, the right-to-left image window centered on $[i,j-d]$is compared to d windows to the right of the left image. The comparison of these windows is carried out using the SHD cost function. The result of comparing the reference window with the target windows in each disparity map is a length vector. d The disparity value is determined by the position of the vector, whose comparison value between windows is minimal.

1: **procedure** TRANSFORMADA CENSUS DISPERSA(I_i,W_c)
2: **Input:** I_i: Imagen de entrada en escala de grises
3: W_c: Tamaño de la ventana de Census
4:
5: **Output:** I_o: Imagen Transformada
6:
7: $I_W \leftarrow$ ancho de la imagen I_i en pixeles
8: $I_H \leftarrow$ alto de la imagen I_i en pixeles
9: $I_o[I_W, I_H, \frac{W_c^2-1}{2}] \leftarrow$ Declaracion de la imagen de salida e inicializacion en 0
10: $d \leftarrow \frac{W_c-1}{2}$ parámetro requerido para la transformada dispersa de Census
11:
12: **for** $i \leftarrow d+1$, $I_W - d$ **do** ▷ Recorrido por las Filas de la imagen
13: **for** $j \leftarrow d+1$, $I_H - d$ **do** ▷ Recorrido por las Columnas de la imagen
14: **for** $m \leftarrow 1$, $d+1$ **do** ▷ Recorrido Filas impares de la ventana w
15: **for** $n \leftarrow 1$, d **do** ▷ Recorrido Columnas pares de la ventana w
16: **if** $I_i[i-(d+1)+2\times m-1, j-(d+1)+2\times n] \geq I_i[i,j]$ **then**
17: $I_o\left[i, j, int\left[\frac{w\times(2\times(m-1))+2\times n+1)}{2}\right]\right] \leftarrow 1$
18: **end if**
19: **end for**
20: **end for**
21: **for** $m \leftarrow 1$, d **do** ▷ Recorrido Filas pares de la ventana w
22: **for** $n \leftarrow 1$, $d+1$ **do** ▷ Recorrido Columnas impares de la ventana w
23: **if** $I_i[i-(d+1)+2\times m, j-(d+1)+2\times n-1] \geq I_i[i,j]$ **then**
24: $I_o\left[i, j, int\left[\frac{w\times(2\times(m-1)+1)+2\times n)}{2}\right]\right] \leftarrow 1$
25: **end if**
26: **end for**
27: **end for**
28: **end for**
29: **end for**
30: **return** I_o
31: **end procedure**

Figure 40. Algorithm for the Software implementation of the Scattered Census transform.

The calculation of the difference between two windows used to calculate the disparity map is carried out using Hamming's Sum of Distances. The algorithm that describes this method is presented in Figure 42. To find the degree of difference between two windows of equal size, the XOR logic operation is applied between the windows and then the 1's obtained in the operation are counted. This logical operation allows you to find the difference between strings of bits, because if two bits of different strings are compared, the result is 1 if the bits are different and 0 if they are the same. This indicates that the amount of 1's allows you to determine the degree of difference between the windows, 0 indicates that the windows do not have different elements, and a value $S > 0$ indicates that the windows are bit differentiated S from each other.

procedure MAPA DE DISPARIDAD(I_{left},I_{rigth},W_h,d)
  Input: I_{left}: Transformada census de la Imagen izquierda
    I_{rigth}: Transformada census de la Imagen derecha
    W_h: Tamaño de la ventana de correlacion de SHD
    d: cantidad de pixeles de disparidad

  Output: M_{L2R}: Mapa de disparidad Resultante imagen izquierda como referencia
    M_{R2L}: Mapa de disparidad Resultante imagen derecha como referencia

  $I_W \leftarrow$ ancho de la imagen I_i en pixeles
  $I_H \leftarrow$ alto de la imagen I_i en pixeles
  $W_d \leftarrow \frac{W_h-1}{2}$ Parámetro para recorrido de la ventana
  $Costo_{L2R}[d] \leftarrow 0$ Vector para almacenar el calculo de similitud para las d disparidades
  $Costo_{R2L}[d] \leftarrow 0$ Vector para almacenar el calculo de similitud para las d disparidades

  for $i \leftarrow W_d + 1$, $I_H - W_d$ **do** ▷ Recorrido por las Filas de la imagen
    for $j \leftarrow W_d + d$, $I_W - W_d$ **do** ▷ Recorrido por las Columnas de la imagen
      for $k \leftarrow 1$, d **do** ▷ Recorrido por las d ventanas candidatas
        $Costo_{L2R}[k] \leftarrow SHD\left\{ I_{left}\Big[Ventana[i,j]\Big], I_{rigth}\Big[Ventana[i,j-k]\Big] \right\}$
        $Costo_{R2L}[k] \leftarrow SHD\left\{ I_{left}\Big[Ventana[i,j-d+k]\Big], I_{rigth}\Big[Ventana[i,j-d]\Big] \right\}$
      end for
      $M_{L2R} \leftarrow \arg\min_d \{Costo_{L2R}[d]\}$
      $M_{R2L} \leftarrow \arg\min_d \{Costo_{R2L}[d]\}$
    end for
  end for
  return M_{R2L}, M_{L2R}
end procedure

Figure 41. Algorithm for the implementation of disparity calculation software.

To improve the result of the disparity map there is a method called Left-Right Consistency Check (LRCC). This method is widely for the perfection of the results of any stereo matching method. This method checks the correspondence between previously obtained disparity maps. If a pixel with coordinates $[\mathrm{x}_l, \mathrm{y}_l]$ in the left image has a disparity d then the pixel in the coordinates $[\mathrm{x}_r - \mathrm{d}, \mathrm{y}_r]$ must have disparity $-d$. Pixels for which disparity estimates differ are rejected. This control is quite effective and is particularly good at removing mismatches due to occlusions.

```
1: procedure SHD(Ventana_1,Ventana_2)
2:     Input: Ventana_1: Ventana de Hamming de tamaño W_h
3:            Ventana_2: Ventana de Hamming de tamaño W_h
4:
5:     Output: S_o: Similitud existente entre Ventana_1 y Ventana_2
6:
7:     S_o ← 0
8:     for i ← 1 , W_h do                          ▷ Recorrido por las Columnas de las ventanas
9:         for j ← 1 , W_h do                      ▷ Recorrido por las Filas de las ventanas
10:            S_o ← S_o + ∑ Xor(Ventana_1[i,j], Ventana_2[i,j])   ▷ Calculo de la similitud
11:        end for
12:    end for
13:    return S_o
14: end procedure
```

Figure 42. Algorithm for SHD Software Implementation.

```
1: procedure LRCC(M_L2R,M_R2L,T_h)
2:     Input: M_L2R: Mapa de disparidad de la imagen izquierda como referencia
3:            M_R2L: Mapa de disparidad de la imagen derecha como referencia
4:            T_h: umbral de verificación de consistencia
5:
6:     Output: M_final: Resultado de la verificación de consistencia izquierda-derecha
7:
8:     I_W ← ancho del mapa de disparidad en pixeles
9:     I_H ← alto del mapa de disparidad en pixeles
10:
11:    for i ← 1 , I_W do                          ▷ Recorrido por las Columnas de la imagen
12:        for j ← 1 , I_H do                      ▷ Recorrido por las Filas de la imagen
13:            xl ← j
14:            xr ← xl − M_L2R[i, xl]
15:            xlp ← xr + M_R2L[i, xr]
16:            if |xl − xlp| < T_h then
17:                M_final[i, j] ← M_L2R[i, j]
18:            else
19:                M_final[i, j] ← 0
20:            end if
21:        end for
22:    end for
23:    return M_final
24: end procedure
```

Figure 43. Algorithm for LRCC Software Implementation.

Figure 43 presents the algorithm that describes the software implementation of the LRCC testing method. The procedure is performed on the entire image by going through the pixels from left to right and from top to bottom. Each pixel on the disparity map M_{L2R} is compared to the pixel d positions to the left of the disparity map M_{R2L}. If the value of the comparison is less than the selected threshold value T_h, the pixel in M_{L2R} is considered valid, otherwise

it is set to 0 indicating that it is an incorrect pixel. Annex F presents the development of stereo matching using Matlab® software.

6.2.1 Hardware architecture

According to (Dunn & Corke, 1997; Fife & Archibald, 2013; Miyajima & Maruyama, 2003) many of the high-performance systems for area-based real-time stereo vision (such as the one used in this work) employ an architecture equivalent to that in Figure 44. This allows the execution of the algorithms through temporal parallelism since it is a computational architecture of *Stream Processing*. This means that, in each clock cycle, a pixel enters and in turn a response is generated.

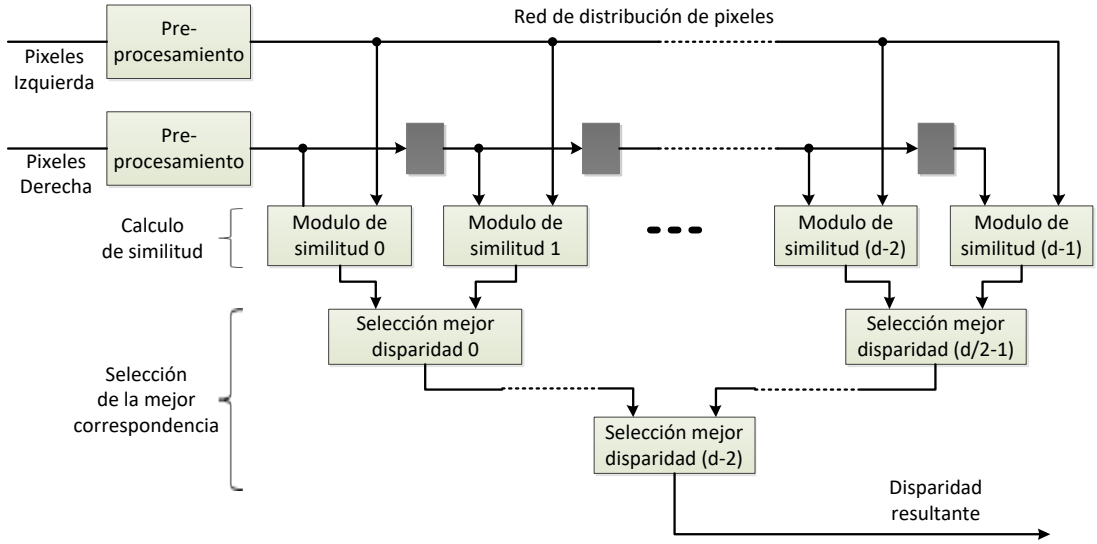


Figure 44. General stereo correlation architecture. Figure adapted by the author from (Fife, 2011; Fife & Archibald, 2013).

As can be seen in Figure 44, the images that enter the stereo correlation module are pre-processed before calculating the disparity map. This pre-processing is commonly based on filters such as LoG or Median, among others. In this work, the pre-processing method used corresponds to the sparse Census transform. Once the pre-processing method is applied to stereo images, the pixel flow of one image is delayed with respect to the other in such a way that, in each clock cycle, one pixel of the original image and each pixel of the displaced image enter the respective similarity modules. Therefore, each similarity module calculates the similarity value for a pair of pixels at a specific disparity level. That is, similarity modulus 0 calculates the similarity for all pairs of pixels with disparity 0, similarity modulus 1 calculates similarity for all pixels with disparity 1, and so on, for a total of d disparity levels. In most cases, similarity modules are based on summation over a pixel window. In this work, the similarity method used corresponds to the Hamming Sum of Distances (SHD). Finally, each "best-of-the-choice" module generates the disparity that has the best similarity score, choosing between the score of its disparity level and the best of all previous disparity levels. The output of the best-selection module $d-1$ corresponds to the output of the disparity

values for each input pixel, resulting in the disparity map (Dunn & Corke, 1997; Fife, 2011; Fife & Archibald, 2013).

Although this architecture allows parallel execution of stereo correspondence, it has a critical path through the disparity selection modules, which limits too much the maximum clock frequency at which it can function properly. The aforementioned critical path is composed $\log_2(d)$ of selection modules, which implies that for large disparity values d there is a propagation delay that can be excessive (Fife, 2011; Fife & Archibald, 2013).

In (Fife, 2011; Fife & Archibald, 2013) a modification to this architecture is proposed, so that the execution is carried out through *Pipeline*. This architecture allows d disparities to be calculated in each clock cycle, thus increasing its speed. In addition, it allows the calculation of disparity maps with the left and right images as a reference, ideal for the implementation of the LRCC algorithm (Miyajima & Maruyama, 2003). However, similarity modules must be designed in-house with *Pipeline* to maximize system performance. Figure 45 shows the stereo correlation pipeline architecture.

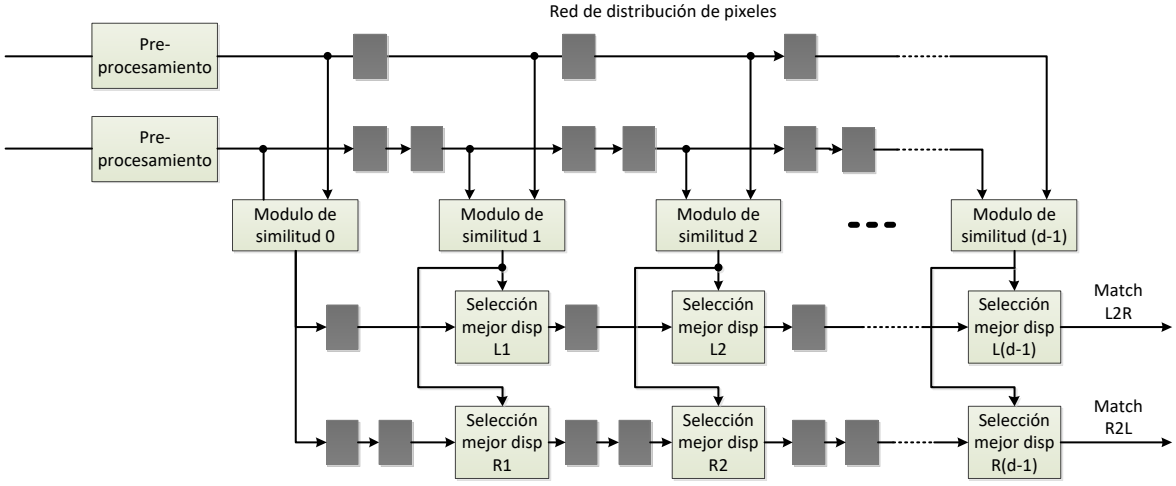


Figure 45. Stereo *correlation pipeline* architecture . Figure appended by the author from (Fife, 2011; Fife & Archibald, 2013).

Figure 46 shows the schematic diagram of the stereo matching module and the connection interface of the stereo matching module. The i_data_L and *i_data_R* signals correspond to the pixels of the left and right images respectively. This data is captured by the module on each ascending edge of the i_clk clock signal. The i_tresh_LRCC signal allows you to select the threshold for the calculation of the left-right correspondence check (LRCC). The o_data signal corresponds to the disparity map, which is generated pixel by pixel on each ascending flank of the i_clk clock signal. The control signals *i_rst_n, i_valid* and *o_valid* behave similarly to the median filter. In addition, the module has several parameters that allow you to select its behavior in the synthesis process. The parameter W_c is used to select the size of the Census window, which must be odd greater than or equal to 3. The parameter

W_h allows you to select the size of the Hamming window, which must be odd greater than or equal to 3. The parameter M sets the width of the image to be processed. The parameter D selects the number of disparity levels to be calculated, while the parameter N sets the number of bits of pixels entering the processing module.

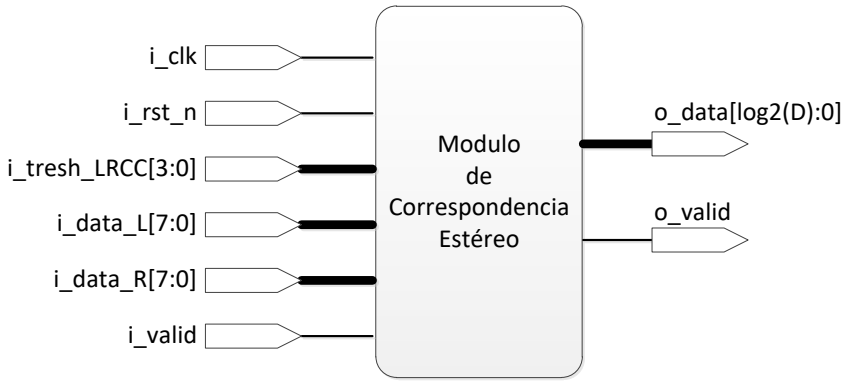


Figure 46. Schematic diagram of the stereo correspondence module. Source: author.

6.2.2 SHD Similarity Module

The similarity module calculates the Hamming Sum of Distances (SHD) over a window of size $W_h \times W_h$. According to (Dunn & Corke, 1997) the implementation of the similarity module must be carried out in two stages: in the first stage, the Hamming Distance is calculated by comparing two pixels (previously pre-processed by Census), using the XOR logic operation and counting the resulting number of 1's bits. In the second stage, the bits are added over a window of size $W_h \times W_h$. The result corresponds to the measure of similarity of the input pixels. Figure 47 presents the structure used in this work for the implementation of the SHD similarity module.

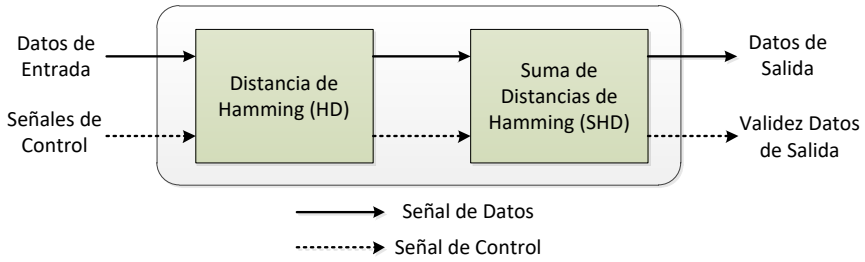


Figure 47. SHD Similarity Module. Flow of data signals and control. Source: author.

6.2.2.1 Calculating the Hamming Distance

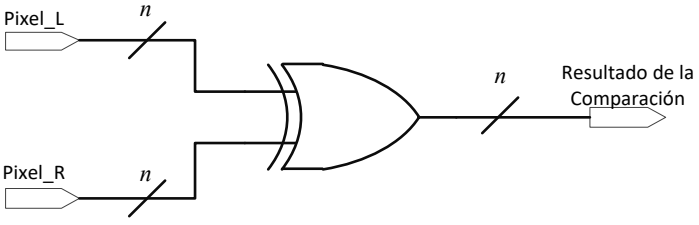
Figure 48 presents the hardware designed in this work for the calculation of the Hamming distance. The pixels that enter the similarity module, coming from the Census transformation, are compared using the XOR logic operation. Since the size of each pre-processed pixel is $n = \frac{W_c^2 - 1}{2}$ bit, logic gates are required n to perform such a comparison.

See Figure 48(a). The result of the comparison process is used to obtain the value of the Hamming distance as explained in section 3.1.3. To comply with this procedure, a pipeline configuration of tree adders was designed, as shown in Figure 48 (b). The *Pipeline* design allows for increased performance of the architecture, in which the critical path of the sum total is divided into stages. This allows you to work at higher clock speeds, only limited by the slower adder layer. The number of pipeline stages depends primarily on the number of bits obtained in the Census transformation. In this way, $\log_2(n)$ pipeline stages and $n-1$ adders are required. The maximum value of the Hamming Distance is represented in $\log_2(n)$ bits.

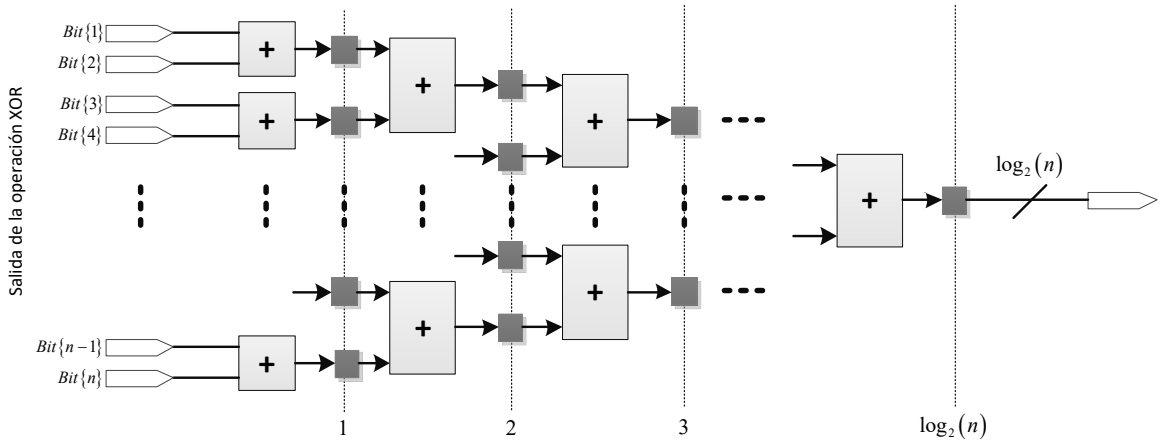
Like the designed median filter module, the hardware for calculating the Hamming distance delivers valid data after the first $\log_2(n)$ few clock cycles. To report this fact to the next processing stage, a validity-generating circuit is designed, which has the same architecture shown in Figure 39. For the case of the Hamming Distance calculation module, the comparison value used in the validity generator is $\log_2(n)$.

6.2.2.2 Sum of Hamming Distances (SHD)

After calculating the Hamming Distance of a pair of pixels entering the similarity module, the next step is to add the Hamming distances within a window. This procedure can be carried out using a movable window generator, similar to the one used in the median filter, but in this case with the information of the Hamming distances. Once the window is built, the next step is to perform the summing process using a *pipeline* configuration of tree adders. Although this implementation is functionally correct, it is extremely inefficient in the use of FPGA physical resources, mainly in terms of memory bits. Equation (19) presents the calculation necessary to know the total number of bits required by this type of implementation. In this equation corresponds d to the disparity levels to be calculated, and therefore to the number of similarity modules required, M it is the width of the image, W_h the size of the Hamming window and n the number of bits of the Census transform. This type of implementation was carried out preliminarily, which required more than 200% of the memory bits available in the device, so it was totally unfeasible for use in this work.



(a)



(b)

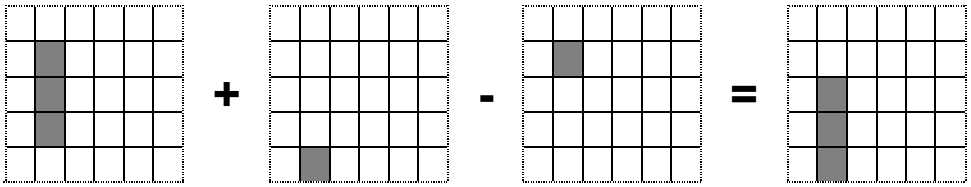
Figure 48. Calculation of the Hamming distance. (a) comparison of pixels using the XOR operation. (b) Configuration of tree adders to obtain the quantity of 1's. Source: author.

$$(W_h - 1) \times (M - W_h) \times \log_2(n) \times d \tag{19}$$

The works developed by (Dunn & Corke, 1997; Faugeras et al., 1993; Miyajima & Maruyama, 2003) present an alternative method, to the moving window generator, for the calculation of the Hamming Sum of Distances. This method employs a window sum optimization technique that involves using small buffers to calculate the sums of the top and bottom rows of the Hamming correlation window. The row sums are then used to calculate the total sum of the window. This method greatly reduces the memory usage required by the similarity module.

Although this method is very good at using memory bits within the FPGA, in (Fife, 2011; Fife & Archibald, 2013) a modification is presented that allows the required memory to be reduced by up to 30%. This modification consists of first calculating the sum of the columns, inside the window, and then using those sums to calculate the total sum of the window. Figure 49 presents this method graphically in order to illustrate its operation more clearly. This method requires, for the sum of the columns, only one memory buffer length M, where each buffer position requires a maximum of $\log_2(n \times W_h)$ bits. To calculate the total sum of the window, a buffer of W_h positions equivalent to the width of the Hamming window is required. Equation (20) presents the calculation for the maximum number of memory bits required by this type of implementation.

$$(W_h + M) \times \log_2(n \times W_h) \times d \tag{20}$$



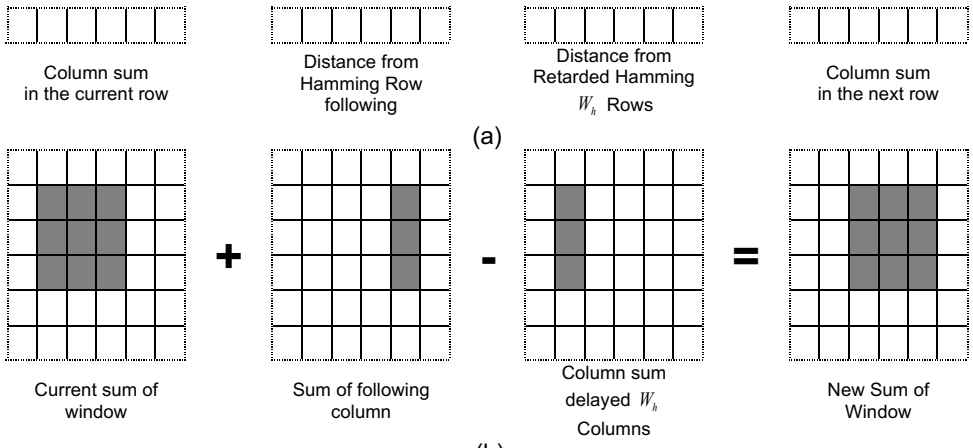


Figure 49. Calculation of the Hamming distance. (a) Procedure for adding the columns in the window. (b) procedure for calculating the total sum of the window from the sum of columns. Figure adapted by the author from (Fife, 2011)

Figure 50 shows a comparison of the memory bit usage required by the mobile window builder-based implementation and the window-sum optimization-based implementation. The figure presents the memory bits required for different Hamming window sizes W_h. The comparison was made for $d = 64$ disparity levels, Census window size, $W_c = 7$ and image width $M = 640$. As can be seen, the blue color curve, which represents the implementation of the moving window generator, increases rapidly as the Hamming window grows. This means requiring up to 3 Mbit of memory for a 17X17 Hamming window. On the other hand, the red curve, which represents the implementation of the window sum optimization method, increases slightly with a memory requirement that does not exceed 400 Kbits. Therefore, it can be concluded that window sum optimization is extremely efficient in the use of memory bits, making it ideal for use in this work.

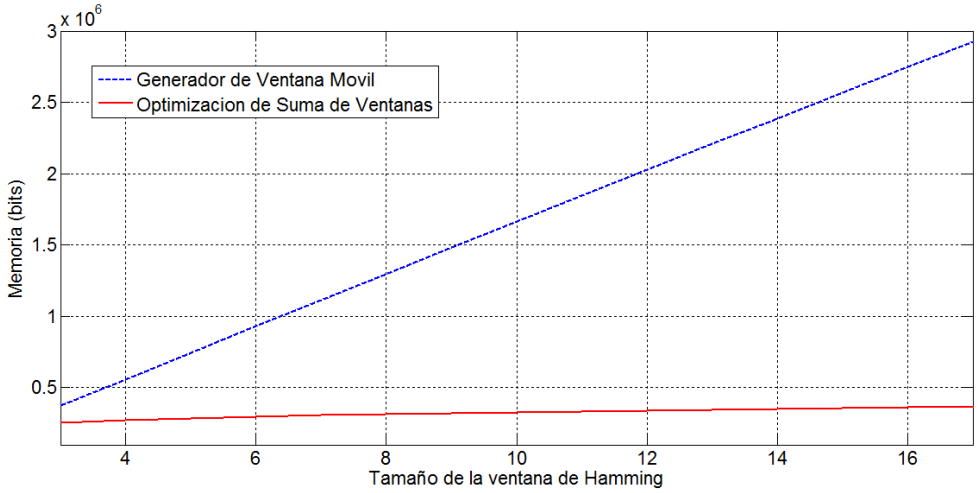


Figure 50. Memory requirements of the similarity module using moving window generator (blue) and Window Sum Optimization (red). Fountain...

In order to make the calculation more efficient, the hardware implementation of the SHD using the window sum optimization method requires obtaining the value of the Hamming distance for the upper and lower pixels of the Hamming correlation window. This implies that the Census transform delivers two pixels on the same column, but separated vertically by W_h rows. Therefore, not two, but four pixels enter the similarity module, which are compared by two Hamming Distance calculation modules, one for the lower pixel $HD(x,y)$ and one for the upper pixel $HD(x,y-W_h)$.

Figure 49 (a) performs the subtraction operation between the Hamming Distances of the lower $HD(x,y)$ and upper $HD(x,y-W_h)$ pixels of the Hamming window for the x image column. The result of this operation is added with the previous value accumulated in that same column. To obtain the total sum over the Hamming window (Figure 49 (b)) the subtraction operation is executed between the accumulated value in the column x and the accumulated value in the column $x-W_h$. The result of this operation is accumulated to obtain the value of the total Sum of Hamming distances over the window. Figure 51 presents the respective architecture developed in this work. This architecture has four pipeline stages that allow you to increase the performance of the module. The block called *Row Buffer* is a FIFO memory of M positions in which the accumulated sums in each column of the image are stored. The block called *Column buffer* is a FIFO memory of W_h positions in which the accumulated sums of the last columns W_h necessary to perform the calculation of the total sum on the window are stored.

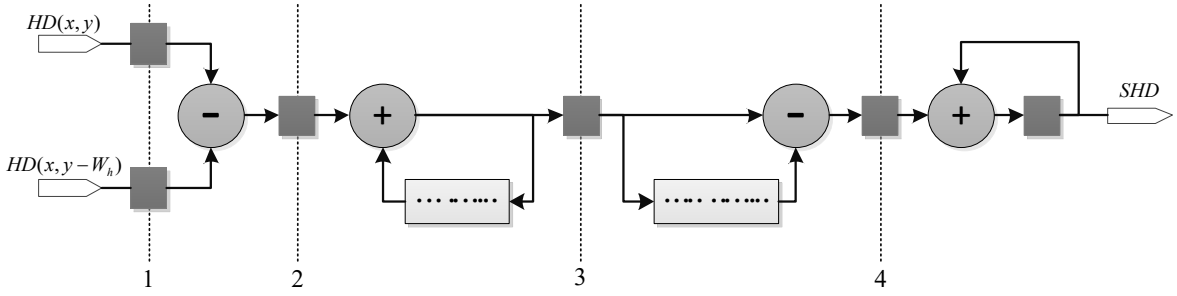


Figure 51. Calculation of the sum of Hamming distances using window sum optimization. Adaptation from (Fife & Archibald, 2013)

To obtain a valid SHD value, it is necessary to build a block that generates the validity signal of the data delivered by this module. The architecture of the validity generator is similar to that shown in Figure 39, however, the comparison value used for the SHD module is $\frac{(W_h+1)\times(\mathrm{M}+1)}{2}-M+3$.

6.2.3 Transform Census Scattered

The architecture developed for the similarity module requires that the bottom and top pixels of the Hamming correlation window ($Pixel(x,y)$ and $Pixel(x,y-W_h)$ both images) be entered into it, in order to calculate the SHD correctly. According to (Fife & Archibald, 2013) the most efficient way to obtain these pixels is to calculate them in the pre-processing module, in this case in the Census transform module. Taking into account the above, the Census transform module was developed to generate two pieces of data on the same column of the image. The module is made up of three main components: a window generator to obtain the windows required in the formation of the two output data, the Census transformation that is applied to each of the windows, and a validity generator that indicates whether the generated data is correct. The flow of input data and control signals from one component to another are shown in Figure 52.

6.2.3.1 Window Builder

To obtain a single output data from the Census transform it is required to have a pixel window as input $W_c \times W_c$. To build this hardware window, you need to cache $W_c - 1$ rows of the image. This is accomplished by designing a moving window generator like the one shown in Figure 35. However, the Census transform module must generate two pieces of data which must belong to the same column of the image but with a difference between each other of W_h rows. To achieve this, a two-window generator must be designed with such separation, which involves caching $W_c + W_h - 1$ rows of the image. Figure 53 shows the architecture of the window builder using the *row buffering cache method*. In this architecture, V_1 it corresponds to the window to calculate the Census transform of the lower pixel and V_2 corresponds to the window to calculate the Census transform of the upper pixel, required by the similarity module.

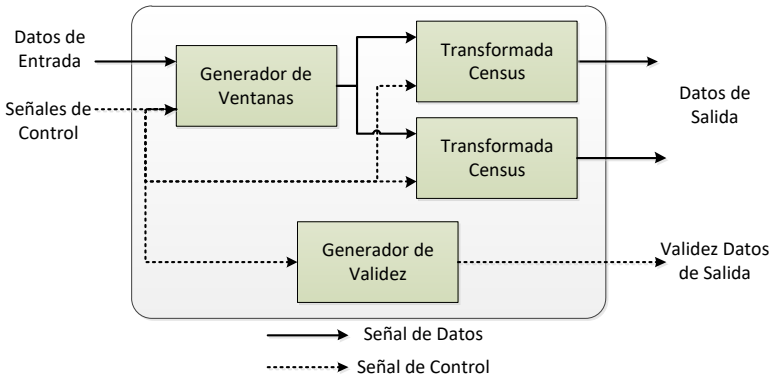


Figure 52.Census Transform Module. Flow of data and control signals. Source: author.

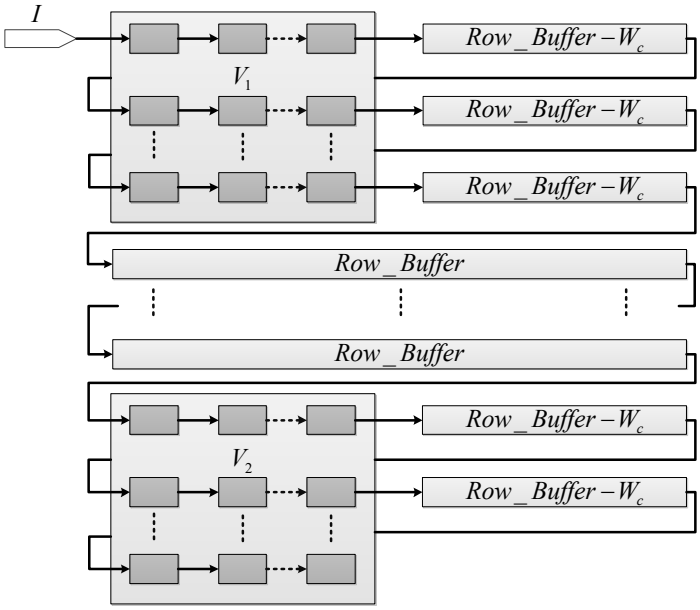


Figure 53.Census Transform Module. Flow of data and control signals. Source: author.

6.2.3.2 Census Disperse Transformation

Sparse Census transformation involves comparing the center pixel of a window with the other pixels within the window in a sparse pattern. In this work, the implementation of the Census transform was carried out using a 50% dispersion pattern as shown in Figure 7 (b). The gray boxes represent the pixels within the window that are compared to the central pixel.

A V_k 50% sparse Census window has a total of pixels $n = \frac{W_c^2 - 1}{2}$ to be compared to the central pixel C_v of that window. Figure 54 shows the architecture used to calculate the Census transform. In this architecture, n bit comparators N are used each. Comparators evaluate whether the central pixel C_v is larger than one pixel P_i (belonging to the dispersion pattern used). If the comparison is true the result is 1, otherwise the result is 0. The concatenation of the bits obtained in the comparison process results in a string of n bits corresponding to the sparse Census transformation T_c.

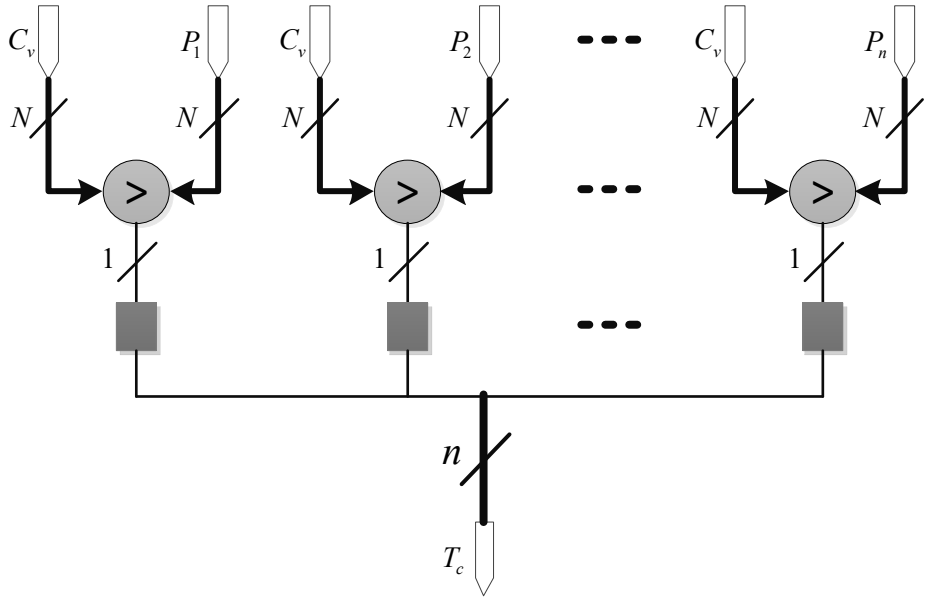


Figure 54.Calculation of the scattered census transform. Source: author.

To obtain a valid value of the sparse Census transform, a component is required to produce the validity signal with an architecture similar to the one presented in Figure 39. However, the comparison value used for the modulus of the Census transform is $\frac{(W_c+1)\times(\mathrm{M}+1)}{2}-M$

.

6.2.4 Calculation of disparity

In the architecture shown in Figure 45, the selection of the disparity value is carried out by means of an array of interconnected modules that evaluate the similarity values obtained for the d disparity levels contemplated. These modules look for the best similarity value by comparing the calculated similarities. The level of disparity whose similarity is better than the others corresponds to the disparity value of the pixel analyzed. The criterion for selecting the best similarity value used in this work is the minimum, so the arrangement of modules developed looks for the level of disparity with the lowest similarity value.

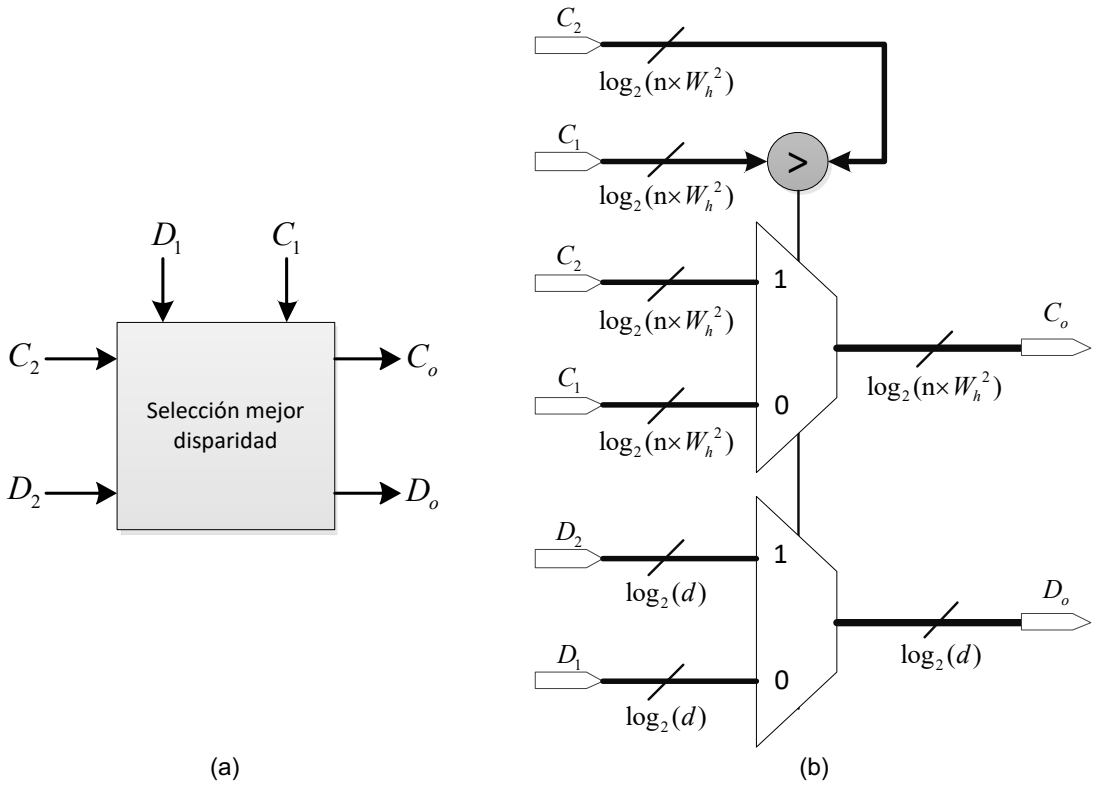


Figure 55.Module for selecting the best disparity value. (a) schematic diagram of the module. (b) internal architecture of the module. Source: author.

Figure 55 (a) presents the schematic diagram of the module for selecting the best disparity value. The and module inputs C_1 D_1 correspond to the similarity value C_1 of the disparity level D_1, and the module and inputs C_2 D_2 correspond to the similarity value C_2 of the disparity level D_2. The module performs the comparison of the similarities C_1 and C_2, and generates the outputs C_o and D_o. Figure 55 (b) presents the architecture of the selection module developed in this work. The module is composed of a magnitude comparator and two multiplexers. The comparator evaluates whether the similarity value C_1 is greater than the similarity value C_2. If the comparison is true, the output C_o takes the value C_2 and the output D_o takes the value of D_2. If the result of the comparison is false, C_o it takes the value C_1 and the output D_o takes the value of D_1. The number of bits required by the module is determined by the disparity levels d, the size of the Census window, W_c and the size of the Hamming sale W_h.

6.2.5 Left-Right Consistency Check (LRCC)

Left-right consistency checking (LRCC) is a widely used method for disparity map refinement. This method requires the prior calculation of the disparity maps taking as reference the left (L2R) and right (R2L) images of the stereo pair. The hardware architecture shown in Figure 45 allows the calculation of the two disparity maps required by this method. However, there is a pixel lag of d the R2L disparity map with respect to the L2R disparity map. This lag is corrected, before calculating the LRCC, by implementing d cascading records with the L2R disparity map.

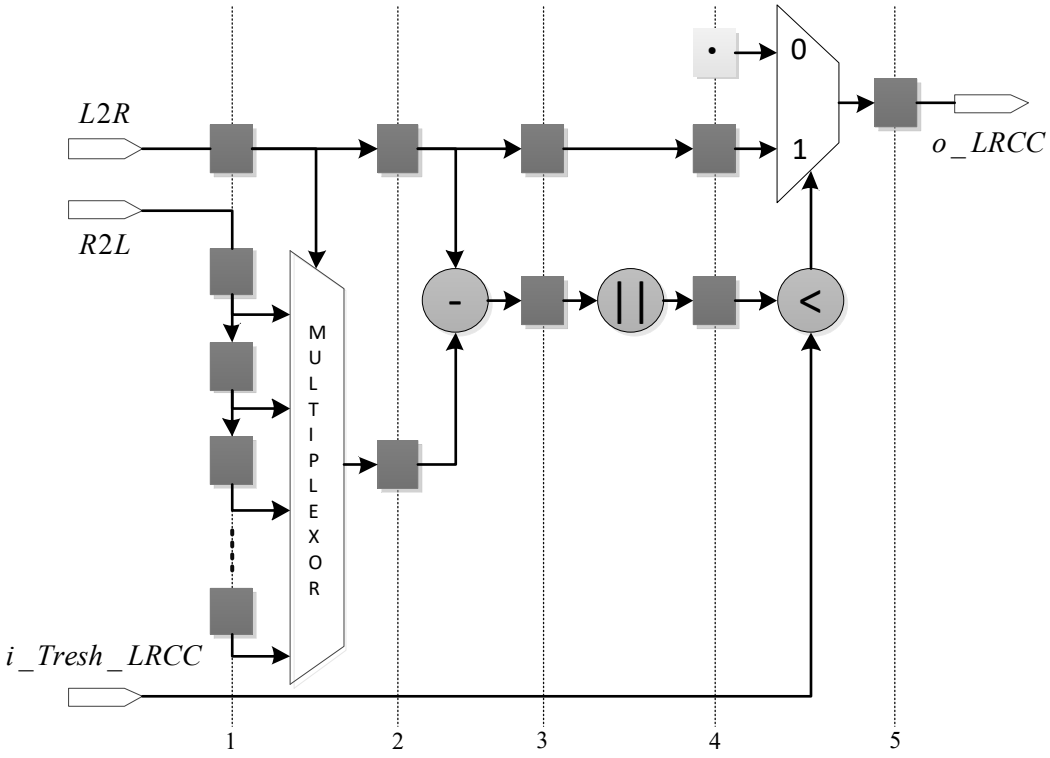


Figure 56.Calculation of the LRCC left-right consistency check. Source: author.

The FPGA design of the LRCC module for the refinement of the disparity map is carried out taking into account the definition presented in equation (21). According to this equation, the absolute value of the difference between the disparity for one pixel on the L2R map and the disparity of one pixel on the R2L map as a function of the pixel's disparity on the L2R map must be less than a threshold value T_h. Ideally, T_h it should be equal to 1, indicating that the disparity of one pixel on the L2R map and its equivalent on the R2L map are equal. If the inequality is true, it means that the pixels compared in both disparity maps are consistent. Otherwise, the disparity value for the compared pixel is considered not to correspond to an allowable disparity value.

$$\left|d_{L2R}(x) - d_{R2L}(d_{L2R}(x))\right| < T_h \tag{21}$$

Figure 56 shows the hardware developed for the calculation of the LRCC. The pixels of the R2L map enter an array of registers connected in cascade. These registers together with a multiplexer allow you to select the disparity value of the R2L map based on the disparity of one pixel of the L2R map. Since the maximum calculated disparity value is determined by d disparity levels, records are required d for this purpose in total. The hardware module calculates the absolute value of the difference between the disparity value at the output of the multiplexer and the disparity value of the L2R map and compares the result of the operation with the selected threshold value. In case of a favorable result, the disparity value of the L2R map is sent abroad, otherwise a 0 is sent instead indicating that this value is not consistent in both disparity maps.

The hardware module was designed to execute disparity map refinement using 5 pipeline stages in order to maintain a robust design for the entire stereo matching module. Due to the *Pipeline*, the disparity map has an initial delay of five valid pixels, so it is necessary to implement a component for the generation of the output validity signal. This validity generator has the architecture shown in Figure 39 and uses a comparison value of 4 for the particular case of the LRCC module.

6.3 Segmentation

Segmentation is an image processing method used in extracting regions of interest. The segmentation process labels the pixels of the region that you want to extract from the background of the image. In this work, two segmentation methods are used to extract the region of the shape of the hand: segmentation by skin color and segmentation of the disparity map.

Skin color segmentation allows you to assign a value of 1 to pixels that have a color similar to that of human skin, and 0 for the other pixels present in the scene (see Appendix G). Disparity map segmentation allows extracting regions or objects that are in a range of disparity levels. Pixels corresponding to regions extracted from the disparity map are tagged with a 1, and ignored pixels are tagged with 0. The selection of the range of disparity levels is made in this case taking into account the depth zone where the movement of the hand is executed during the performance of a gesture. The combination of these two segmentation methods allows the region of the shape of the hand to be extracted more effectively, which will be analyzed later to characterize the gesture performed. The segmentation process was initially developed in software using MATLAB.®

Figure 57 presents the algorithm that describes the software implementation of skin color segmentation for images in the YCbCr color model. The segmentation process is carried out at the pixel level by traversing the image from top to bottom and from left to right. For each pixel, the chrominance components are evaluated according to the selected threshold values. See equation (22). If the chrominance values are within the set ranges, the pixel is labeled with a value of 1, otherwise it is labeled with a value of 0.

```
1: procedure SKIN SEGMENTATION(IM_YCbCr)
2:     Input: IM_YCbCr: Imagen en el modelo de color YCbCr
3:
4:     Output: IM_segm: Segmentacion por color de piel
5:
6:     I_W ← ancho de la imagen en pixeles
7:     I_H ← alto de la imagen en pixeles
8:
9:     for i ← 1 , I_W do                    ▷ Recorrido por las Columnas de la imagen
10:        for j ← 1 , I_H do                 ▷ Recorrido por las Filas de la imagen
11:            if 77 < C_b[i, j] < 127 && 133 < C_r[i, j] < 173 then
12:                IM_segm[i, j] ← 1
13:            else
14:                IM_segm[i, j] ← 0
15:            end if
16:        end for
17:    end for
18:    return IM_segm
19: end procedure
```

Figure 57.Algorithm for the implementation of skin color segmentation in software. Fountain...

$$P_{skin} = \begin{cases} 1 & \text{if } 77 \le C_b \le 127 \ \ \& \ 133 \le C_r \le 173 \\ 0 & \textit{Other Case} \end{cases} \tag{22}$$

Figure 58 presents the algorithm that describes the implementation of the disparity map segmentation in software. The segmentation process is carried out at the pixel level, going through the image from top to bottom and from left to right. For each pixel of the disparity map, the disparity is evaluated according to the selected thresholds $T_L < d[i,j] < T_H$. These thresholds are selected experimentally taking into account the space required by the person when performing a gesture. If the disparity value $d[i,j]$ is within the limits, the pixel is labeled with a 1, otherwise it is labeled with 0.

```
1: procedure DISPARITY SEGMENTATION(MD)
2:     Input: MD: mapa de disparidad con d niveles de disparidad
3:
4:     Output: MD_segm: Segmentación del mapa de disparidad resultante
5:
6:     I_W ← ancho de la imagen en pixeles
7:     I_H ← alto de la imagen en pixeles
8:     T_H ← umbral superior de comparación
9:     I_L ← umbral inferior de comparación
10:
11:    for i ← 1 , I_W do                    ▷ Recorrido por las Columnas de la imagen
12:        for j ← 1 , I_H do                ▷ Recorrido por las Filas de la imagen
13:            if T_L < d[i, j] < T_L then
14:                MD_segm[i, j] ← 1
15:            else
16:                MD_segm[i, j] ← 0
17:            end if
18:        end for
19:    end for
20:    return MD_segm
21: end procedure
```

Figure 58.Algorithm for the implementation of disparity map segmentation in software.

Finally, the two resulting segmentations are intersected, obtaining the extraction of the region that corresponds to the shape of the hand of the gesture made. Annex F presents the development of the segmentation process using Matlab® software.

6.3.1 Hardware architecture

The design of the hardware segmentation process is carried out using a computational architecture to *Stream Processing*. The segmentation hardware module is composed of three main components: skin color segmentation, disparity map segmentation, and the intersection of segmentations. The flow of input data and control signals from one functional block to another is shown in Figure 59.

Figure 60 shows the schematic diagram of the segmentation module designed. This diagram shows the module's connection interface. The i_Cb and *i_Cr* signals correspond to the chrominance components of the input image in the YCbCr color model. The i_L2R signal corresponds to the input of the pixels of the disparity map. This data is captured by the module on each ascending edge of the clock signal *i_clk*. The o_data signal corresponds to the segmentation result and is generated pixel by pixel on each ascending flank of the i_clk clock signal. The control signals *i_rst_n, i_valid,* and *o_valid* have similar behavior to the median and stereo correspondence filter modules. In addition, the module has several parameters that allow you to select its behavior in the synthesis process. The parameter D selects the number of disparity levels available on the disparity map. The parameter N allows you to select the number of bits in which the chrominance components are represented.

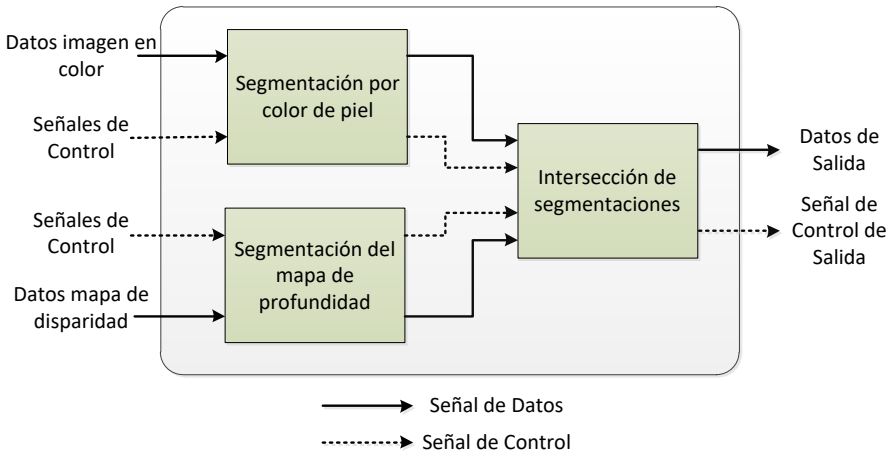


Figure 59.Segmentation Module. Flow of data signals and control. Source: author.

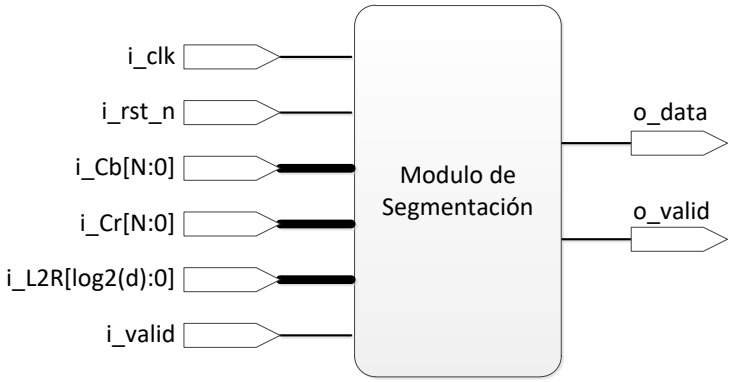


Figure 60.Schematic diagram of the segmentation module in hardware. Source: author.

Below is a detailed description of the internal components that make up the architecture of the segmentation module.

6.3.2 Skin color segmentation

The hardware design of the segmentation by skin color was carried out using the architecture shown in Figure 61. In this equation, the comparison of each color component with the threshold values established in equation (22) is observed. For this module, 4 bit comparators each are required N . Each comparator results in a 1 if the comparison is true and a 0 if the comparison result is false. If the chrominance values of an input pixel are within the set comparison values, the module generates a 1 in the output, otherwise it generates a 0. When the output is equal to 1, it indicates that the pixel belongs to a skin-colored region.

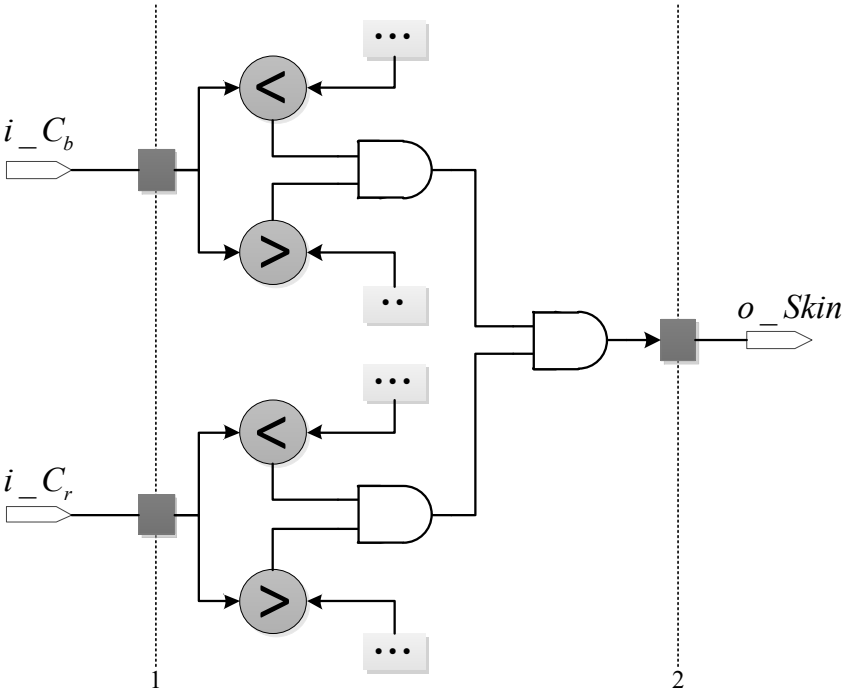


Figure 61.Calculation of segmentation by skin color. Source: author.

Since the module designed for the calculation of skin color segmentation has inputs and outputs, two valid clock cycles are required to obtain a valid segmentation data at output. For this reason, an additional component is required to indicate whether the generated pixel is valid or not. This component is designed based on the architecture shown in Figure 39.

6.3.3 Disparity map segmentation

The hardware design of the disparity map segmentation was carried out using the architecture shown in Figure 62. This figure shows the comparison of the input disparity value taking into account the threshold T_L and T_H selected values. This module has a similar architecture to that used in skin color segmentation. In this case, only 2 bit comparators each are required N . Each comparator results in a 1 if the comparison is true, and a 0 if the comparison result is false. If the disparity value of an input pixel is within the set comparison values, the module generates a 1 on the output, otherwise it generates a 0. When the output is equal to 1, it indicates that the pixel belongs to a spatial area defined for the execution of the gesture. The selection of threshold values was carried out experimentally, obtaining the best results for $d = 48$ disparity levels and $T_L = 33$ and $T_H = d - 2$.

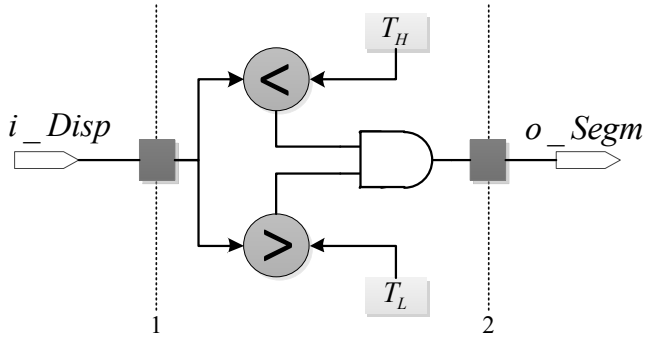


Figure 62.Calculation of the segmentation of the disparity map. Source: author.

Since the module designed for the calculation of the segmentation of the disparity map has registers at the input and output, two valid clock cycles are needed to obtain a valid segmentation data at the output. Therefore, an additional component is required to indicate whether the generated pixel is valid or not. This component is similar to that required by the skin color segmentation module.

Finally, the segmentations are intersected by means of an AND logic operation to obtain the final result of the segmentation taking into account that they are binary images.

The intersection process requires that the skin color segmentation and the disparity map segmentation be fully synchronized. In this case, the disparity map segmentation is out of date with respect to the skin color segmentation due to the latency in the stereo matching calculation module. To synchronize the two segmentations, it is enough to delay the segmentation by skin color by the number of pixels of latency of the disparity map as shown in Figure 30. The segmented image is a binary image, which can contain unwanted pixels that can be removed by morphological filtering.

6.4 Morphological Opening Operation

Aperture morphological operation is a filter widely used in binary images, especially segmented images. This morphological operation allows you to eliminate unwanted pixels obtained as a result of a segmentation process. In this work, this type of filtering is used in order to eliminate isolated pixels that do not belong to the shape of the hand extracted in the segmentation process explained in the previous section.

The morphological opening operation consists of consecutively applying a morphological erosion operation followed by a morphological expansion operation using the same structuring element (see Appendix G). The morphological opening operation was initially developed in software using MATLAB.® This implementation allows us to understand the structure and functioning of the morphological operations Erosion and Expansion, for the subsequent development of its FPGA architecture. In addition, the implementation in software is considered as a reference model for the verification of the results obtained in hardware.

Figure 63 presents the algorithm that describes the implementation in software of the morphological operation of erosion on a binary image. The erosion process is carried out by going through the pixels of the image from top to bottom and from left to right. Elements within a pixel-centered window V with coordinates $[i, j]$ are intersected with a structuring element EE. If the result of the intersection is again the structuring element, the pixel is set to 1 and remains in the image as part of the object. If, on the other hand, it is different, then the pixel is set to 0, eroding the region of the object to which the pixel belongs.

```
1:  procedure EROSION(I_bin, W)
2:      Input: I_bin: imagen binaria
3:
4:      Output: O_bin: imagen erosionada
5:
6:      I_W ← ancho de la imagen en pixeles
7:      I_H ← alto de la imagen en pixeles
8:      EE[W, W] ← Elemento estructurante de tamaño W × W
9:
10:     for i ← W²/2 , I_W − W²/2 do          ▷ Recorrido por las Columnas de la imagen
11:         for j ← W²/2 , I_H − W²/2 do      ▷ Recorrido por las Filas de la imagen
12:             if Ventana[i, j] ∧ EE = EE then
13:                 O_bin[i, j] ← 1
14:             else
15:                 O_bin[i, j] ← 0
16:             end if
17:         end for
18:     end for
19:     return O_bin
20: end procedure
```

Figure 63.Algorithm for the implementation of the morphological operation of Erosion in software.

Figure 64 presents the algorithm that describes the implementation in software of the morphological operation of dilation on a binary image. The dilation process is carried out by going through the pixels of the image from top to bottom and from left to right. Elements within a pixel-centered window V with coordinates $[i, j]$ are compared to a structuring element EE. If there is an element of the window, which belongs to the region of the object and which matches at least part of the pattern of the structuring element $\neg\text{Ventana}[i, j] \wedge \text{EE} \neq \text{EE}$, the central pixel of the window is set to 1 by dilating the region of the object with that pixel. If, on the other hand, there is no match, then the region of the object is not dilated with that pixel.

```
1: procedure DILATACION(I_bin,W)
2:     Input: I_bin: imagen binaria
3:
4:     Output: O_bin: imagen dilatada
5:
6:     I_W ← ancho de la imagen en pixeles
7:     I_H ← alto de la imagen en pixeles
8:     EE[W,W] ← Elemento estructurante de tamaño W × W
9:
10:    for i ← W²/2 , I_W − W²/2 do          ▷ Recorrido por las Columnas de la imagen
11:        for j ← W²/2 , I_H − W²/2 do      ▷ Recorrido por las Filas de la imagen
12:            if ¬Ventana[i,j] ∧ EE ≠ EE then
13:                O_bin[i,j] ← 1
14:            else
15:                O_bin[i,j] ← 0
16:            end if
17:        end for
18:    end for
19:    return O_bin
20: end procedure
```

Figure 64.Algorithm for the implementation in software of the morphological operation of Dilation.

To perform the morphological aperture operation, it is enough to erode the image first and then apply the dilation operation. Annex F presents the development of the morphological operations erosion and expansion using the MATLAB® software.

6.4.1 Hardware architecture

The design of the aperture morphological filter in hardware is carried out using a Stream Processing computational architecture. The hardware module of the aperture morphology filter is composed of two main components. The morphological erosion filter and the morphological expansion filter. The flow of input data and control signals from one functional block to another is shown in Figure 65.

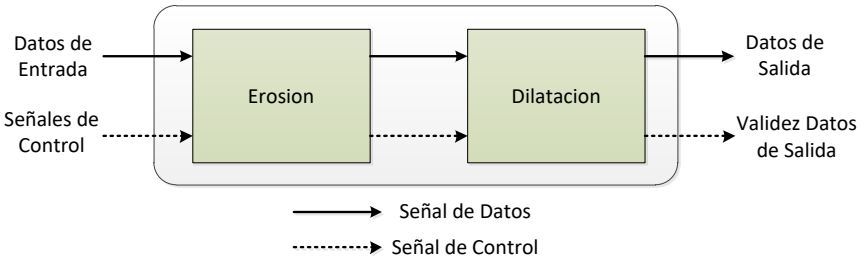


Figure 65. Module of the morphological operation of opening. Flow of data signals and control. Source: author.

Figure 66 presents the schematic diagram of the module of the morphological aperture filter developed. This diagram shows the module's connection interface. The i_dato signal corresponds to the binary image obtained in the segmentation module. The data is captured

by the module on each ascending edge of the clock signal *i_clk*. The o_dato signal corresponds to the result of the morphological aperture filter, which is generated pixel by pixel on each ascending flank of the i_clk clock signal. The control signals *i_rst_n, i_valid,* and *o_valid* behave similarly to median, stereo matching, and segmentation filter modules. In addition, the module has several parameters that allow you to select its behavior in the synthesis process. The parameter W is used to select the window size required by the filter. The parameter M sets the width of the image to be processed.

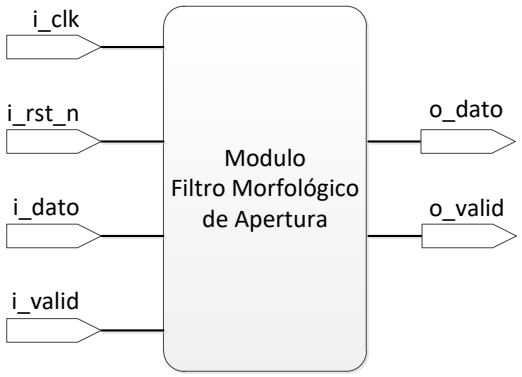


Figure 66.Schematic diagram of the module of the morphological operation of aperture in hardware. Source: author.

6.4.2 Erosion and expansion

The hardware design of the morphological erosion and expansion filters is carried out using three components: mobile window generator, morphological filter and validity generator. Figure 67 presents a general diagram of the design of the morphological filters used in this work. According to (Bailey, 2011, p. 264) erosion and dilation are dual morphological operations, since the dilation of the image is equivalent to the erosion of the background and vice versa. Taking this characteristic into account, it is possible to design a hardware module that allows the implementation of either of the two morphological operations.

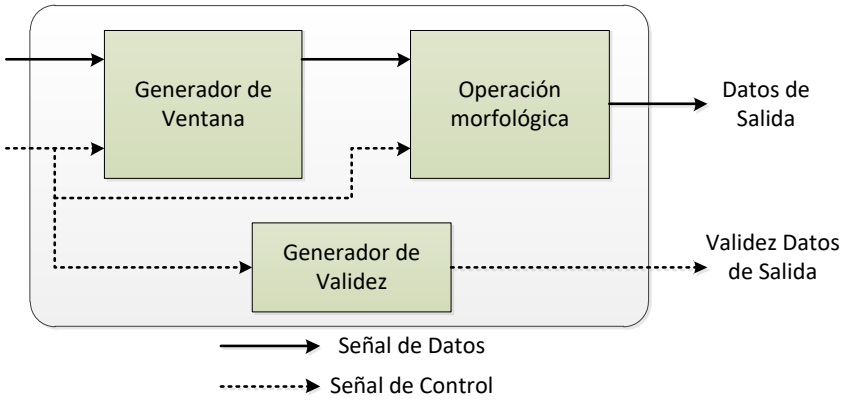


Figure 67.General diagram of the hardware implementation of a morphological filter. Source: author.

Since morphological filters are local filters, a pixel window is required to perform the respective processing. For this purpose, a mobile window generator is designed that caches the last $W-1$ windows, without the need to have the complete image. In this way, all the pixels of the window can be accessed in parallel. The window generator is designed based on the architecture shown in Figure 35. In order to design a module with the ability to apply the erosion or expansion filter to an image, the window architecture is modified by adding an XOR gate between the pixels entering the generator and an operation selection line. Figure 68 shows the structure of the window generator designed for morphological filtration.

The design of the morphological filter is carried out taking into account two important elements. A window V and a structuring element EE. The structuring element contains the pattern used by the filter to perform the required operation. Both the structuring element and the window must be the same size, which is determined by the synthesis parameter W. The number of elements in the window is $k=W^2$, being the same number of elements in the structuring element.

Figure 69 shows the design made for the morphological filter. Each element is V masked by an element of EE using the logical function $f_i=\overline{EE_i}+V_i$ for $1\le i\le k$. This allows you to take into account in the filtering process only the pixels in the window for which $EE_i=1$. After masking the pixels, the logical operation AND is applied to the elements f_i obtained in the masking process, resulting in a morphological erosion operation.

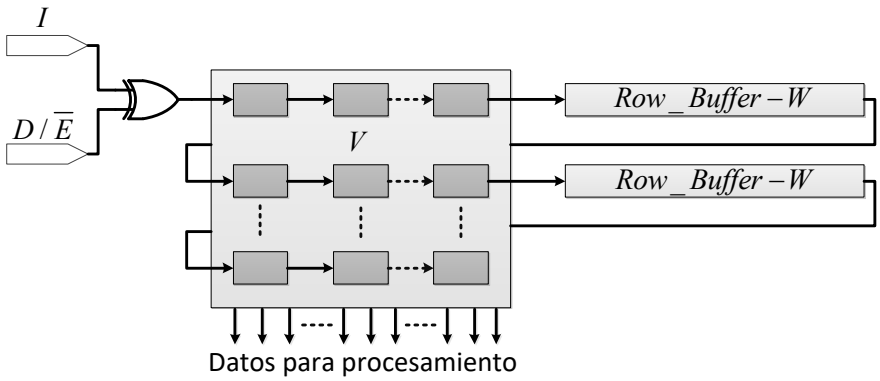


Figure 68.Architecture of the window generator for morphological filtering. The signal $D/\overline{E}$ allows the selection of the morphological operation to be used. Figure adapted by the author from (Bailey, 2011).

The morphological operation of dilation is obtained by complementing the input pixels and the result of the AND operations. This complement causes the pixels belonging to the background of the image to take a value of 1 within the window generator, so that when they are eroded, what is really obtained is a dilation of the object. The AND logical operation is

performed using a pipelined tree structure. The number of pipeline states is $\log_2(k)$+2, as the tree structure increases or decreases depending on the size of the window used. The selection of the operation mode of the processing module is done using the input $D/\overline{E}$. If the input is 0, the module functions as a morphological erosion filter, if the input is 1 the module functions as an expansion filter.

Since the module designed for the calculation of the segmentation of the disparity map has a moving window generator and a pipeline processing structure*, valid clock cycles* are needed $\frac{(W+1)\times(M+1)}{2} - M + \log_2(k) + 2$ to obtain a valid data at the output. For this reason, an additional component is required to indicate whether the pixel is valid or not. This is designed based on the architecture shown in Figure 39. This validity generator uses a comparison value for $\frac{(W+1)\times(M+1)}{2} - M + \log_2(k) + 1$ implementation in the morphological filtering module.

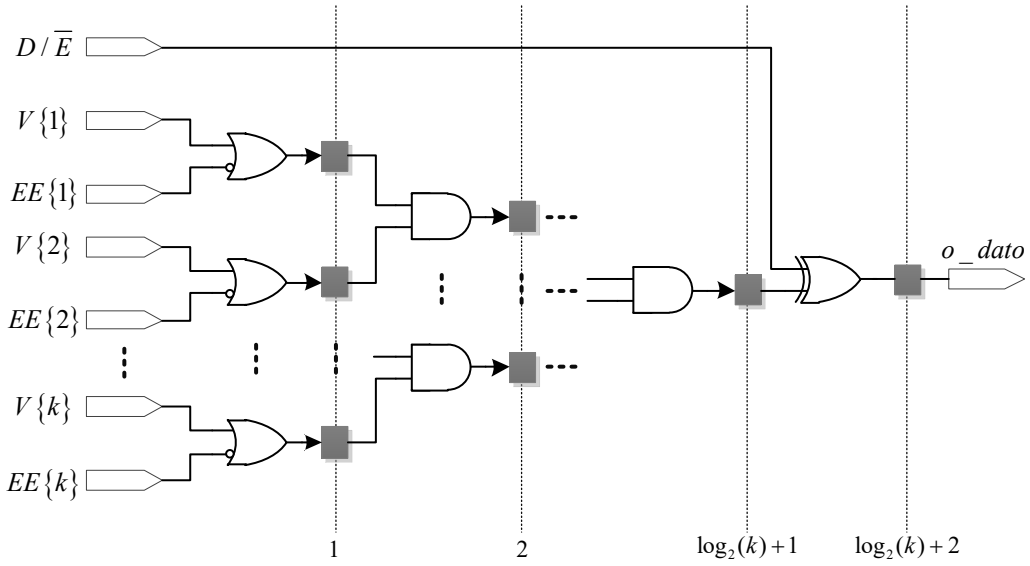


Figure 69.Architecture for basic morphological operations Erosion and expansion. The signal $D/\overline{E}$ allows the selection of the morphological operation to be used. Source: author.

The segmentation, stereo correspondence, median filter and aperture morphological filter modules were parametrically described in VHDL language. Annex D describes each of these modules and presents the simulation results obtained using the Modelsim-Altera software.

7 Feature extraction

Feature extraction is the third stage of a vision-based gesture recognition system. In this stage, the pre-processed images are taken in order to extract the necessary information that describes unique characteristics for a certain gesture. These characteristics are used to distinguish one gesture from another at a later stage of recognition.

This chapter presents the process used in this work for the extraction of characteristics of a gesture from the images obtained in the pre-processing stage. This process is developed by executing four consecutive phases: calculation of the hand centroid for the extraction of the gesture path, moving average filter for smoothing the gesture path, detection of valid gesture using start-end conditions and extraction of characteristics based on the orientation of the gesture path. Below are the algorithms used in each of the phases of the process, as well as their implementation in FPGAs.

7.1 Calculating the Centroid of the Hand Shape

Any dynamic gesture is constructed from the trajectory of the movement of the hand, also known as the trajectory of the gesture. In most situations, it is possible to differentiate one gesture from another simply by observing the characteristic trajectory of each one. In a vision-based recognition system, the trajectory of the gesture is constructed by joining the centroid values of the hand shape obtained from a sequence of images (see Figure 70). In this work the centroid of the shape of the hand is obtained from the image produced in the pre-processing stage. This image is a binary image that contains as its only object the region of the shape of the hand captured at an instant in time.

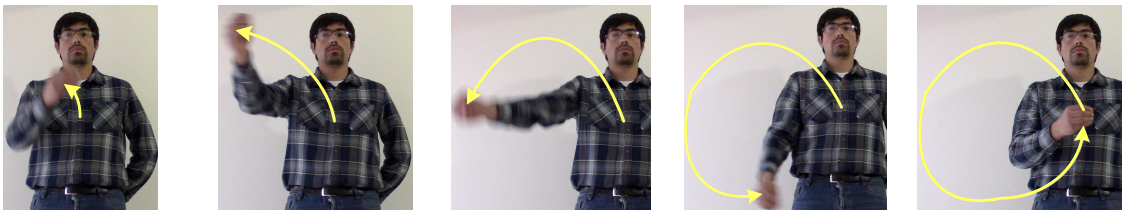


Figure 70.Centroid path of the hand for the "Rotate Right 90°" gesture. Source: author.

There are several methods that allow the calculation of the centroid of a region contained in a digital image. A widely used method for this purpose is the use of first-order two-dimensional geometric moments. Equation (23) presents the definition of the two-dimensional geometric moments of order $p+q$ for a row-by-column digital image $M \quad N$ (Binh et al., 2005). The function $f(x,y)$ corresponds to the region or object of analysis within the image. This function in this work corresponds to the region of the shape of the hand. The momentum M_{00} describes the area in $f(x,y)$ pixels, and the relationship to the momentum,

and M_{10} M_{01} allows the centroid (x_c, y_c) of the region $f(x,y)$ to be calculated as shown in equation (24).

$$M_{pq} = \sum_{x=1}^{N} \sum_{y=1}^{M} x^p y^q f(x,y) \tag{23}$$

$$x_c = \frac{M_{10}}{M_{00}} \qquad\qquad y_c = \frac{M_{01}}{M_{00}} \tag{24}$$

Figure 71 presents the algorithm that describes the procedure for calculating the centroid of the shape of the hand by using the first-order two-dimensional geometric moments. The centroid is calculated from the binary image obtained in the pre-processing stage. The image is traversed pixel by pixel from top to bottom and from left to right. For each pixel in the image, the moments M_{00}, M_{10} and M_{01}.

```
1:  procedure CENTROIDE(I_bin)
2:      Input: I_bin: Imagen en binaria resultado del proceso de segmentación
3:                    y filtrado en la etapa de pre-procesamiento
4:
5:      Output: x_c: Coordenada x del centroide
6:              y_c: Coordenada y del centroide
7:
8:      I_W ← ancho de la imagen en pixeles
9:      I_H ← alto de la imagen en pixeles
10:     M_00 ← 0
11:     M_10 ← 0
12:     M_01 ← 0
13:
14:     for i ← 1 , I_W do                    ▷ Recorrido por las Columnas de la imagen
15:         for j ← 1 , I_H do                ▷ Recorrido por las Filas de la imagen
16:             M_00 ← M_00 + I_bin[i, j]
17:             M_10 ← M_10 + i × I_bin[i, j]
18:             M_01 ← M_01 + j × I_bin[i, j]
19:         end for
20:     end for
21:     x_c ← M_10 / M_00
22:     y_c ← M_01 / M_00
23:     return x_c, y_c
24: end procedure
```

Figure 71.Algorithm for calculating the centroid of the shape of the hand in a binary image.

You can see that the algorithm input is a size image $I_W \times I_H$. However, the result is only two pieces of data x_c and y_c they correspond to the coordinates of the calculated centroid. This implies that the information of a complete image is encoded only in two pieces of data that

correspond to the position of the hand at an instant in time t during the performance of a gesture. Consequently, the trajectory of the gesture, observed in consecutive images, is represented by a set of points as shown in equation (25), where (x_t, y_t) it refers to the point of the centroid of the hand at time *t* and *T* is the length of the trajectory of the hand gesture.

$$Trayectoria\_Gesto = \{(x_1, y_1), (x_2, y_2), ..., (x_t, y_t), ..., (x_T, y_T)\} \quad (25)$$

The hardware implementation for the calculation of the centroid was developed based on the algorithm presented in Figure 71. The design made for the hardware module is composed of a coordinate generator that allows to know the position of a pixel within the image, three adders-accumulators used to calculate the two-dimensional geometric moments of the first order, and two divisors for the calculation of the *x* and *y* coordinates of the centroid. Figure 72 presents the architecture developed.

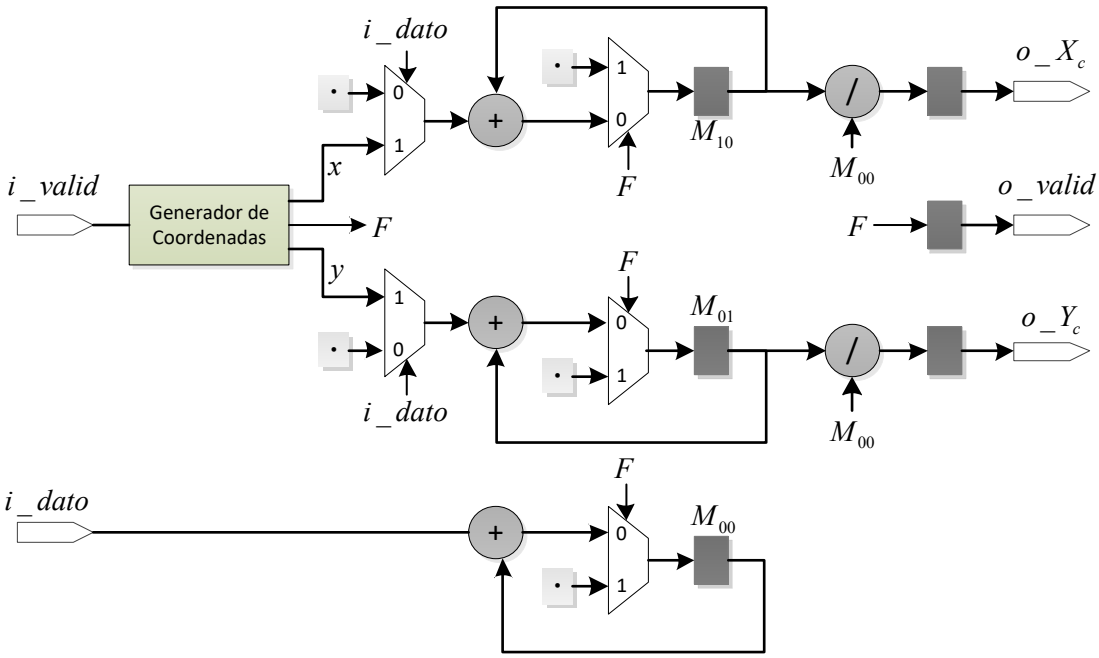


Figure 72. Architecture designed for the calculation of the centroid. Source: author.

In this architecture the coordinate generator produces three output signals: the *x* and *y* coordinates, and an F output that is set to 1 during a clock cycle, indicating that a complete image has been processed. Each time an image has been processed, the *x*, e *and y* coordinates are set to 0 again in order to wait for a new image to enter the module to be processed. The adders-accumulators for the calculation of geometric moments are controlled by the binary value of the input pixel and by the signal F of the coordinate generator. The pixel value enables (object region pixel) or disables (background pixel) accumulators depending on their value. The F signal resets the accumulators in order to

prepare the module to process a new image. Likewise, the F signal allows the passage of the calculated value of the centroid and informs the outside of said event.

Figure 73 presents the schematic diagram of the module for the calculation of the centroid of the hand. This diagram shows the interface between the module and the outside world. The *i_dato* signal corresponds to the value of the pixels coming from the morphological opening operation and the *i_valid* signal indicates the validity of these pixels. On each ascending edge of the clock signal *i_clk* a valid pixel of the binary image is captured by the module. The *o_Xc* and *o_Yc signals* correspond to the calculated value of the centroid. These signals are generated by the module when a complete image has been analyzed. The o_valid signal indicates when a new valid centroid value has been generated. The i_rst_n signal has a similar behavior to the hardware modules already explained in the previous capiture. Additionally, the module has two parameters that allow you to select its behavior in the synthesis process. The parameters M and N allow you to set the width and height of the image to be processed.

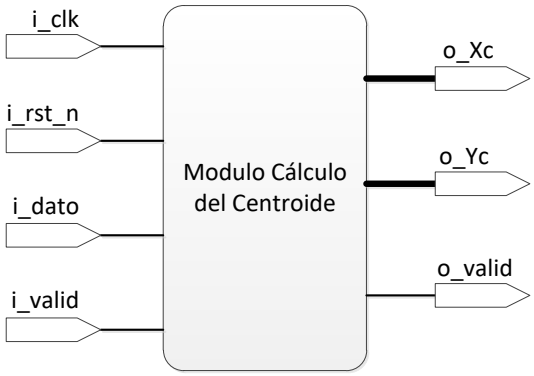


Figure 73.Schematic diagram of the module for calculating the centroid of the hand in hardware. Source: author.

7.2 Gesture Anti-Aliasing: Moving Average Filter

The trajectory of the hand gesture is determined by connecting the dots of the centroid as described in the previous section. Input images are often unstable due to the change in lighting conditions between images, messy backgrounds, and shaking during movement, which will cause frequent and sharp changes in the centroid. In order to effectively overcome these unexpected changes, the trajectory points are smoothed out using the moving average filter, which is a special case of the regular FIR filter. This filter outputs the average value of the last N samples in the input. Equation (26) presents the equation that describes this type of filter.

$$y[n] = \frac{1}{N}\sum_{k=0}^{N-1} x[n-k] \tag{26}$$

Two moving average filters are used for gesture smoothing, one for the centroid coordinate x and the other for the coordinate y. Figure 74 shows an application example for smoothing the "Rotate Right 90°" gesture. Figure 74 (a) shows the trajectory of the gesture in which the presence of noise is evident. Figure 74 (b) shows the result of smoothing the gesture path by applying the two moving average filters of length 4.

(a) (b)

Figure 74.Smoothing the "Rotate Right 90°" gesture using the Moving Average Filter of Length 4. Author source

The hardware implementation of the moving average filter is carried out using the recurring form shown in equation (27). This recurrent form allows the implementation of this filter with less arithmetic hardware compared to the normal form of the equation (26). Figure 75 shows the architecture developed for the hardware calculation of the moving average filter, which was designed to perform operations in fixed-point arithmetic, with 10 bits for the integer part and 8 bits for the fractional part.

$$y[n] = y[n-1] + \left(\frac{x[n]}{N} - \frac{x[n-N]}{N} \right) \tag{27}$$

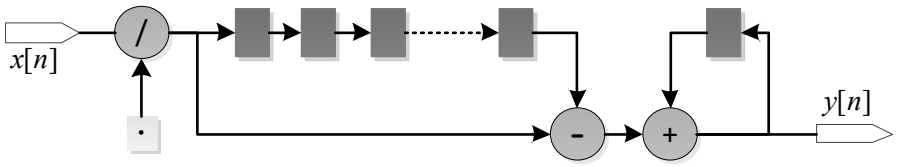


Figure 75.Hardware architecture for the calculation of the N-length moving average filter. Source: author.

Figure 76 shows the schematic diagram of the hardware module designed for gesture smoothing showing the module's interface connecting to the outside. The i_Xc and *i_Yc* signals correspond to the previously calculated centroid of the hand shape. The centroid data is captured on the ascending edges of the i_clk clock signal, as long as the i_valid signal is high. The o_Xc and *o_Yc signals* correspond to the output of the centroid resulting from the smoothing of the gesture. The o_valid signal indicates that a new centroid value has been processed. The i_rst_n signal has the same behavior as in the modules explained in previous sections. The hardware module additionally has an N parameter, which allows you to select the filter length in the synthesis process. In this work, it was experimentally found that the appropriate value for filter smoothing is N=4.

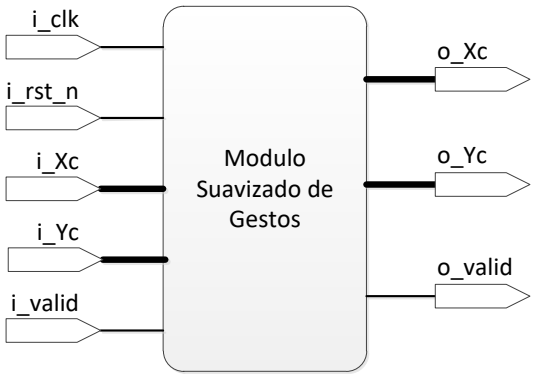


Figure 76.Schematic diagram of the gesture path smoothing module in hardware. Source: author.

7.3 Gesture *Spotting*

According to (M. O. S. M. Elmezain, 2010; M Elmezain, Al-Hamadi, Appenrodt, et al., 2008; M Elmezain et al., 2009; Mahmoud Elmezain & Al-Hamadi, 2012) a gesture can be divided into two types. Communicative gesture and non-communicative gesture. The communicative gesture (gesture-principal) is defined as a part of the hand's trajectory that carries explicit information, while a non-communicative gesture represents involuntary movements which are used to connect main gestures. Non-communicative gestures are performed at the beginning (pre-gesture) and at the end (post-gesture) of a main gesture.

In a gesture recognition system, one of the most difficult problems to solve is the detection of the main gesture ("*Gesture Spotting*"). Normally, the detection process is carried out by searching for the starting and ending points of the gesture based on a continuous sequence of hand movements. However, this process is too complex due to the evident variability of the form, performance and duration of a gesture, as well as the meaningless information associated with it (pre-gesture and post-gesture). For this reason, the gestures used in this work are performed taking into account a relative position of common beginning and end, simplifying the performance of the pre-gesture and post-gesture respectively. This allows the gesture to be detected only when the hand is set in motion.

The performance of a gesture is carried out in three phases, taking into account the natural composition of a gesture: in the first phase (pre-gesture), the hand must be stopped in the selected starting position, the second phase (main gesture) consists of the execution of the gesture by moving the hand on the desired trajectory until it reaches the starting position again, In the third phase (post-gesture) the hand stops indicating the completion of the executed gesture. The hand then remains stationary until the start of a new gesture. Figure 77 shows an example illustrating the execution phases corresponding to the "Rotate Right 90°" gesture. This figure shows the behavior of the centroid during the performance of the gesture in 80 consecutive images. The blue graph corresponds to the coordinate of x the centroid and the red graph corresponds to the coordinate y of the centroid.

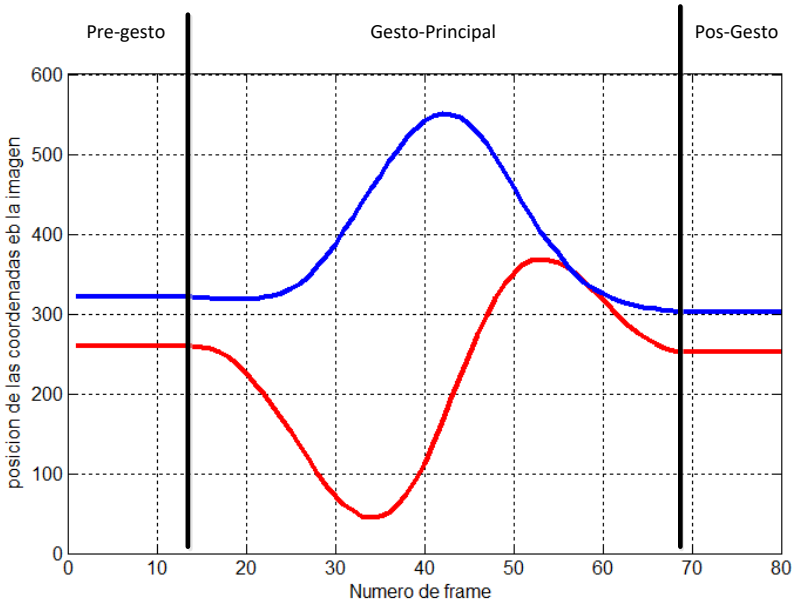


Figure 77. Phases of the execution of the "Rotate right 90º" gesture. Source: author.

According to the behavior of the gesture shown in Figure 77. The pre-gesture and post-gesture correspond to the absence of movement. For this reason, the value of the centroid in these phases of the execution of the gesture should not be taken into account as part of it. This allows the gesture-principal to be detected by observing the movement of the centroid in consecutive images. To determine the movement of the centroid over time t, the difference of the centroid with respect to K previous centroid values is calculated. If there is a difference greater than a threshold value T_m , the centroid (x_t, y_t) is considered valid and therefore forms part of a principal-gesture.

Figure 78 presents the algorithm developed for the detection of the principal-gesture from the movement of the centroid coordinates in consecutive images. To determine the movement of the centroid, the difference of each of its coordinates with K previous values is calculated. If any of the coordinates presents a difference greater than a threshold value, the centroid is considered valid and therefore the gesture at that instant of time is detected. The values of K y T_m were experimentally calculated in this work obtaining $K = 8$ and $T_m = 30$.

procedure DETECCION DEL GESTO($C_x[t]$,$C_y[t]$)
    Input: $C_x[t]$: Coordenada x del centroide en un instante de tiempo t
            $C_y[t]$: Coordenada y del centroide en un instante de tiempo t

    Output: G_{valido}: salida que indica la validez del gesto en el tiempo t

    $F_x[i] \leftarrow |C_x[t] - C_x[t-i]|$ para $3 \leq i \leq K$
    $F_y[i] \leftarrow |C_y[t] - C_y[t-i]|$ para $3 \leq i \leq K$

    if $F_x[i] > T_m$ o $F_y[i] > T_m$ **then**
        $G_{valido}[t] \leftarrow 1$
    else
        $G_{valido}[t] \leftarrow 0$
    end if
    return G_{valido}
end procedure

Figure 78. An algorithm that describes the process for detecting a gesture.

The hardware implementation of gesture detection is based on the algorithm presented in Figure 78. Figure 79 shows the hardware design made for the detection of a lead-gesture. This module performs the process of detecting the gesture while it is executed by the user. The module captures each centroid value coming from the gesture smoothing module and calculates the difference from the previous values. If one of the calculated differences is greater than the selected threshold value, the o_Gvalido output is turned high indicating that the captured centroid values correspond to a lead-gesture.

Figure 80 shows the schematic diagram for the gesture detection module, indicating the module's interface to the outside. The *i_Xc* and *i_Yc* signals correspond to the centroid values coming from the gesture smoothing module. Centroid values are captured on each ascending edge of the i_clk reloc signal, as long as the i_valid signal is at a high logical level. The *o_Xc* and *o_Yc* signals correspond to the output centroid values detected as part of a gesture-main. The *o_gesto signal* indicates the detection of a gesture-principal, which remains at 1 during gesture detection and goes to 0 when there is no detection of a gesture. The o_valid signal indicates the validity of the output data.

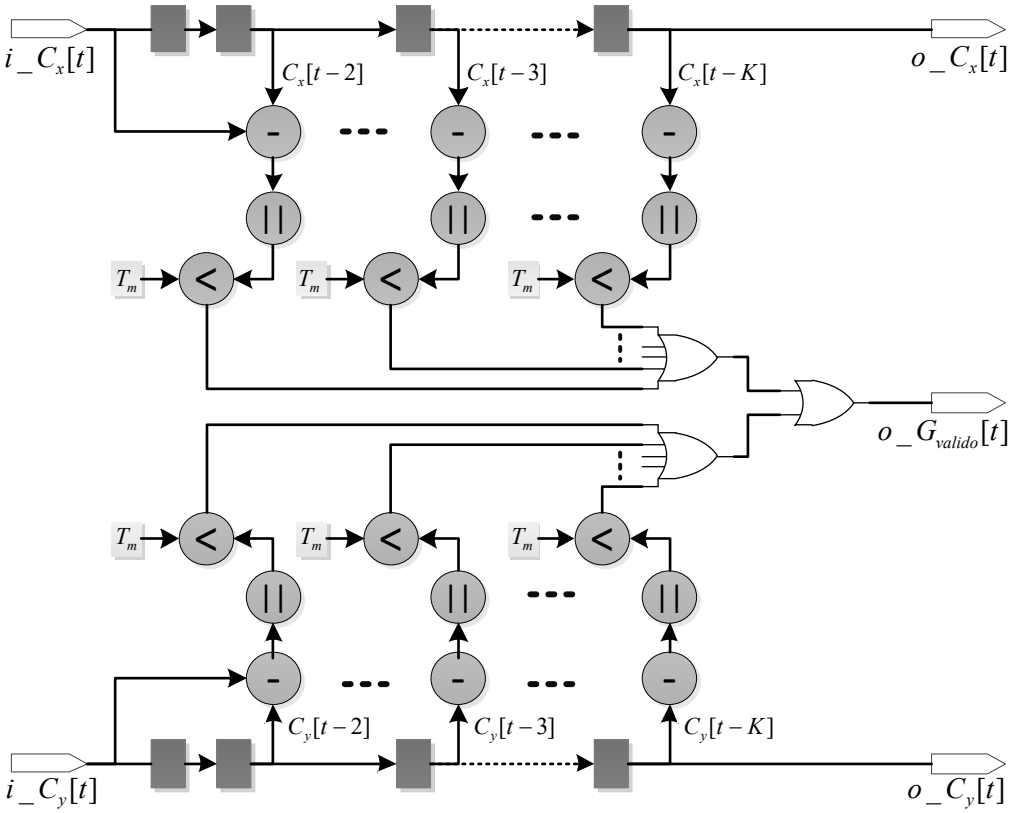


Figure 79. Hardware implementation of valid gesture detection. Source: author.

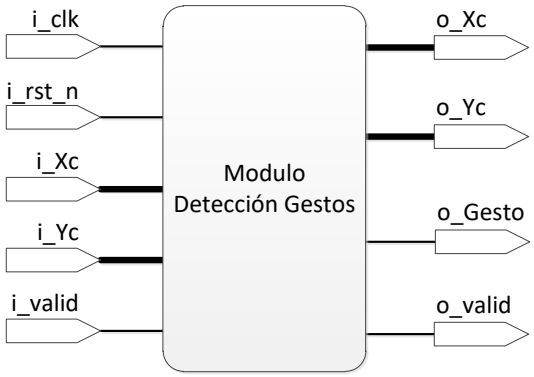


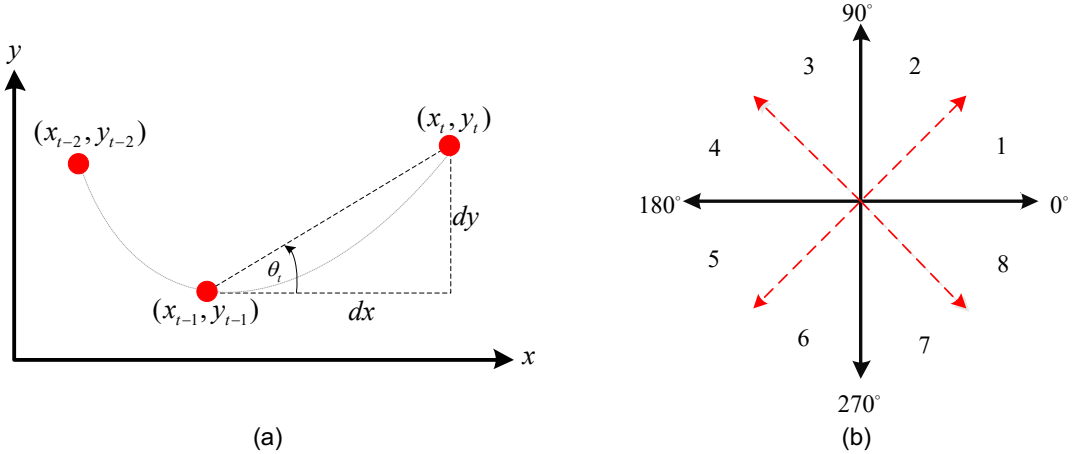
Figure 80. Schematic diagram of gesture detection module implemented in hardware. Source: author.

7.4 Feature Extraction

Selecting the best features for hand gesture path recognition plays an important role in the operation of the system. These characteristics are used in the recognition stage to properly distinguish one gesture from another. There are currently three basic characteristics in the literature: location, orientation and speed. According to (M Elmezain, Al-Hamadi, Appenrodt, et al., 2008) the orientation characteristic is the best in terms of accuracy results. For this reason, orientation is selected as the main feature of the recognition system in this work.

The trajectory of a gesture is a spatio-temporal pattern that is made up of a set of points of the centroid of the hand (x_{mano}, y_{mano}). The orientation feature used to describe the gesture allows the direction of the hand to be calculated as it travels through Cartesian space during the gesture. Therefore, the orientation is determined by two consecutive points of the gesture path as shown in equation (28), where θ_t it represents the orientation of the centroid at the instant of time t, and T represents the total length of the gesture.

$$\theta_t = \tan^{-1}\left(\frac{y_t - y_{t-1}}{x_t - x_{t-1}}\right); \quad t = 2,3,...,T \tag{28}$$



(a) (b)

Figure 81. Orientation calculation. (a) Orientation between two consecutive points of the centroid. (b) discrete coding from 1 to 8 dividing the orientation into 45º. Source: author.

Once the orientation has been extracted (see Figure 81 (a)) a normalization or quantization process must be performed to obtain the discrete symbols that are used as input to the HMMs. This quantization process is carried out by dividing the orientation by 45º, which allows obtaining the finite symbol alphabet required by the recognition system based on HMMs.

Any orientation value is represented by a specific symbol, for example if the orientation is given in a range of 0 to 45º the symbol corresponds to 1. If the orientation is in the range 135º to 180º, the assigned symbol corresponds to 4 and so on for the other orientation values (see Figure 81 (b)). This quantization process allows the gesture made to be transformed into a sequence of discrete observations. Figure 82 shows the algorithm that describes the extraction of the orientation θ_t and its quantization value O_t for two consecutive centroid values (x_t, y_t) and (x_{t-1}, y_{t-1}). The quantization value O_t enters directly into the recognition system based on HMMs, since this value corresponds to a discrete observation of the gesture at the instant of time t.

procedure CARACTERISTICAS(Cx_t,Cx_{t-1},Cy_t,Cy_{t-1})
    Input: Cx_t: Coordenada x del centroide en un instante de tiempo t
        Cx_{t-1}: Coordenada x del centroide en un instante de tiempo $t-1$
        Cy_t: Coordenada y del centroide en un instante de tiempo t
        Cy_{t-1}: Coordenada y del centroide en un instante de tiempo $t-1$

    Output: O_t: símbolo correspondiente a la cuantizacion de la orientación calculada

    $dx \leftarrow (Cx_t - Cx_{t-1})$
    $dy \leftarrow (Cy_t - Cy_{t-1})$

    if $dx \geq 0$ y $dy > 0$ **then**
        $\theta_t \leftarrow \arctan\left(\frac{dy}{dx}\right)\left(\frac{180}{\pi}\right)$
    else if $dx < 0$ y $dy \geq 0$ **then**
        $dx \leftarrow |dx|$
        $\theta_t \leftarrow 180° - \arctan\left(\frac{dy}{dx}\right)\left(\frac{180}{\pi}\right)$
    else if $dx < 0$ y $dy < 0$ **then**
        $dx \leftarrow |dx|$
        $dy \leftarrow |dy|$
        $\theta_t \leftarrow 180° + \arctan\left(\frac{dy}{dx}\right)\left(\frac{180}{\pi}\right)$
    else if $dx > 0$ y $dy < 0$ **then**
        $dy \leftarrow |dy|$
        $\theta_t \leftarrow 360° - \arctan\left(\frac{dy}{dx}\right)\left(\frac{180}{\pi}\right)$
    end if
    $O_t \leftarrow truncar\left(\frac{\theta_t}{45°}\right)$
    return O_t
end procedure

Figure 82. An algorithm that describes the extraction of the orientation and its quantization value for two consecutive centroid values. Fountain...

7.4.1 FPGA-HPS Connection Interface and Feature Extraction

In this work, the process of extracting the orientation as a characteristic that describes the trajectory of the centroid is carried out in the physical processor system (HPS) of the Cyclone V SOC device used in this project. The decision to use this device resource is mainly based on three important reasons: orientation calculation and quantization do not require intensive processing, HPS does not make additional use of FPGA logic resources, and HPS has the ability to easily perform arithmetic operations with floating-point representation.

The reason why the orientation calculation does not require intensive processing is mainly because the operation involves only two consecutive points (centroid values), unlike an image which contains too much information that must be processed by applying several transformations simultaneously during the capture time. For this reason, the implementation of the orientation calculation can be developed in a conventional processor.

The HPS of the Cyclone V SoC device contains a microprocessor-based system that includes a microprocessor unit (MPU) with two ARM Cortex-A9 cores, a Flash memory controller, an SDRAM memory controller, on-Chip memory, peripheral connection

interfaces, etc. This system is a physically implemented resource within the Cyclone V SOC chip totally independent of the FPGA zone itself, which means that its use does not require additional FPGA hardware for its operation as is the case with SoftCores such as the Nios II. Therefore, the use of HPS allows all the reconfigurable logical resources of the FPGA to be available for the development of the hardware required in image processing, such as centroid extraction, gesture smoothing, and gesture-main detection.

Because the extraction of the orientation feature and gesture recognition based on HMMs require arithmetic operations with numerical floating-point representation (such as the operation $\tan^{-1}$), and since the processing intensity for such calculations is not high compared to image processing, these operations can be perfectly implemented in the HPS system.

To perform orientation extraction as the main feature in gesture recognition, it is necessary to establish communication between the HPS and the processing hardware developed in the FPGA so that the gesture detected in hardware can be manipulated by the processor. However, hardware developed on the FPGA cannot communicate directly with the HPS. For this, the device has special interfaces for this purpose.

According to the official documentation of the device manufacturer (see Annex A) the HPS system and the FPGA communicate internally through three specific interfaces: *HPS-to-FPGA Bridge*, *FPGA-to-HPS Bridge* and Lightweight HPS-to-FPGA Bridge. The *Lightweight HPS-to-FPGA Bridge* allows the processor to easily access peripherals developed in the FPGA via the *Avalon-Memory-Maped (Avalon-MM)* bus. In this type of interface, the processor behaves as a master using the *Avalon-MM* bus, and the peripherals on the FPGA side as slaves to the *Avalon-MM* bus. The other interfaces have a similar operation with more advanced features which are not addressed in this work.

In order for the processor to be able to access the information of the detected gesture, an additional hardware module is designed in the FPGA. This module is designed to store the data corresponding to a detected gesture and keeps it until the processor can access it. This module allows synchronization of the data obtained in hardware with the processing developed in the HPS system. The developed storage module receives the *data stream* from the gesture detection module and stores it in a FIFO memory. To extract the data from the FIFO memory, a slave interface compatible with the Avalon-MM bus is designed, which connects to the HPS system via the *Lightweight HPS-to-FPGA Bridge*. Annex A presents the documentation of the Avalon-MM bus used for the design of the slave interface taking into account the specifications for compatibility with that bus. Figure 83 shows a block diagram illustrating the interconnection between the FPGA processing system and the HPS system via the *Lightweight HPS-to-FPGA Bridge* via Avalon-MM bus connection.

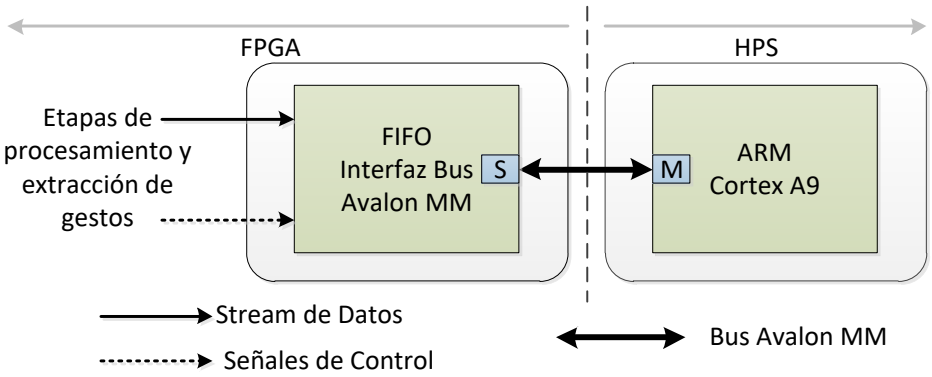


Figure 83. Interconnection between FPGA and HPS processing through the *Lightweight HPS-to-FPGA Bridge*. Source: author.

Table 12. Map of FIFO memory module registers.

In-Memory Mapped Records								
Offset en bytes	**Registry Name**	**R/W**	**Description of Bits**					
			31… 5	4	3	2	1	0
0	Dato_X	R	Value of the x-coordinate of the centroid					
4	Dato_Y	R	Coordinate and centroid value					
8	Reserved	-	-	-	-	-	-	-
12	Control	RW	-	FIFO - FULL	FIFO-CLR	Gesture	FIFO Data	GO
16	Longitud_X	R	Amount of data available for x					
20	Longitud_Y	R	Amount of data available for and					

The HPS system processor interacts and communicates with the hardware developed in FPGAs using the Avalon-MM bus through six memory-mapped 32-bit registers. Table 12 shows the map of records available in the designed FIFO storage module. The Dato_X and Dato_Y registers are read-only and allow the processor to read the centroid value for a detected gesture. The Longitud_X and Longitud_Y registers are read-only registers and allow you to determine the amount of data available in the module waiting to be read by the processor. Each time a reading is made on the records Dato_X or Dato_Y the value of the records Longitud_X or Longitud_Y decreases. The control register is a read-write register which allows you to control the behavior of the FIFO module.

Table 13. Description of the bits that make up the FIFO module's control register.

Control Register Bits			
Bit Number	**Bit Name**	**R/W**	**Description**
0	GO	R/W	Enables FIFO memory writes from the processing module on FPGAs 1=enable 0=disable
1	FIFO Data	R	Indicates whether there is data available to be read. 1=Data available. 0=no data available
2	Gesture	R	Indicates whether the captured gesture is valid 1= No valid gesture detected 0= a valid gesture detected
3	FIFO-CLR	R/W	Clear the contents of the FIFO memory

			1= Clear FIFO memory. When the deletion is finished, it is automatically set to 0
4	FIFO-FULL	R	Indicates that the FIFO memory has been full. 0= memory has available space 1= memory is completely filled

The control register has bits for controlling the module and bits that indicate its status. Bit 0 (GO) is a read and write bit that allows you to enable or disable data input from the gesture detection and processing system. Bit 1 (Dato_FIFO) is a read-only bit and indicates the availability of data from the centroid to be read by the processor. Bit 2 (Gesture) is a read-only bit and informs whether the value of the centroid corresponds to a detected gesture. Bit 3 (FIFO-CLR) is a read-write bit and allows the contents of FIFO memory to be erased. To clear the FIFO memory this bit must be set to logical 1, it automatically goes to 0 when the memory data has been deleted. Bit 4 is a read-only bit and indicates that FIFO memory has reached its maximum capacity. Table 13 describes the function of each bit of the control register.

The implementation of feature extraction and recognition system is accomplished by developing a software application for a Linux operating system that runs on the device's HPS. This application accesses the data of the detected gesture through the records mapped in memory, extracts the characteristics and evaluates the HMMs to classify the gesture performed. The extraction of the characteristics is carried out by a software function from the algorithm presented in Figure 82. The software implementation of the recognition system is presented in the next chapter.

The gesture smoothing and gesture detection modules were developed following a parametric description in VHDL language. A description of the modules is provided in annex D. Annex E shows the simulation results obtained using the Modelsim-Altera software for the case.

8 Recognition

This chapter presents the development of the last stage that makes up the recognition system proposed in this work. This stage takes the features extracted from the previous stage and performs a sorting process to determine the gesture to which they belong, in order to send a command to control the actions of a LEGO NXT Robot. In this work, the Hidden Markov HMM Models are used to carry out the classification process of the hand gestures used.

HMMs can be used in different ways to perform gesture recognition tasks. A fairly common way (used in this work) is to model each of the gestures using an HMM. Gesture recognition using this structure is done through two main steps. i) construct an HMM λ for each gesture

and estimate the parameters (A, B, π) by means of a training process carried out with the Waum-Welch algorithm, using as training data a set of observation vectors for each gesture. ii) recognize an unknown gesture from the sequence of observations obtained in the feature extraction stage. This step calculates the probability of each model with respect to the sequence of gesture observations. The model whose probability is greater than the probability of the other models will correspond to the input gesture made. Figure 84 presents a block diagram of the HMM-based recognition system used in this work.

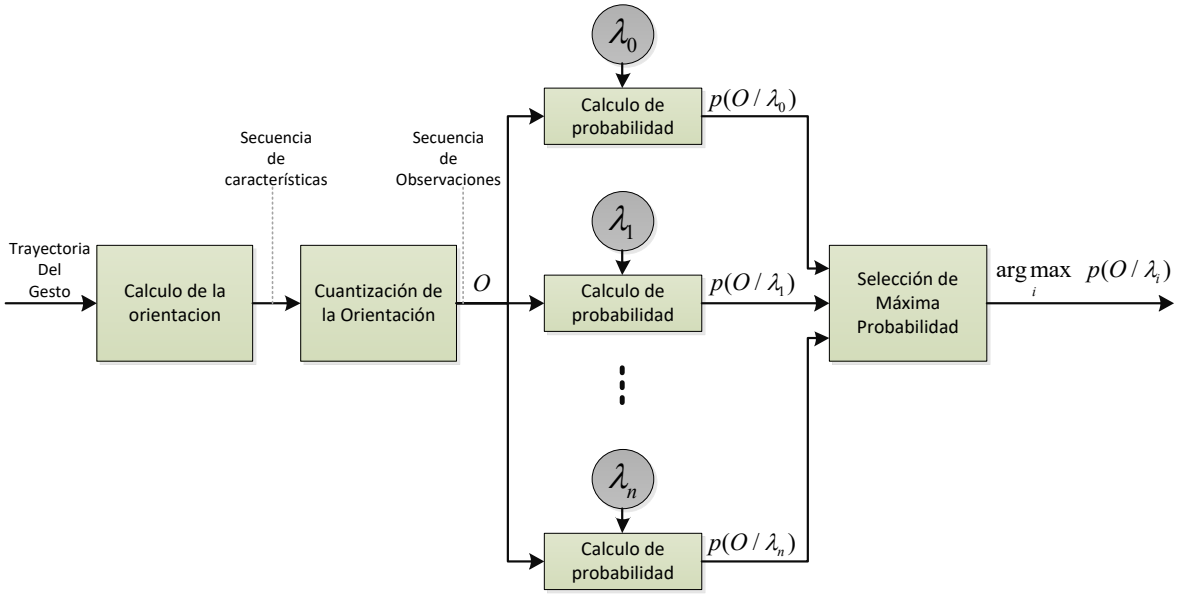


Figure 84. Block diagram of the gesture recognition stage using HMM. Adaptation based on (M. O. S. M. Elmezain, 2010; Rabiner, 1989).

8.1 HMM Model Selection

To build an HMM, the model must be selected according to three important parameters: selection of the model topology (ergodic, or left-right), selection of the observation symbols to be used by the model (discrete or continuous) and selection of the size of the model (number of hidden states). Unfortunately, there is no theoretical rule that allows selecting these parameters in a simple way, mainly because the model depends on the data to be modeled (Rabiner, 1989).

An HMM can be built from three different topologies: fully connected (ergodic model) in which any state can be reached from any other state, left-right in which each state can go to the next states or return to itself, and LRB (left-right banded) in which each state can only go to the next state or return to itself. Figure 85 presents an example of the 6-state LRB topology. For pattern recognition applications, left-right (LR) topology has been shown to perform better than ergodic (FC) topology (N. Liu, Lovell, Kootsookos, & Davis, 2004; Siriboon, Jirayusakul, & Kruatrachue, 2002). In the particular case of hand gesture recognition, the LRB topology presents better results than the conventional LR topology (M

Elmezain, Al-Hamadi, Appenrodt, et al., 2008; M Elmezain, Al-Hamadi, & Michaelis, 2008; M Elmezain et al., 2009; Mahmoud Elmezain, Al-Hamadi, Appenrodt, & Michaelis, 2009; N. Liu et al., 2004). For the present work, the LRB topology is selected for the construction of the HMM of the recognition system.

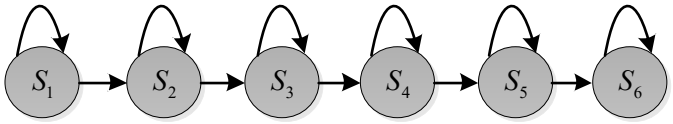


Figure 85. 6-state LRB (Left-Right Banded) topology for an HMM. Figure adapted by the author from (M. O. S. M. Elmezain, 2010; Rabiner, 1989)

Once the HMM topology was selected, the observation symbols to be used by the model were then selected, which can be discrete or continuous. Observations in the recognition of hand gestures are usually considered to be discrete symbols (M Elmezain, Al-Hamadi, Appenrodt, et al., 2008; M Elmezain, Al-Hamadi, & Michaelis, 2008; M Elmezain et al., 2009; Mahmoud Elmezain et al., 2009; N. Liu et al., 2004; Siriboon et al., 2002). In this work, the sequence of observations is obtained by quantizing the sequence of orientations calculated in the feature extraction stage. Therefore, the observation symbols used by the model are discrete symbols 1-8.

Deciding how many states the model requires to adequately describe a particular sequence of observations is a complex task. When the number of training data samples is insufficient, the use of an excessive number of states causes an overtraining problem. In addition, the discriminating power of HMMs decreases when an insufficient number of states are used, because more than one segmented part of the graph pattern is modeled in a single state. However, there are several methods and criteria developed with the purpose of adequately selecting this number of states, such as the Akaike Information Criterion (AIC), the Bayes Information Criterion (BIC), cross-validation, etc. (Le, Chatelain, & Berenguer, 2014). In this work, the *k-fold-Cross-Validation cross-validation* method is used to select the appropriate number of HMMs used in the proposed recognition system.

8.2 Initializing HMM Parameters with LRB Topology

Before starting the iterative Baum-Welch algorithm, the initial values for all parameters in the HMMs must be assigned. A good initialization of the (A, B, π) parameters of the HMMs allows to achieve better results in the recognition system. According to (M Elmezain, Al-Hamadi, & Michaelis, 2008; N. Liu et al., 2004) the initialization of the parameters of the HMMs should be performed as shown below, taking into account that the transition matrix A is the first parameter to be initialized as defined in equation (29).

$$A = \begin{pmatrix} a_{11} & 1-a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & 1-a_{22} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix} \tag{29}$$

The elements of the diagonal a_{ii} of the transition matrix can be selected to indicate approximately the average duration d in each state. Equation (30) shows the calculation of the average duration per state, being T the length of the path of the gesture performed and N the number of states. In this case, an average value of $T = 60$ since the gestures used have variable lengths between 50 and 80 data points was taken. Equation (31) presents the calculation of the transitions a_{ii}.

$$d = \frac{T}{N} \tag{30}$$

$$a_{ii} = 1 - \frac{1}{d} \tag{31}$$

The observation matrix B is determined by equation (32). Because HMM states are discrete, all elements in the array B can be initialized with the same value for all states. B is a matrix of $N \times M$ where b_{im} is the probability of emitting a symbol v_m in the state i.

$$B = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1M} \\ b_{21} & b_{22} & \cdots & b_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ b_{N1} & b_{N2} & \cdots & b_{Nm} \end{pmatrix};\ b_{im} = \frac{1}{M} \tag{32}$$

Since state transitions can only be executed on the same or the next state on each observation. The initial probability vector π should be initialized as shown in equation (33). This ensures the correct start of the model in the first state.

$$\pi = (1 \quad 0 \quad \cdots \quad 0)^T \tag{33}$$

8.3 HMM Training

Training of the initialized parameters (A, B, π) of an HMM is carried out using the Baum-Welch algorithm (see section 3.4.1). This algorithm takes as inputs the initialized model parameters and a set of discrete vectors to estimate a new transition matrix A and a new observation matrix B. The trained model must be flexible enough to represent a test sequence not used during training. Training is repeated until the transition and observation matrices converge or the maximum number of iterations is reached (M Elmezain, Al-Hamadi, & Michaelis, 2008).

To perform the training process of each of the HMMs of the gesture recognition system, the HMM toobox for MATLAB developed by Kevin Murphy (Murphy, 1998) is used. Training is carried out using the *dhmm_em* function of the toolbox. This function performs the training of an HMM with discrete outputs using the Baum-Welch algorithm. The training parameters used are: maximum number of iterations 5000 and convergence tolerance 0.0001. The training process is carried out *offline* in MATLAB. To carry out the training, 60 samples were taken for each gesture, performed by 3 different people, obtaining a total of 1260 samples.

8.4 K-fold *Cross Validation* and HMM Number of State Selection

Cross-validation is a statistical method of evaluating and comparing learning algorithms by dividing data into two segments: one used to train a model and the other used to validate the trained model. In typical cross-validation, training and validation sets must be crossed in successive rounds so that each dataset has a chance to be validated.

The basic form of cross-validation is k-fold cross-validation (Refaeilzadeh, Tang, & Liu, 2009). In this type of validation, the data is first divided into k segments or groups of equal size. Subsequently, k iterations of training and validation are performed in such a way that within each iteration a different set of data is used for validation, while the remaining k-1 groups are used for training. Figure 86 shows an example taking a value of k = 3. The darker section of the data is used for training while the lighter sections are used for validation. In machine learning applications, 10-fold cross-validation (k = 10) is the most common (Refaeilzadeh et al., 2009).

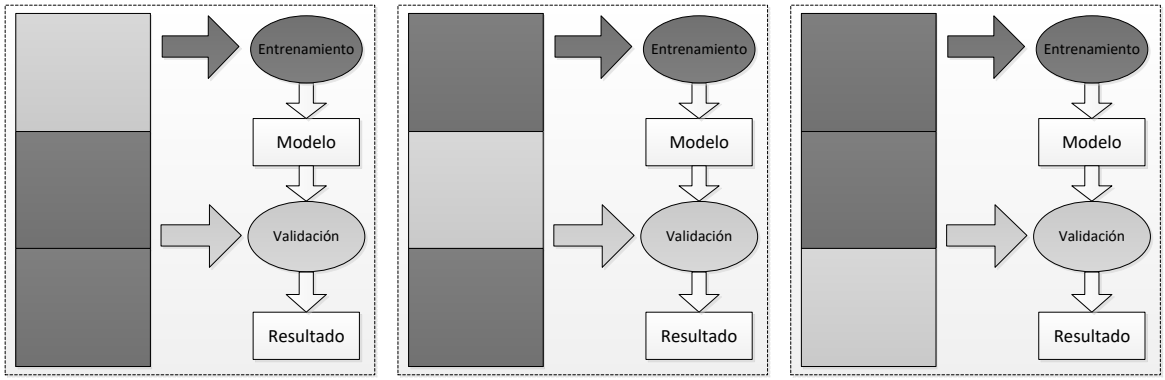


Figure 86. 3-fold cross-validation procedure. Figure adapted by the author from (Refaeilzadeh et al., 2009)

Cross-validation is critical in selecting the right parameters that maximize a model's performance. In this work, the k-fold cross-validation method is used to appropriately select the size (number of hidden states) of the HMMs used in the proposed gesture recognition system. The k-fold cross-validation for the selection of the appropriate size of the HMMs is carried out as follows:

i. Build Q HMMs with a number n of hidden states each. Also, initialize the parameters (A, B, π) as explained in section 8.2. The number Q of HMMs to build depends on the number of gestures to be recognized. In the particular case of this work, 7 HMMs are built.

ii. Divide the available data for each gesture into K equal groups. The same amount of data must be available for each of the gestures used in the training. In this work, a database with 180 samples for each gesture is used.

iii. Perform the training of each HMM separately, using the data of the gesture that corresponds to the model. To carry out this training, K-1 groups obtained in the previous stage must be used, leaving 1 for validation. The training process is carried out as explained in section 8.3.

iv. Validate the HMM-based classifier using data that was not used in training. This procedure is carried out by evaluating the validation data of each gesture, using all the trained HMMs. This assessment is made by calculating the probability $P(O/\lambda_i)$ in each HMM, using the *forward* algorithm (see section 3.4.1). Once the probabilities of the HMMs have been calculated, each of the validation data is classified as a certain gesture whose model obtains the maximum probability value. Finally, the accuracy of the classifier is obtained ACC_k from the confusion matrix with the validation data.

v. Repeat steps *iii* and *iv* K times and calculate the cross-validation accuracy using equation (34).

$$CV(\lambda) = \frac{1}{K}\sum_{k=1}^{K} ACC_k(\lambda) \tag{34}$$

vi. Repeat steps i through v for HMMs of different sizes and get the accuracy of cross-validation for each model.

After performing the validation process for the classifier based on HMMs with different sizes, the number of states that produces the highest accuracy value of the classifier is finally selected. In this work, the cross-validation process is carried out using 2 to 20 states through k-fold cross-validation with a value of k=10. The training and validation is carried out *offline* using the MATLAB software and the best result obtained in this process is implemented in the gesture recognition system developed in the SoC system used.

8.5 Implementation of the Recognition Stage in the HPS

As explained in section 3.4.1, the problem 1 of an HMM is to evaluate the probability of a sequence of observations given a model λ. This problem can be solved efficiently using the *Forward algorithm*. Forward's algorithm runs in three main stages: initialization, recursion, and completion. Initialization calculates the value of the Forward α algorithm *variable* from the parameters of (A,B,π) the trained HMM and the first observed symbol. Figure 87 presents a description of the initialization procedure of the *Forward algorithm*.

```
1: procedure HMM FW INIT(A,B,π,O_1)
2:     Input: A: Matriz de Transición de Estados del HMM
3:            B: Matriz de Observaciones del HMM
4:            π: Vector de Probabilidad inicial del HMM
5:            O_1: Primer simbolo de la secuencia de observaciones
6:
7:     Output: α: variable del algoritmo de forward inicializada
8:
9:     for i ← 1, N do
10:        α_1[i] ← π[i] × B[i][O_1]
11:    end for
12:    return α
13: end procedure
```

Figure 87. Initialization of the forward algorithm.

```
1: procedure HMM FW ITER(A,B,π,O_t,α_{t−1})
2:     Input: A: Matriz de Transición de Estados del HMM
3:            B: Matriz de Observaciones del HMM
4:            π: Vector de Probabilidad inicial del HMM
5:            O_t: símbolo de la secuencia de observaciones en el instante de tiempo t
6:            α_{t−1}: variable del algoritmo de forward calculada en t − 1
7:
8:     Output: α_t: valor de la variable de forward obtenida a partir de la observacion O_t
9:
10:    for i ← 1, N do
11:        α_t[i] ← 0
12:        for j ← 1, N do
13:            α_t[i] ← α_t[i] + α_{t−1}[i] × A[j][i]
14:        end for
15:        α_t[i] ← α_t[i] × B[i][O_t]
16:    end for
17:    return α_t
18: end procedure
```

Figure 88. Evaluation of the forward algorithm for an observed symbol at an instant of time t.

The recursion stage updates the value of the variable α at the time slot t by taking the symbol observed at that time slot and the parameters (A,B,π) of the trained HMM. Figure 88 presents the procedure that describes the recursion stage of the *Forward* algorithm at an instant of time t.

The completion stage calculates the probability of the observed sequence, given the model λ with parameters (A,B,π). At this stage, the probability of observing the entire sequence is given by the sum of all possible final states present in the variable α calculated for the last observed symbol. Figure 89 presents the procedure describing the completion of the *Forward* algorithm for calculating the probability of an HMM, given a sequence of observations O.

```
1: procedure HMM FW END(α_{t-1})
2:     Input: α_{t-1}: variable del algoritmo de forward calculada en t − 1
3:
4:     Output: P: el valor de la probabilidad calculada P(O/λ)
5:
6:     P ← 0
7:     for i ← 1, N do
8:         P ← P + α_{t-1}[i]
9:     end for
10:    return P
11: end procedure
```

Figure 89. Completion of the Forward algorithm to calculate the total probability of the HMM.

The recognition stage is implemented in the HPS by developing a software application to be run on a Linux operating system. For the development of the software application, the guide set out in the manufacturer's documentation is taken into account, which can be consulted in Annex A. This application is designed to interact with the image processing system developed in FPGA (see section 7.4.1), recognize the gesture made using the HMM-based classifier, and send a command to the LEGO NXT robot to execute the action accordingly.

The application is designed using an FSM that allows the HMM-based classifier in Figure 84 to be implemented. This FSM extracts the value of an observation symbol at any moment in time t and calculates the probability of said symbol using the *Forward* algorithm applied to each of the previously trained HMMs. In addition, the FSM classifies the observed gesture trajectory according to the model whose probability is higher than the probability of the other models used. The result of this classification is used to send a command to the robot with the action corresponding to the gesture performed.

Figure 90 shows the diagram of the status of the FSM used to carry out the recognition stage. Table 14 presents the description of the inputs used to determine the transitions of the WFTU. These entries are derived from the memory-mapped control log of the FIFO storage peripheral explained in section 7.4.1. The WFTU starts in the S0 state. If a valid gesture is currently available, the machine remains in S0 and gives the command to delete the contents of the FIFO component. If there is no gesture available, the FSM goes to the S1 state and enables the FIFO component to receive data.

In the S1 state, it is evaluated if there is a gesture available, if so, the machine transitions to the S2 state, otherwise it is returned to S0 starting the process again. When the FSM is in the S2 state there are three possibilities: i) there is a valid gesture but there is no data available to be processed therefore it remains in the S2 state, ii) there is a valid gesture and there is also data available, in this case a centroid data is extracted from the FIFO memory and transitioned to the S3 state, iii) if there is no valid gesture, a transition is made to the indicated S5 state at the end of the gesture.

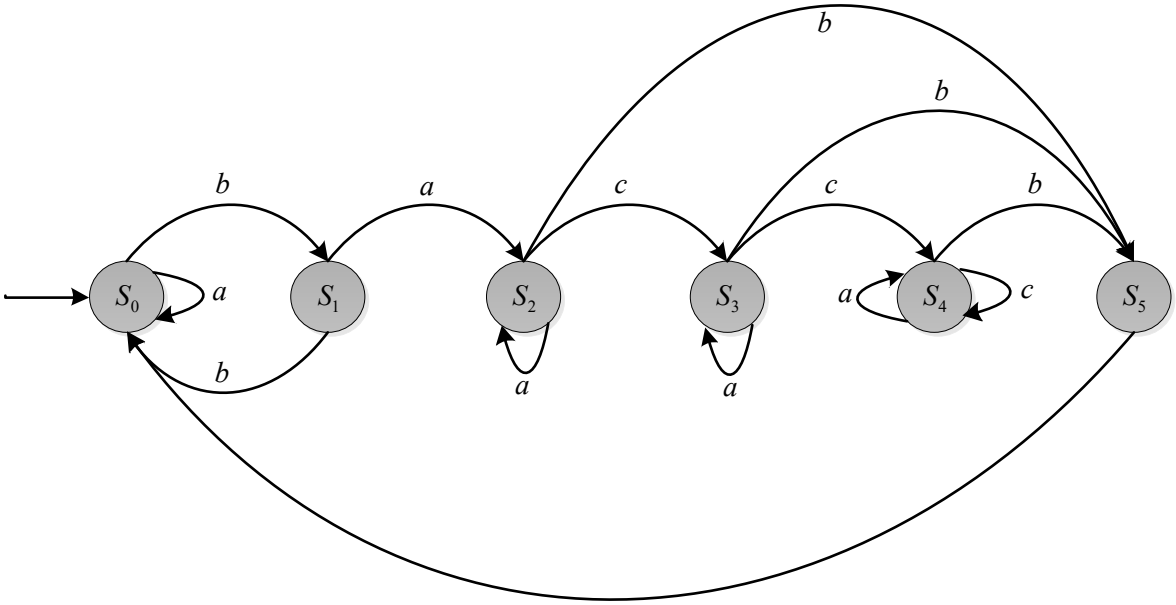


Figure 90. Diagram of the states of the recognition stage developed in software. Source: author.

Table 14. Description of the WSF entries of the recognition system.

WFTU Tickets	To	B	C
Description	Valid gesture	Invalid gesture	Valid gesture and data available in FIFO

When the FSM is in the S3 state, any of the following actions are performed: i) if a valid gesture exists but no data is available for processing, the FSM remains in the S3 state, ii) if a valid gesture exists and data is available, the FSM transitions to the S4 state, extracts a centroid value from the FIFO storage edge, it performs the extraction of the first observation symbol (see Figure 82) and initializes the Forward algorithm , iii) if there is no valid gesture, the FSM transitions to the S5 state.

When the FSM is in the S4 state, any of the following actions are performed: i) if there is a valid gesture but no data is available, the FSM remains in the S4 state, ii) if there is a valid gesture and available data, the FSM remains in the S4 state, extracts a centroid value from the FIFO storage peripheral, obtains the corresponding observation symbol and performs

the recursion step of the Forward algorithm , iii) if there is no valid gesture, the FSM transitions to the S5 state.

In the S5 state, the completion stage of the Forward algorithm is performed, the gesture performed is classified according to the HMM with the highest probability value obtained, and the command is sent to the robot as appropriate. In this state, the FSM finishes the process of recognizing a gesture, so it transitions to the S0 state to start the process with a new gesture.

8.5.1 Control del robot Lego Mindstorms NXT

To control the actions performed by the robot, the recognition system sends the commands via Bluetooth communication. To establish this type of communication with the robot, the RN-42 Bluetooth device is used. The communication between the developed application and the Bluetooth device is carried out through an RS232 IP core of the Altera® University Program, which allows the device to be configured and paired with the Robot. This IP *core* communicates with the HPS via the Avalon-MM bus via the *Lightweight HPS-to-FPGA Bridge* in a manner similar to the module explained in section 7.4.1. The specifications of the RS232 module used are presented in Annex A. Figure 91 shows a block diagram showing the components involved in the robot's communication with the recognition system.

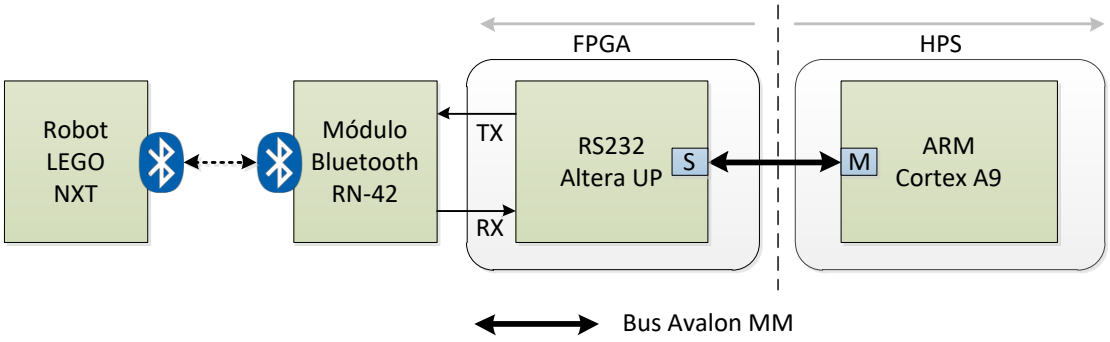


Figure 91. Communication between the recognition system and the LEGO NXT Robot. Source: author.

The RN-42 Bluetooth device is configured to operate at 115,200 baud, 8 data bits, 1 stop bit, and 0 parity bits. In addition, the software application developed for the recognition system pairs with the LEGO NXT robot using the RN-42 device as the master. These configurations are carried out following the reference manual of the device, which can be consulted in Annex A.

To send the actions to be carried out on the robot according to the gesture performed, it is necessary to use the message exchange protocol established by the robot manufacturer (see Annex A). This message exchange protocol requires sending a string of bytes from the master device. Figure 92 presents the description of this protocol. Bytes 0 and 1 correspond to the total length in bytes of the message to be sent. Byte 0 being the LSB part and byte 1 being the MSB part. Byte 2 allows the receiver to be configured to send the transmission status after receiving the message (0x80 returns state, 0x00 does not return state). Byte 3

states that a write will be performed. Byte 4 selects the mailbox where the message will be sent (0-9). In byte 5 sets the length in bytes of the message to be sent (N+1). From byte 6 onwards, you have N positions to send a message. The N+6 byte informs the completion of the message, therefore it should always be 0x00. In accordance with the above, Table 15 presents the commands sent to the robot from the application, taking into account the protocol established by the robot manufacturer.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	N bytes	Byte N+6
Longitud LSB	Longitud MSB	Retorno estado	0x09	Mailbox	N+1	Mensaje	0x00

Figure 92. LEGO NXT Bluetooth message exchange protocol. Source: author.

Table 15. List of commands used to control the robot according to the gesture made.

Identified gesture	**Command to Send**
Go ahead	{0x0D,0x00,0x80,0x09,0x00,0x09,'A','d','e','l','a','n','t','e',0x00}
Behind	{0x0A,0x00,0x80,0x09,0x00,0x06,'A','t','r','a','s',0x00}
Right	{0x0C,0x00,0x80,0x09,0x00,0x08,'D','e','r','e','c','h','a',0x00}
Left	{0x0E,0x00,0x80,0x09,0x00,0x0A,'I','z','q','u','i','e','r','d','a',0x00}
Rotate Right 90º	{0x0C,0x00,0x80,0x09,0x00,0x08,'R','o','t','a','r','_','D',0x00}
Rotate Left 90º	{0x0C,0x00,0x80,0x09,0x00,0x08,'R','o','t','a','r','_','I',0x00}
Stop	{0x09,0x00,0x80,0x09,0x00,0x05,'S','t','o','p',0x00}

The commands shown in the table above are sent to the robot only if the recognition system classifies the gesture performed as one of the 7 intended gestures. The robot receives the command and extracts only the message that is contained within it. According to the information available in this message, the robot initiates the execution of the corresponding action.

Annex H presents the source codes of the software application designed to carry out the recognition and interaction stage with the LEGO NXT robot. Likewise, the program developed in the robot to detect the commands and execute the corresponding actions for each one is presented.

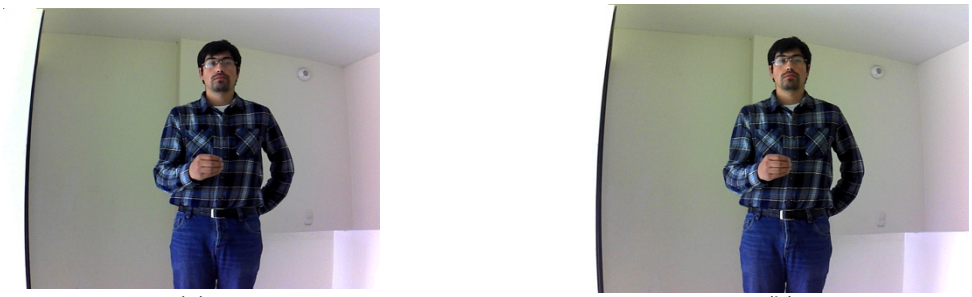
9 Results

This chapter presents the results obtained in each of the four stages that make up the gesture recognition system proposed in this work. Image processing algorithms are evaluated individually based on their implementation in software. In addition, the results obtained by the total operation of the system implemented in the DE1-SoC development board are presented.

Each of the developed hardware modules for image processing is functionally tested by describing a *test-bench* executed on MODELSIM.® Additionally, Altera®'s DSPBUILDER library is used to perform the test of the designed hardware, using images in the MATLAB® Simulink environment. The simulation results using Simulink are compared with the results obtained by running the algorithms implemented in MATLAB. For the testing of the algorithms in software, MATLAB R2013b is used on a computer with Windows 7 operating system and an Intel Core i7-4790 processor at a frequency of 3.6GHz.

9.1 Gesture Capture

The capture of the gestures is done by a stereo vision system made up of two ArduCam cameras which have an OV5642 sensor. Four hardware modules are used to acquire the images, which allow the stereo images to be configured, captured and synchronized with a resolution of 640x480 pixels and a capture speed of 30fps (see chapter 5). Figure 93 shows a pair of stereo images obtained using the designed capture system. In this figure, it can be seen that the captured images have a barrel distortion, which can be corrected using a subsequent rectification process.



(a) (b)

Figure 93. Stereo images obtained in the capture stage. (a) Left image. (b) Right image. Source: author.

Stereo matching requires images to be pre-rectified. However, the development of such a rectification process in this work may be too complicated due to two main reasons. First of all, extracting the test images from the FPGA to the PC, to obtain the calibration and rectification parameters in MATLAB, is too expensive. This is because it requires the design of additional hardware that allows the captured pixels to be sent to the computer through a common interface. On the other hand, the design and implementation of stereo image

rectification in hardware is too complex due to the type of arithmetic operations involved, as well as the strict restrictions associated with *stream-processing processing*. Therefore, in this work it was decided to perform the calculation of the disparity map without carrying out the rectification process in the capture stage.

To compensate for the lack of the rectification process, the stereo camera system is carefully constructed to maintain (manually) an acceptable horizontal alignment that allows the disparity map to be calculated well enough for the purposes of this work. It is worth mentioning that the results obtained, despite not implementing the rectification process, are quite good and although the images present the barrel distortion this does not significantly affect the operation of the system.

9.2 Pre-processing stage

The pre-processing stage is made up of four phases of image processing. Median filtering, stereo matching, segmentation, and morphological filtering. These phases are applied consecutively to the captured images in order to extract the region of the hand present in the scene, to later extract the trajectory of the centroid and carry out the recognition of the gesture made. This stage is explained in more detail in Chapter 6.

A b C d

Figure 94. Results of the filter applied to the *lenna* image. a) Original image. b) image with salt and pepper noise at 5%. c) image leaked in FPGA. d) image leaked in MATLAB. Source: author.

The median filter is applied to the captured images, in order to reduce the noise produced by the camera in the acquisition process. According to (Ahmed, Elatif, & Alser, 2015) the maximum performance of the median filter is obtained for a window of size 3X3 under different densities of salt- and pepper-type noise. Therefore, in this work, a median filter with a 3X3 window size was implemented in hardware.
As can be seen in the images in Figure 94, the result of the median filter in the hardware implementation Figure 94 (c) is equal to the software implementation Figure 94 (d). However, to quantitatively evaluate the degree of similarity of both results with respect to the original image Figure 94 (a), the PSNR (Signal to Peak Noise Ratio) is used.

PSNR is a widely used metric in image comparison and is defined as shown in equation (35). In this, it MSE represents the mean square error between the original or reference

image R and the image to be purchased Q and n is the number of bits used to represent a pixel.

$$MSE = \frac{1}{N \times M} \sum_{i=1}^{M} \sum_{j=1}^{N} (R(i,j) - \mathrm{Q(i,j)}^2$$
$$PSNR = 10 \log_{10} \left(\frac{2^n - 1}{MSE} \right) \tag{35}$$

Another important feature to buy between the two implementations of the median filter is the execution time required by said filter. To obtain the execution time of the algorithms in MATLAB, the *timeit* function is used. The hardware execution time is determined by the latency of the filter as shown in equation (36), where T_{PCLK} is the pixel period, M is the width of the image, and N the height of the image. For the hardware implementation, a pixel frequency of $100MHz$.

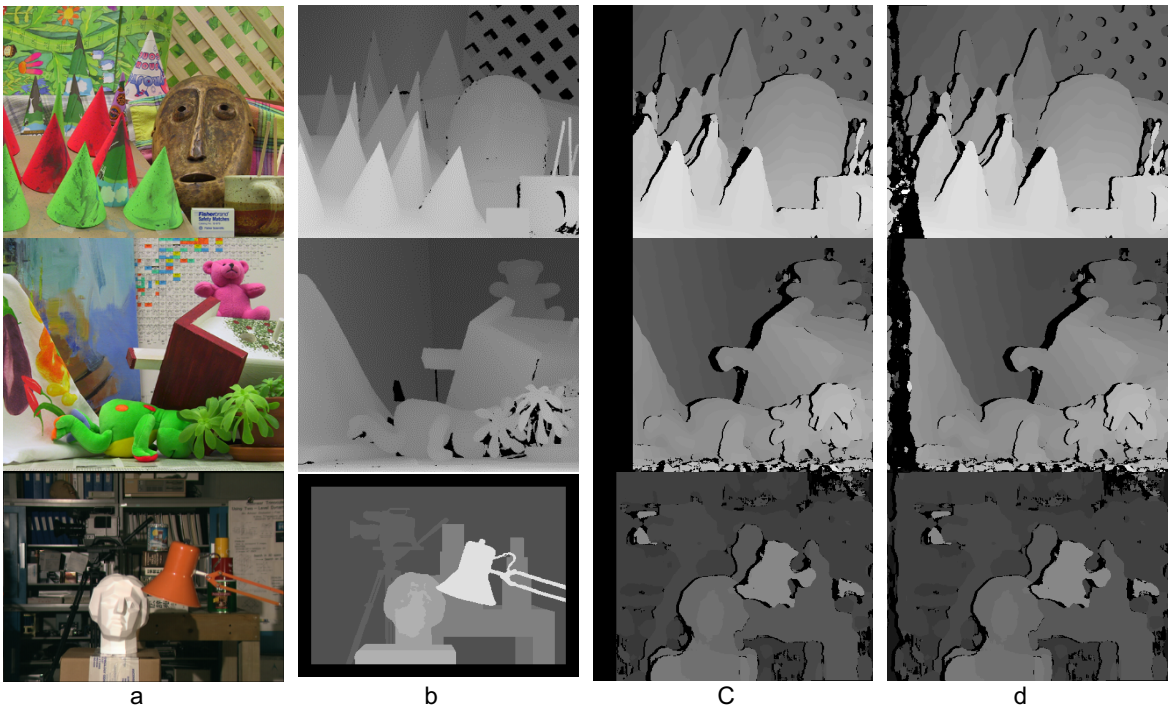
$$t_{FPGA} = T_{PCLK} \times (M + 11 + M \times N) \tag{36}$$

Table 16 presents the comparison results using PSNR and the execution time required by both implementations. As shown in the table, the PSRN is 36.1848 dB for the two images resulting from the filtering process, indicating that hardware deployment gets exactly the same results as running in MATLAB. However, the execution time on hardware is only 2.63 *ms* while on software it requires 4.2 *ms*. This implies that the implementation in hardware is approximately 1.6 times faster than its counterpart in software.

Table 16. Results of the median filter implemented in Software and Hardware.

Implementation	PSNR	IN
Hardware	36.1848 dB	2.63 *ms*
MATLAB	36.1848 dB	4.2 *ms*

After filtering the captured images, the second phase of gesture pre-processing involves calculating the disparity map of the captured stereo images. The method to calculate the disparity map in this work is made up of the 50% dispersed Census transform and the SHD. According to (Fife, 2011; Fife & Archibald, 2013), the best disparity map results are obtained when a Census window size of $W_c = 7$ and a SHD window size of $W_h = 13$. To evaluate the performance of the module developed in hardware, the stereo images *cones, Teddy* and *Tsukuba* obtained from the Middlebury database are used (Scharstein & Szeliski, 2002, 2003).



a b C d

Figure 95. Results of the execution of the stereo matching algorithm implemented in Software and Hardware. a) left image of the stereo pair used: *cones, teddy* and *tsukuba*. b) *ground-truth* c) disparity map in MATLAB. d) FPGA disparity map. Source: author.

Figure 95 presents the stereo images used to evaluate the behavior of the stereo matching algorithm implemented in hardware. For each of the stereo images used, the disparity map is calculated in both software and hardware. The images in column (a) correspond to the left image of the stereo pair. The images in column (b) correspond to the *ground-truth* of the disparity map of each of the images used, and columns (c) and (d) correspond to the disparity map obtained in software and hardware, respectively. In this figure it can be seen that the disparity maps calculated in software and hardware are similar, which indicates that the implemented hardware architecture perfectly complies with the specifications of the selected stereo matching algorithm.

Table 17 presents the percentage of matching pixels of the disparity maps calculated in this work with respect to the *ground-truth* disparity maps. In this table it can be seen that the software implementation has a low percentage of effectiveness compared to the results obtained in hardware. This is because the pixels on the left stripe of the disparity map obtained in software are taken as occlusion by the LRCC algorithm (see Figure 95(c)). However, when buying the two disparity maps, without taking into account these occlusion pixels, the result is that there is no difference, therefore, it was verified that the hardware implementation is perfectly equivalent to the software implementation. On the other hand, the disparity map obtained in this work has a percentage of effectiveness similar to the results obtained by other works in the literature. For example, (Fife, 2011) obtains as results: *cones* 87.76%, *teddy* 87.03% and *Tsukuba* 90.92%. This indicates that the performance of

the stereo matching algorithm implemented in hardware presents a fairly good result compared to similar methods reported in the literature.

Table 17. Percentage of correctly matched pixels of the stereo matching algorithm implemented in Software and Hardware.

Test Images	Matlab	Hardware
Cones	80.8071	88.0041
Teddy	80.6516	87.1227
Tsukuba	87.7729	90.7899

Table 18. Stereo correspondence runtime in software and hardware.

Test Images	Matlab	Hardware
Cones	375.6181 s	1.73 ms
Teddy	378.5892 s	1.73 ms
Tsukuba	127.6772 s	1.14 ms

Table 18 presents the execution time required by the stereo matching algorithm to calculate the disparity map. According to the results obtained, it is observed that the hardware implementation is extremely fast since it is in the order of *the ms* while the software implementation exceeds 100 seconds. The runtime results in software were calculated using the MATLAB *timeit* function . The hardware execution time is determined by the latency of the stereo matching module as shown in equation (37), being M the width of the image and d the disparity levels. To calculate the hardware execution time, the walls shown in Table 19 were used.

$$t_{FPGA} = T_{PCLK} \times \left[\frac{(M+1)(W_c + W_h + 2)}{2} - 2M + \log_2\left(\frac{W_c^2}{2}\right) + 2d + 11 + M \times N \right] \quad (37)$$

Table 19. Parameters used to calculate execution time in hardware.

Test Images	Size	M	Wc	Wh	d	Tpclk
Cones	450x375	450	7	13	64	10 ns
Teddy	450x375	450	7	13	64	10 ns
Tsukuba	384x288	384	7	13	32	10 ns

Several experimental tests were carried out with different sizes of Census window and SHD window. This was necessary because, as mentioned above, the implementation of the rectification process in the capture stage was not carried out in this work. According to the tests carried out, it was found that with values of $W_c = 7$, $W_h = 29$ and $d = 48$ a disparity map good enough for the development of this work is obtained. Figure 96 shows the resulting disparity map using the images captured by the recognition system with the aforementioned parameters.

A b c

Figure 96. A disparity map calculated for an image during gesture performance. a) Capture of the left image. b) Capture of the right image. c) Disparity map calculated by the developed system. Source: author.

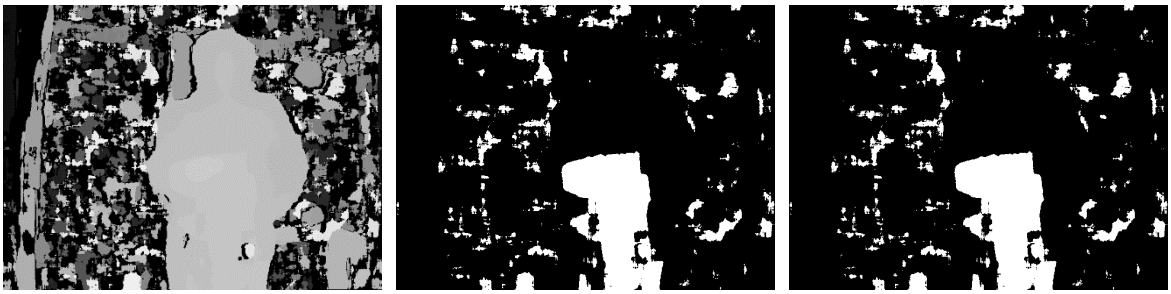
Once the disparity map is calculated, the next step in the gesture pre-processing stage is segmentation. To extract the region of the hand from the scene, two types of segmentation are used. Skin color segmentation and disparity map segmentation. The former allows you to extract the regions of the image with a similar color to the skin, while the latter allows you to extract only objects that are at a certain distance from the camera.

Figure 97 presents the results obtained from the skin color segmentation. Image (a) corresponds to the left image of the stereo pair captured by the system. Images (b) and (c) are the segmentation results obtained in software and hardware respectively. As can be seen, the result of segmentation in hardware is exactly the same as the result in software. Therefore, it can be stated that the hardware developed adequately complies with the design specifications set for this algorithm.

A B c

Figure 97. Segmentation by skin color. (a) Left image of the captured stereo pair. (b) Skin Color Segmentation in MATLAB. (c) Segmentation by skin color in FPGAs. Source: author.

Figure 98 shows the result obtained from the segmentation of the disparity map. Figure 98 (a) corresponds to the left image of the stereo pair captured by the system, while the images in Figures 98 (b) and (c) correspond to the result of the segmentation of the disparity map obtained in software and hardware respectively. Like skin color targeting, disparity map targeting in hardware and software are the same. Therefore, the developed hardware corresponds to the specified segmentation algorithm.



A B c

Figure 98. Disparity map segmentation. (a) disparity map calculated from the stereo images captured. (b) disparity map segmentation in MATLAB. (c) Segmentation of the disparity map in FPGAs. Source: author.

As observed in the previous images, the segmentation into hardware and software presents the same results. However, the hardware execution time is much less than that required for software execution. Table 20 presents the runtime results. In this table, it is observed that in software the segmentation of the disparity map has a shorter execution time with respect to the segmentation by skin color. On the other hand, the hardware implementation of the segmentations takes exactly the same time to execute. To calculate the execution time in software, the MATLAB *timeit* function is used . Hardware execution time is calculated using equation (38), where M and N are the width and height of the image and T_{PCLK} the pixel period of the processing system.

$$t_{FPGA} = T_{PCLK}\left[M \times N + 2\right] \tag{38}$$

Table 20. Software and hardware disparity map segmentation runtime.

Segmentation	Matlab	Hardware
Skin Color	41.7 ms	3.07 ms
Disparity Map	4.8 ms	3.07 ms

The final result of the segmentation process is obtained by intersecting the segmentations calculated by skin color and the disparity map. This allows you to remove those objects or regions of the image that can affect the proper functioning of the system. However, the segmentation result usually has noise which must be filtered out to get the region of the hand cleaner. To carry out this filtering process, the morphological operation of opening is applied.

To B C
Figure 99. Morphological opening operation. (a) Segmented image. (b) morphological opening operation in MATLAB. (c) morphological operation of opening in FPGAs. Source: author.

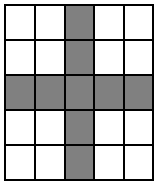


Figure 100. Structuring element used in the morphological operation of opening. Source: author.

Figure 99 a) presents the result of the morphological filtering process applied to the segmented image. As you can see, the result obtained from the segmentation process is very good. However, there is a considerable amount of noise that needs to be eliminated. For this, several experimental tests were carried out with different sizes of processing windows, obtaining the most favorable result for a window of size 11x11 using the structuring element of Figure 100. The images shown in Figure 99 (b) and Figure 99 (c) show the result obtained from the aperture morphological filtering with the window size selected for deployment in software and hardware, respectively. In these figures it can be seen that the noise is eliminated in its entirety although the shape of the hand tends to deform increasing its area. This deformation is not critical since in this work no characteristics based on appearance are used, therefore the result of the morphological operation is acceptable.

Additionally, in Figure 99 b) and c) it can be seen that the result of the morphological opening operation obtained in hardware is equal to the result of the operation in software. However, the hardware execution time is much less than the software execution as shown in Table 21. To calculate the execution time in software, the MATLAB *timeit* function is used . Hardware execution time is calculated using equation (39), where M and N are the width and height of the image, W_o the size of the filter window, and T_{PCLK} the pixel period of the processing system.

$$t_{FPGA} = T_{PCLK}\left[M \times N + 2 \times \left(\frac{(W_o+1)(M+1)}{2} - M + \log_2\left(W_o^{\,2}\right) + 2\right)\right] \tag{39}$$

Table 21. Execution time of the morphological opening operation in software and hardware.

Matlab	Hardware
1.2633 s	42.050 ms

9.3 Feature Extraction

The feature extraction stage is composed of four phases: centroid calculation, gesture path smoothing, gesture detection, and orientation extraction and quantization; Each of these

phases is explained in Chapter 7. Figures 116 to 122 show the result of the extraction of characteristics obtained by the recognition system during the performance of each gesture.

As explained in section 7.4, in this paper the orientation of two consecutive centroid values is used as the only characteristic that describes each gesture. Therefore, in each processed image a discrete symbol (1-8) is obtained depending on the trajectory described by the hand at that instant of time. The number of symbols extracted for a gesture depends on its duration, which is represented by the number of *frames* processed.

Figure 101 shows the characteristics obtained for the "Forward" gesture. In this figure it can be seen that the gesture lasts approximately 45 *frames* and the predominant symbols are (7, 6, 2 and 3).

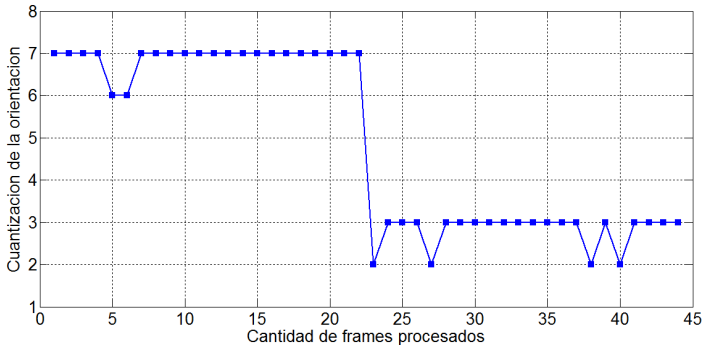


Figure 101. Sequence of discrete symbols representing the "Go ahead" gesture. Source: author.

Figure 102 presents the characteristics obtained for the "Back" gesture. In this figure it can be seen that the gesture lasts approximately 40 *frames* and the predominant symbols are (2, 1, 3, 8, 7 and 6).

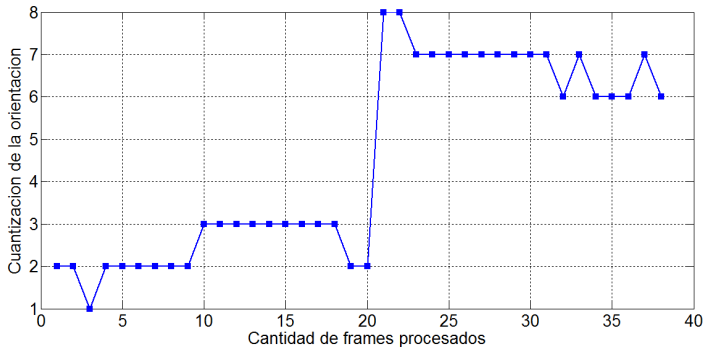


Figure 102. Sequence of discrete symbols representing the "Back" gesture. Source: author.

Figure 103 presents the characteristics obtained for the "Right" gesture. In this figure it can be seen that the gesture lasts approximately 45 *frames* and the predominant symbols are (5, 6 and 1).

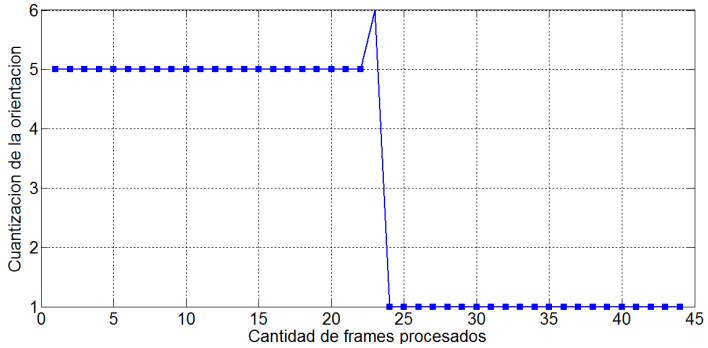


Figure 103. Sequence of discrete symbols representing the "Right" gesture. Source: author.

Figure 104 shows the characteristics obtained for the "Left" gesture. In this figure you can see that the gesture lasts approximately 50 *frames* and the predominant symbols are (1, 7, 6, 5 and 4).

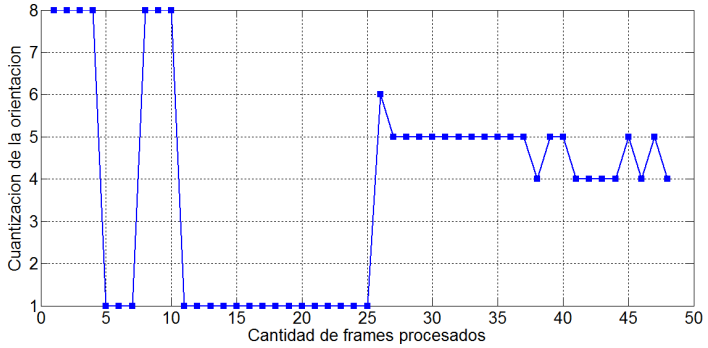


Figure 104. Sequence of discrete symbols representing the "Left" gesture. Source: author.

Figure 105 shows the characteristics obtained for the "Rotate Right 90º" gesture. In this figure it can be seen that the gesture has an approximate duration of 60 *frames*. This gesture uses all symbols (1-8).

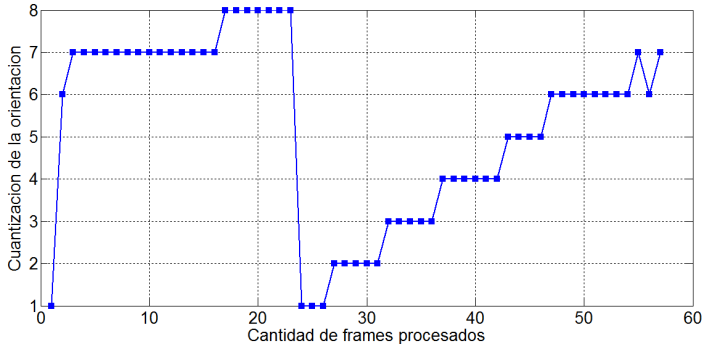


Figure 105. Sequence of discrete symbols representing the "Rotate Right 90°" gesture. Source: author.

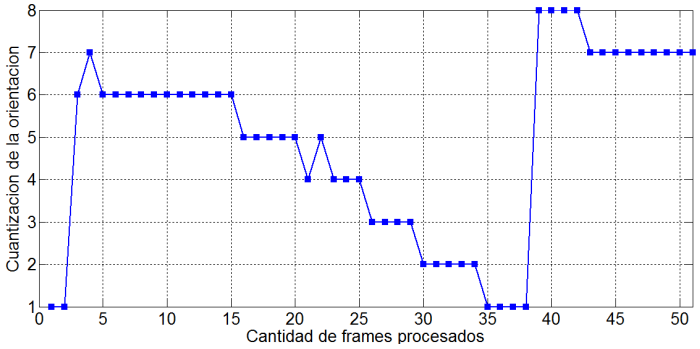


Figure 106. Sequence of discrete symbols representing the "Rotate Left 90°" gesture. Source: author.

Figure 106 shows the characteristics obtained for the "Rotate Left 90º" gesture. In this figure it can be seen that the gesture has an approximate duration of 51 *frames*. This gesture, like the previous gesture, uses all the discrete symbols for its description, however the order in which the symbols appear as the gesture is performed is different.

Figure 107 shows the characteristics obtained for the "Stop" gesture. In this figure it can be seen that the gesture lasts approximately 80 *frames*. This gesture is the one that requires the most *frames* to be represented and requires all symbols except 3 for its representation.

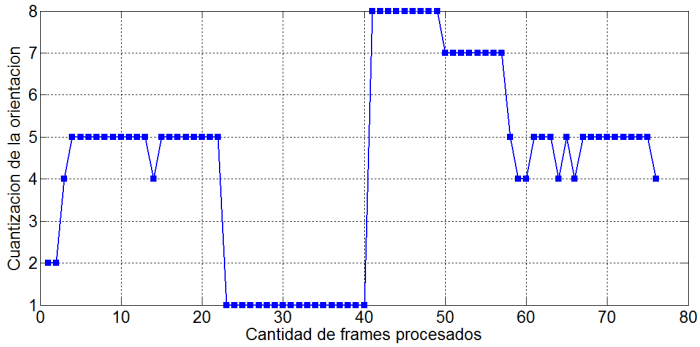


Figure 107. A sequence of discrete symbols representing the "Stop" gesture. Source: author.

According to the characteristics obtained for each of the gestures, it can be observed that there is a significant difference in the performance of each of them, which allows the classifier based on HMMs to make the recognition process more easily.

9.4 Gesture Recognition

As explained in Chapter 8, k-fold cross-validation allows the number of hidden states of the HMMs used in the recognition stage to be properly selected. To carry out the cross-validation, a database was built consisting of 1260 samples corresponding to the seven gestures used in this work. The samples in the database were obtained from three different people, taking 60 samples per person for each of the gestures.

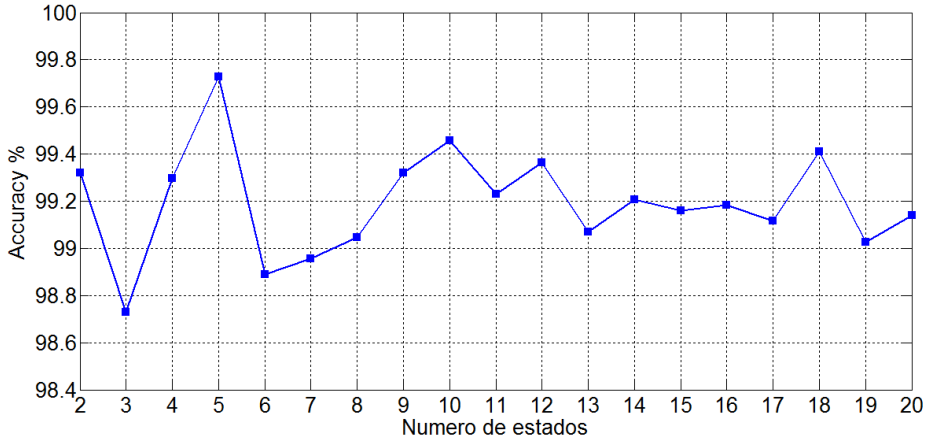


Figure 108. 10-fold cross-validation results to determine the accuracy of the HMM-based classifier with different number of hidden states. Source: author.

Figure 108 presents the results of the 10-fold cross-validation performed on the HMM-based classifier. The validation was carried out for different sizes of the classifier using from 2 to 20 hidden states. As can be seen in this figure, the classifier composed of 5 states presents the maximum performance, obtaining 99.7% accuracy. On the other hand, the classifier has the lowest performance with 98.7% when 3 hidden states are used. Taking into account the results obtained through the selected cross-validation method, the implementation of the recognition system based on HMMs is carried out using 5 hidden states.

9.5 Implementation of the Gesture Recognition System in FPGA-SoC

The gesture recognition system was implemented on the DE1-SOC development board, which has a Cyclone V FPGA-SoC device. Figure 109 shows the implementation of the gesture recognition system. As you can see, the cameras and the RN42 Bluetooth module are connected via the GPIO ports of the DE1-SoC development board. The VGA port is used to debug the operation of the system by displaying the processed images on a monitor. In addition, pushbuttons, switches, and LEDs are used to interact with the recognition system as shown in Table 22.

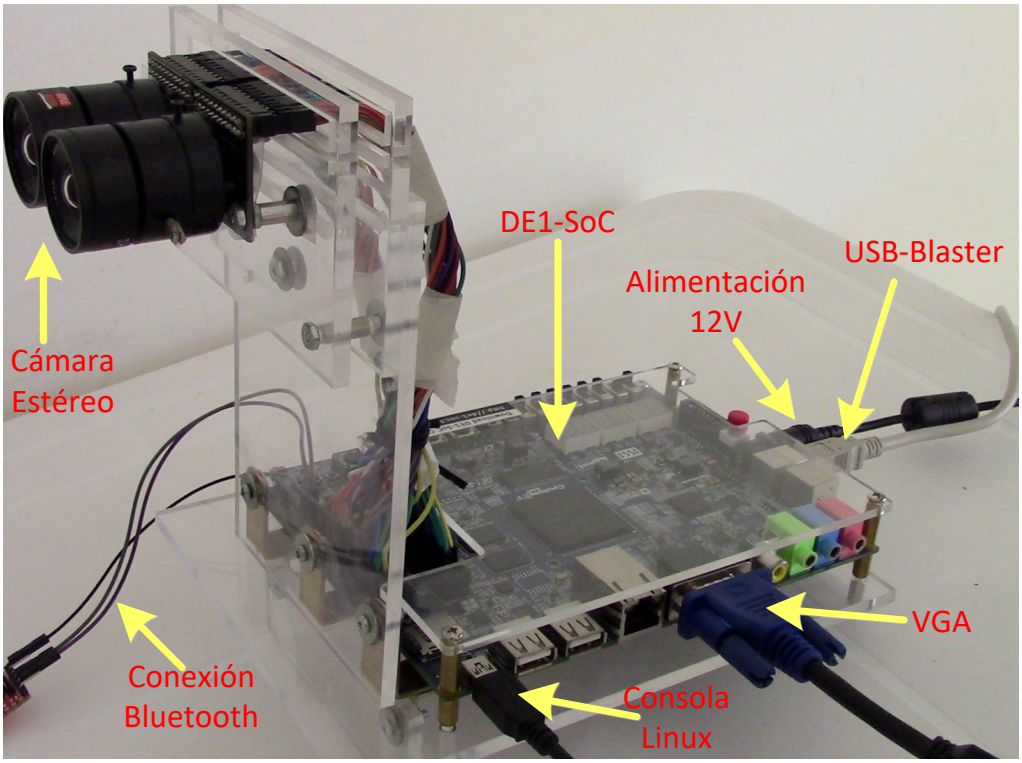


Figure 109. Implementation of the gesture recognition system. Source: author.

Table 22. Description of the functionality of the buttons, switches and LEDs of the De1-SoC card in the gesture recognition system.

Name	Description
KEY[0]	Reset of the developed image processing system
KEY[1]	Camera settings to stop automatic light exposure adjustment
KEY[2]	Pause image capture
KEY[3]	Start image capture
SW[0]	Enable image capture
SW[9:7]	Selection of the image to be displayed by VGA port. "000" Right image of the stereo system "001" Left image of the stereo system "010" Disparity Map "011" Skin Color Segmentation "100" Disparity Map Targeting "101" Segmentation after applying the aperture morphological operation
LEDR[4]	Main gesture detection.

The designed gesture recognition system is implemented on the DE1-SoC development board using the synthesis software of Altera Quartus II v 14.1. Table 23 presents the results of the synthesis process. As can be seen, the stereo correspondence module consumes about 37% of the logical resources and 16% of the memory bits available on the device. The capture stage consumes 35% of the memory bits. This is due to the Sync Module, which must have enough memory to synchronize the stereo images during capture. The other hardware modules do not exceed 10% of the FPGA portion's resource utilization. In total,

the developed system makes use of about 60% of the available resources, plus an additional 10% used by the debugging hardware, which is not part of the recognition system.

Table 23. FPGA-SoC device resources used by the gesture recognition system.

	Alms		Records		Memory Bits		HPS	
	Quantity	%	Quantity	%	Quantity	%	Quantity	%
Available Resources	32070	-	64140	-	4065280	-	1	-
Capture Stage	1010	3.1	1371	2.2	1048576	25.8	0	0
Median Filter	2298	7.1	3876	6.1	61056	1.5	0	0
Stereo Matching	11977	37.2	30897	48.2	661556	16.3	0	0
Segmentation	49	0.2	27	0.04	0	0	0	0
Morphological opening operation	1350	4.1	3259	5.4	101888	25	0	0
Centroid Calculation	1013	3.1	921	1.4	0	0	0	0
Moving Average Filter	65	0.2	194	0.3	0	0	0	0
Gesture detection	301	1	562	0.8	0	0	0	0
Feature extraction and gesture recognition	911	2.8	1409	2.1	14336	0.3	1	100%
Debugging Hardware	2967	9.2	3303	5.1	454581	11.2	0	0
Total resources consumed	21941	68	45819	71.4	2341913	57.6	1	100%

The gesture recognition system was developed using a parametric description in VHDL language. This description allows you to select any size of images to be processed as well as other operating characteristics of the system. It is important to note that the parametric description is reflected in the synthesis process, so there are limitations on the amount of device resources used according to the required configuration. The system processes the images at the same time as they are captured by the camera.

The total processing time of a complete image is determined by equation (40) using the following parameters: VGA images (640x480), Census window $W_c = 7$, Hamming window, $W_h = 29$ disparity levels $d = 48$, morphological operation window, $W_o = 11$ and pixel frequency of $100MHz$. According to the system's operating specifications, a total processing time of 3.252940 ms is obtained, which implies that the system is able to process at a speed of approximately 307 fps.

$$
\begin{aligned}
L_{me} &= M + 11 \\
L_{sm} &= \frac{(M+1)(W_c + W_h + 2)}{2} - 2M + \log_2\left(\frac{W_c^2}{2}\right) + 2d + 11 \\
L_{op} &= 2 \times \left[\frac{(M+1)(W_o + 1)}{2} - M + \log_2\left(W_o^2\right) + 2 \right] \\
L_{se} &= 2 \\
t_{total} &= T_{PCLK} \times \left[M \times N + L_{me} + L_{sm} + L_{op} + L_{se} \right]
\end{aligned}
\tag{40}
$$

Although the processing speed that the developed system can reach is quite high, in this work a processing speed of 30fps was used, mainly due to the limitations in the capture speed of the cameras used in this work.

The complete gesture recognition system implemented in the FPGA-SoC device was put to the test by controlling the actions of a LEGO NXT robot. The test was carried out by executing each gesture 100 times to evaluate the behavior of the system with gestures different from those used in the training and validation process used. The result of this test is presented in the confusion matrix in Table 24.

According to the confusion matrix obtained in the evaluation process, it can be determined that the gesture recognition system designed and implemented in the FPGA-SoC device has a total recognition rate of 99.7%. The system presents excellent results with recognition rates above 99%. Additionally, it is observed that the recognition system tends to confuse the "Right" gesture with the "Stop" gesture and the "Rotate Right 90º" gesture with the "Stop" gesture. However , this confusion doesn't affect the overall performance of the recognition system too much.

Table 24. Confusion matrix of the reconnaissance system implemented.

	Go ahead	**Behind**	**Right**	**Left**	**Rotate Right 90º**	**Rotate Left 90º**	**Stop**
Go ahead	100	0	0	0	0	0	0
Behind	0	100	0	0	0	0	0
Right	0	0	99	0	0	0	1
Left	0	0	0	100	0	0	0
Rotate Right 90º	0	0	0	0	99	0	1
Rotate Left 90º	0	0	0	0	0	100	0
Stop	0	0	0	0	0	0	100

10 Conclusions and Future Work

In this work we present the development of a hand gesture recognition system, implemented in an FPGA-SoC device and using stereoscopic vision, with application in the Human-Robot interaction. The developed recognition system is composed of four main stages: capture, pre-processing, feature extraction and recognition of the gesture performed. The system is implemented on a single chip as follows. The first three stages of the system are implemented in the FPGA zone of the device, while the reconnaissance stage is implemented in the HPS zone using an application developed for Linux. The system is able to recognize seven hand gestures and control the actions of a LEGO NXT robot according to the identified gesture. The gestures used in this work are performed using the right hand only. The recognition system identifies gestures with a recognition rate of more than 99%.

The hardware design methodology of the image processing algorithms of the gesture recognition system, based on the development of these algorithms in software, is extremely useful since it allows the designer to know in depth their behavior and thus facilitate their subsequent mapping in hardware. In addition, the software implementation of the algorithms used in this work is taken as a reference model for the verification of results. This allows to effectively evaluate whether the results obtained by the developed hardware correspond to the expected results.

The capture and processing of stereo images, carried out by the first three stages of the recognition system, have a high processing power, managing to extract the region of the hand, for each of the captured images, in a time of less than 3.5 ms. This allows real-time image processing with a processing speed of more than 300fps. The hardware implementation of the image processing algorithms used in this work has a much higher performance than the software execution of these algorithms.

The stereo matching algorithm used in this work presents excellent results for the calculation of the disparity map, taking into account that the stereo images are not rectified in the capture stage. Despite the high computational complexity required by the stereo matching algorithm, it is important to note that the calculation of the disparity map is performed in real time, during the capture of the images from the cameras used.

The *k-fold cross-validation* process is quite useful for evaluating the performance of different classifiers or different parameters of the same classifier. In this work, HMMs were used as classifiers of the gesture recognition system. Cross-validation *allows k_fold* to properly select the number of hidden states required by the classifier. This training and validation process was carried out *off-line* using MATLAB software. The training of the HMMs is carried out using the Waum-Welch algorithm, available in the HMM toolbox for MATLAB developed

by Kevin Murphy. According to the results obtained, it is established that the classifier obtains its maximum performance when 5 hidden states are used.

The HPS used to implement the recognition stage of the developed system has advantages compared to the use of embedded processors or *Soft-Cores*. First, the HPS system is a physical resource of the FPGA-SoC device, which means that its use does not require the logical resources of the FPGA. This allows the user to design high-performance hardware in the reconfigurable area of the device, without allocating logical elements for the processor synthesis required for software execution. On the other hand, the HPS has two processors that allow high-performance multi-threaded processing. In addition, HPS supports running a Linux operating system, this allows the user to develop custom applications faster than the development of *bare-metal* applications required on most *soft-core* processors. The only disadvantage found in the development of this work is related to the documentation of the HPS system. This documentation does not clearly explain the operation of the bridges between the HPS zone and the FPGA zone. This makes it a bit difficult to connect the processor with custom modules designed on the FPGA.

The use of FPGA devices in gesture recognition applications is an excellent alternative to conventional systems and technologies that perform similar tasks. The high performance that can be obtained from FPGAs and their low power consumption makes them an ideal technology for the development and continuous improvement of gesture recognition systems in a wide spectrum of Human-Computer interaction applications.

10.1 Future Work

The gesture recognition system presented in this work is oriented to the Human-Robot interaction using hand gestures as the main modality of interaction. Although the developed system works quite well, its interaction with the robot is a bit limited. This limitation is mainly due to the restriction used in the performance of gestures, since the correct identification of the three phases that make up a gesture is a fairly complex task to perform in real time and even more so if this involves the development of a hardware architecture that takes care of this task. For this reason, the development or evaluation of algorithms that allow gesture recognition to be carried out taking into account an interaction more naturally is proposed.

The gesture recognition system developed has great advantages over conventional systems, since it allows processing with high performance and low energy consumption thanks to the parallelism typical of FPGAs. This provides the opportunity to develop systems, not only for Human-Robot interaction, but in other fields of application such as applications for blind people, development of interaction systems for people with hearing disabilities, entertainment systems, home automation, among others.

In this work, a stereo matching algorithm was implemented in hardware using the Census transform and Hamming's Sum of Distances. This algorithm allows the disparity map to be

calculated in stereo images. However, the rectification of the captured images was not carried out as suggested in the literature. Although the results obtained without the rectification process are acceptable, it is suggested as future work to develop an architecture that allows this rectification process to be carried out in order to obtain a more accurate disparity map for its efficient application in other scenarios other than gesture recognition.

Finally, it is proposed as future work, to improve the hardware filtering process of the segmented image, since the morphological filtering used tends to modify the shape of the hand too much and does not allow to obtain significant information about the posture performed. This would allow not only the trajectory of the gesture but also the configuration of the hand to be used in order to recognize more complex gestures. In addition, it is proposed to use other types of characteristics other than orientation and to evaluate the possibility of implementing these algorithms in hardware in an appropriate way.

Bibliografía

Aggarwal, J. K., & Ryoo, M. S. (2011). Human Activity Analysis: A Review. *ACM Comput. Surv.*, *43*(3), 16:1--16:43. http://doi.org/10.1145/1922649.1922653

Ahmed, E. S. A., Elatif, R. E. A., & Alser, Z. T. (2015). Median filter performance based on different window sizes for salt and pepper noise removal in gray and RGB images. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, *8*(10), 343–352.

Al-Rousan, M., Assaleh, K., & Tala'a, A. (2009). Video-based signer-independent Arabic sign language recognition using hidden Markov models. *Applied Soft Computing Journal*, *9*(3), 990–999. http://doi.org/10.1016/j.asoc.2009.01.002

Althoff, F., Lindl, R., & Walchshäusl, L. (2005). Robust multimodal hand- and head gesture recognition for controlling automotive infotainment systems. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-28844476135&partnerID=40&md5=b97f7e1a69ab0ec57362ebc1117d0e2f

Ambrosch, K., Kubinger, W., Humenberger, M., & Steininger, A. (2009). Flexible hardware-based stereo matching. *EURASIP Journal on Embedded Systems*, *2008*(1), 1–12.

Ambrosch, K., Zinner, C., & Kubinger, W. (2009). Algorithmic Considerations for Real-Time Stereo Vision Applications. In *MVA* (pp. 231–234).

Bailey, D. G. (2011). *Design for embedded image processing on FPGAs*. John Wiley & Sons.

Bansal, M., Saxena, S., Desale, D., & Jadhav, D. (2011). Dynamic gesture recognition using hidden markov model in static background. *IJCSI*.

Banz, C., Hesselbarth, S., Flatt, H., Blume, H., & Pirsch, P. (2010). Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation. *Embedded Computer Systems (SAMOS), 2010 International Conference on*. http://doi.org/10.1109/ICSAMOS.2010.5642077

Bellmore, C., Ptucha, R., & Savakis, A. (2011). Interactive display using depth and RGB sensors for face and gesture control. *Image Processing Workshop (WNYIPW), 2011 IEEE Western New York*. http://doi.org/10.1109/WNYIPW.2011.6122883

Bendaoudi, H., & Khouas, A. (2013). Stereo vision IP design for FPGA implementation of obstacle detection system. *Systems, Signal Processing and Their Applications (WoSSPA), 2013 8th International Workshop on*. http://doi.org/10.1109/WoSSPA.2013.6602352

Benko, H., & Wilson, A. (2009). *DepthTouch: Using Depth-Sensing Camera to Enable Freehand Interactions On and Above the Interactive Surface*. Microsoft. Retrieved from https://www.microsoft.com/en-us/research/publication/depthtouch-using-depth-sensing-camera-to-enable-freehand-interactions-on-and-above-the-interactive-surface/

Bergh, M. Van den, Carton, D., Nijs, R. De, Mitsou, N., Landsiedel, C., Kuehnlenz, K., … Buss, M. (2011). Real-time 3D hand gesture interaction with a robot for understanding directions from humans. *2011 RO-MAN*. http://doi.org/10.1109/ROMAN.2011.6005195

Bergh, M. Van den, & Gool, L. Van. (2011). Combining RGB and ToF cameras for real-time 3D hand gesture interaction. *Applications of Computer Vision (WACV), 2011 IEEE Workshop on*. http://doi.org/10.1109/WACV.2011.5711485

Berkeley Design Technology, I. (2005). Inside DSP on Tools: FPGA Tools Bridge Gap Between Algorithm and Implementation | www.bdti.com. Retrieved August 26, 2016, from http://www.bdti.com/InsideDSP/2005/06/15/Idsp3

Binh, N. D., Shuichi, E., & Ejima, T. (2005). Real-Time Hand Tracking and Gesture Recognition System. In *Proceedings of International Conference on Graphics, Vision and Image Processing (GVIP-05* (pp. 362–368).

Biswas, K. K., & Basu, S. K. (2011). Gesture recognition using Microsoft Kinect®. *Automation, Robotics and Applications (ICARA), 2011 5th International Conference on*. http://doi.org/10.1109/ICARA.2011.6144864

Bonato, V., Sanches, A. K., Fernandes, M. M., Cardoso, J. M. P., do Valle Simões, E., & Marques, E. (2004). A Real Time Gesture Recognition System for Mobile Robots. In *ICINCO (2)* (pp. 207–214).

Breuer, P., Eckes, C., & Müller, S. (2007). Hand gesture recognition with a novel IR time-of-flight range camera-A pilot study. *3rd International Conference, MIRAGE 2007: Computer Vision/Computer Graphics Collaboration Techniques*. Universität Siegen, Germany.

Brown, M. Z., Burschka, D., & Hager, G. D. (2003). Advances in computational stereo. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. http://doi.org/10.1109/TPAMI.2003.1217603

Chen, C. P., Chen, Y. T., Lee, P. H., Tsai, Y. P., & Lei, S. (2011). Real-time hand tracking on depth images. *Visual Communications and Image Processing (VCIP), 2011 IEEE*. http://doi.org/10.1109/VCIP.2011.6115983

Chen, F.-S., Fu, C.-M., & Huang, C.-L. (2003). Hand gesture recognition using a real-time tracking method and hidden Markov models. *Image and Vision Computing*, *21*(8), 745–758. http://doi.org/10.1016/S0262-8856(03)00070-2

Chen, L., Wei, H., & Ferryman, J. (2013). A survey of human motion analysis using depth imagery. *Pattern Recognition Letters*, *34*(15), 1995–2006. http://doi.org/10.1016/j.patrec.2013.02.006

Chen, Z.-H., Kim, J.-T., Liang, J., Zhang, J., & Yuan, Y.-B. (2014). Real-time hand gesture recognition using finger segmentation. *Scientific World Journal*, *2014*. http://doi.org/10.1155/2014/267872

Cho, P. C., Li, C. T., & Chen, W. H. (2012). Implementation of low-cost vision-based gesture recognition systems based on FPGA approach (pp. 329–332). Taichung. http://doi.org/10.1109/is3c.2012.90

Colodro, C., Toledo, J., Martínez, J. J., Garrigós, J., & Ferrández, J. M. (2012). Evaluación de algoritmos de correspondencia estereoscópica y su implementación en FPGA. *Jornadas de Computación Reconfigurable Y Aplicaciones (JCRA)*, 88–93.

Coogan, T., Awad, G., Han, J., & Sutherland, A. (2006). Real time hand gesture recognition including hand segmentation and tracking. *2nd International Symposium on Visual Computing, ISVC 2006*. Dublin City University, Dublin 9, Ireland. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-

33845423550&partnerID=40&md5=ff083608b603eade6aa95ad1e3d783b6

Ding, J., Du, X., Wang, X., & Liu, J. (2010). Improved real-time correlation-based FPGA stereo vision system. *Mechatronics and Automation (ICMA), 2010 International Conference on*. http://doi.org/10.1109/ICMA.2010.5588008

Dunn, P., & Corke, P. (1997). Real-time stereopsis using FPGAs. In W. Luk, P. Y. K. Cheung, & M. Glesner (Eds.), *Field-Programmable Logic and Applications: 7th International Workshop, FPL '97 London, UK, September 1--3, 1997 Proceedings* (pp. 400–409). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/3-540-63465-7_245

Elmezain, M., & Al-Hamadi, A. (2012). LDCRFs-based hand gesture recognition. In *2012 IEEE International Conference on Systems, Man, and Cybernetics (SMC)* (pp. 2670–2675). Seoul. http://doi.org/10.1109/ICSMC.2012.6378150

Elmezain, M., Al-Hamadi, A., Appenrodt, J., & Michaelis, B. (2008). A Hidden Markov Model-based continuous gesture recognition system for hand motion trajectory. *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. http://doi.org/10.1109/ICPR.2008.4761080

Elmezain, M., Al-Hamadi, A., Appenrodt, J., & Michaelis, B. (2009). A hidden markov model-based isolated and meaningful hand gesture recognition. *International Journal of Electrical, Computer, and Systems Engineering*, *3*(3), 156–163.

Elmezain, M., Al-Hamadi, A., & Michaelis, B. (2008). Real-time capable system for hand gesture recognition using hidden Markov models in stereo color image sequences. In *16th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision 2008, WSCG'2008* (pp. 65–72). Institute for Electronics, Signal Processing and Communications (IESK), Otto-von-Guericke-University, Magdeburg, Germany. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-63249103687&partnerID=40&md5=300265e495117cb1f03a38fcfb67970d

Elmezain, M., Al-Hamadi, A., Pathan, S. S., & Michaelis, B. (2009). Spatio-temporal feature extraction-based hand gesture recognition for isolated american sign language and arabic numbers (pp. 264–269). Salzburg. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-70450255139&partnerID=40&md5=6d4b8b3cf6216720c935c6ec923790b7

Elmezain, M. O. S. M. (2010). *Hand Gesture Spotting and Recognition Using HMMs and CRFs in Color Image Sequences*. Otto-von-Guericke-Universitat Magdeburg.

Fahn, C.-S., & Chu, K.-Y. (2011). Hidden-Markov-model-based hand gesture recognition techniques used for a human-robot interaction system. *14th International Conference on Human-Computer Interaction, HCI International 2011*. Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei 10607, Taiwan. http://doi.org/10.1007/978-3-642-21605-3_28

Faugeras, O., Hotz, B., Mathieu, H., Viéville, T., Zhang, Z., Fua, P., … Proy, C. (1993). *Real time correlation-based stereo: algorithm, implementations and applications*. INRIA. Retrieved from https://infoscience.epfl.ch/record/176855/files/RR-2013.pdf

Fife, W. S. (2011). *Improved Stereo Vision Methods for FPGA-Based Computing Platforms*. Brigham Young University. Retrieved from http://scholarsarchive.byu.edu/etd/2745

Fife, W. S., & Archibald, J. K. (2013). Improved Census Transforms for Resource-Optimized

Stereo Vision. *IEEE Transactions on Circuits and Systems for Video Technology*. http://doi.org/10.1109/TCSVT.2012.2203197

Freescale Semiconductor. (2013). OV5642.c. Boston, MA, USA. Retrieved from https://github.com/boundarydevices/linux-imx6/blob/boundary-imx_3.0.35_4.1.0/drivers/media/platform/mxc/capture/ov5642.c

Fujimura, K., & Liu, X. (2006). Sign recognition using depth image streams. *7th International Conference on Automatic Face and Gesture Recognition (FGR06)*. http://doi.org/10.1109/FGR.2006.101

Ghotkar, A. S., & Kharate, G. K. (2012). Hand segmentation techniques to hand gesture recognition for natural human computer interaction. *International Journal of Human Computer Interaction (IJHCI)*, *3*(1), 15.

Gonzalez, R. C., & Woods, R. E. (2002). *Digital image processing*. *Prentice Hall* (2d ed.). Prentice Hall.

Grammalidis, N., Goussis, G., Troufakos, G., & Strintzis, M. G. (2001). 3-D human body tracking from depth images using analysis by synthesis. *Image Processing, 2001. Proceedings. 2001 International Conference on*. http://doi.org/10.1109/ICIP.2001.958455

Gregg Hawkes. (2001). Application Note: MicroBlaze and Multimedia Development Board Digital Component Video Conversion 4:2:2 to 4:4:4. Xilinx.

Greisen, P., Heinzle, S., Gross, M., & Burg, A. P. (2011). An FPGA-based processing pipeline for high-definition stereo video. *EURASIP Journal on Image and Video Processing*, *2011*(1), 1–13. http://doi.org/10.1186/1687-5281-2011-18

Gribbon, K. T., Bailey, D. G., & Johnston, C. T. (2005). Design Patterns for Image Processing Algorithm Development on FPGAs. *TENCON 2005 - 2005 IEEE Region 10 Conference*. http://doi.org/10.1109/TENCON.2005.301109

Gribbon, K. T., Bailey, D. G., & Johnston, C. T. (2006). Using design patterns to overcome image processing constraints on FPGAs. *Third IEEE International Workshop on Electronic Design, Test and Applications (DELTA'06)*. http://doi.org/10.1109/DELTA.2006.93

Gudis, E., Wal, G. van der, Kuthirummal, S., & Chai, S. (2012). Multi-Resolution Real-Time Dense Stereo Vision Processing in FPGA. *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*. http://doi.org/10.1109/FCCM.2012.15

Guerrero-Balaguera, J. D., & Perez-Holguin, W. J. (2015). FPGA-based translation system from colombian sign language to text. *DYNA*, *82*, 172–181. Retrieved from http://www.scielo.org.co/scielo.php?script=sci_arttext&pid=S0012-73532015000100022&nrm=iso

Guomundsson, S. A., Larsen, R., Aanaes, H., Pardas, M., & Casas, J. R. (2008). TOF imaging in Smart room environments towards improved people tracking. *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*. http://doi.org/10.1109/CVPRW.2008.4563154

Habermann, A. N. (1972). *Parallel neighbor-sort (or the glory of the induction principle)*. *CMU Computer Science Report (available as Technical report AD-759 248, National Technical Information Service, US Department of Commerce, 5285 Port Royal Rd*

Sprigfield VA 22151). Pittsburgh, Pa. 15213.

Hartley, R. ~I., & Zisserman, A. (2004). *Multiple View Geometry in Computer Vision* (Second). Cambridge University Press, ISBN: 0521540518.

Humenberger, M., Zinner, C., Weber, M., Kubinger, W., & Vincze, M. (2010). A fast stereo matching algorithm suitable for embedded real-time systems. In *Computer Vision and Image Understanding* (Vol. 114, pp. 1180–1202). http://doi.org/10.1016/j.cviu.2010.03.012

Ibarra-Manzano, M. A., Almanza-Ojeda, D. L., Devy, M., Boizard, J. L., & Fourniols, J. Y. (2009). Stereo Vision Algorithm Implementation in FPGA Using Census Transform for Effective Resource Optimization. *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*. http://doi.org/10.1109/DSD.2009.159

Isakova, N., Başak, S., & Sönmez, A. C. (2012). FPGA design and implementation of a real-time stereo vision system. *Innovations in Intelligent Systems and Applications (INISTA), 2012 International Symposium on*. http://doi.org/10.1109/INISTA.2012.6247007

Jansen, B., Temmermans, F., & Deklerck, R. (2007). 3D human pose recognition for home monitoring of elderly. *2007 29th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. http://doi.org/10.1109/IEMBS.2007.4353222

Jena, S. K., Majhi, B., Gupta, A., Sehrawat, V. K., & Khosla, M. (2012). FPGA based Real Time Human Hand Gesture Recognition System. *Procedia Technology*, *6*, 98–107. http://doi.org/http://dx.doi.org/10.1016/j.protcy.2012.10.013

Ji, X., & Liu, H. (2010). Advances in View-Invariant Human Motion Analysis: A Review. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*. http://doi.org/10.1109/TSMCC.2009.2027608

Keskin, C., Erkan, A., & Akarun, L. (2003). Real time hand tracking and 3d gesture recognition for interactive interfaces using hmm. *ICANN/ICONIPP*, *2003*, 26–29.

Kolb, A., Barth, E., & Koch, R. (2008). ToF-sensors: New dimensions for realism and interactivity. *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW '08. IEEE Computer Society Conference on*. http://doi.org/10.1109/CVPRW.2008.4563159

Kollorz, E., Penne, J., Hornegger, J., & Barke, A. (2008). Gesture Recognition with a Time of Flight Camera. *Int. J. Intell. Syst. Technol. Appl.*, *5*(3/4), 334–343. http://doi.org/10.1504/IJISTA.2008.021296

Kurakin, A., Zhang, Z., & Liu, Z. (2012). A real time system for dynamic hand gesture recognition with a depth sensor (pp. 1975–1979). Bucharest. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84869786886&partnerID=40&md5=1064a9303d60d64f0e6abed1bd950251

Lakshmivarahan, S., Dhall, S. K., & Miller, L. L. (1984). Parallel Sorting Algorithms. In M. C. Y. B. T.-A. in Computers (Ed.), (Vol. Volume 23, pp. 295–354). Elsevier. http://doi.org/http://dx.doi.org/10.1016/S0065-2458(08)60467-2

Le, T. T., Chatelain, F., & Berenguer, C. (2014). Hidden Markov Models for diagnostics and prognostics of systems under multiple deterioration modes. In *European Safety and Reliability Conference (ESREL 2014)* (pp. 1197–1204). Wroclaw, Poland: Taylor & Francis - CRC Press/Balkema. Retrieved from https://hal.archives-ouvertes.fr/hal-

01027509

Lee, H. K., & Kim, J. H. (1999). An HMM-Based threshold model approach for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*(10), 961–973. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-0033210184&partnerID=40&md5=a50b9b21daa293081c3f86e3130ac384

Lee, S. H., SNC, C. P. O., & Siew, W. H. (2011). Real time FPGA implementation of hand gesture recognizer system. In *Proceedings of the 15th WSEAS international conference on Computers* (pp. 217–222). World Scientific and Engineering Academy and Society (WSEAS).

Lin, C.-S., & Hwang, C.-L. (1987). New forms of shape invariants from elliptic fourier descriptors. *Pattern Recognition*, *20*(5), 535–545. http://doi.org/http://dx.doi.org/10.1016/0031-3203(87)90080-X

Lin, C.-S., & Jungthirapanich, C. (1990). Invariants of three-dimensional contours. *Pattern Recognition*, *23*(8), 833–842. http://doi.org/http://dx.doi.org/10.1016/0031-3203(90)90130-D

Liu, L., & Shao, L. (2013). Learning Discriminative Representations from RGB-D Video Data. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (pp. 1493–1500). AAAI Press. Retrieved from http://dl.acm.org/citation.cfm?id=2540128.2540343

Liu, N., Lovell, B. C., Kootsookos, P. J., & Davis, R. I. A. (2004). Model structure selection & training algorithms for an HMM gesture recognition system. In F. Kimura & H. Fujisawa (Eds.), (pp. 100–105). Tokyo. http://doi.org/10.1002/0471648272.ch7

Liu, X. L. X., & Fujimura, K. (2004). Hand gesture recognition using depth data. In *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.* (pp. 529–534). Seoul. http://doi.org/10.1109/AFGR.2004.1301587

Malima, A., Özgür, E., & Çetin, M. (2006). A fast algorithm for vision-based hand gesture recognition for robot control. In *2006 IEEE 14th Signal Processing and Communications Applications* (Vol. 2006). Faculty of Engineering and Natural Sciences, Sabanci University, Tuzla, Istanbul, Turkey. http://doi.org/10.1109/SIU.2006.1659822

Manresa, C., Varona, J., Mas, R., & Perales, F. J. (2005). Hand Tracking and Gesture Recognition for Human-Computer Interaction. *ELCVIA Electronic Letters on Computer Vision and Image Analysis; Vol 5, No 3: Special Issue on Articulated Motion & Deformable Objects - August 2005*. Retrieved from http://elcvia.cvc.uab.es/article/view/109/106

Mattoccia, S. (2013). Stereo Vision Algorithms for FPGAs. *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops*. http://doi.org/10.1109/CVPRW.2013.96

Mitra, S., & Acharya, T. (2007). Gesture recognition: A survey. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, *37*(3), 311–324. http://doi.org/10.1109/tsmcc.2007.893280

Miyajima, Y., & Maruyama, T. (2003). A Real-Time Stereo Vision System with FPGA. In P. Y. K. Cheung & G. A. Constantinides (Eds.), *Field Programmable Logic and Application: 13th International Conference, FPL 2003, Lisbon, Portugal, September 1-*

3, 2003 Proceedings (pp. 448–457). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-540-45234-8_44

Mo, Z., & Neumann, U. (2006). Real-time Hand Pose Recognition Using Low-Resolution Depth Images. *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. http://doi.org/10.1109/CVPR.2006.237

Moeslund, T. B., Hilton, A., & Krüger, V. (2006). A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, *104*(2–3), 90–126. http://doi.org/http://dx.doi.org/10.1016/j.cviu.2006.08.002

Moni, M. A., & Shawkat Ali, A. B. M. (2009). HMM based hand gesture recognition: A review on techniques and approaches (pp. 433–437). Beijing. http://doi.org/10.1109/iccsit.2009.5234536

Muñoz-Salinas, R., Medina-Carnicer, R., Madrid-Cuevas, F. J., & Carmona-Poyato, A. (2008). Depth silhouettes for gesture recognition. *Pattern Recognition Letters*, *29*(3), 319–329. http://doi.org/10.1016/j.patrec.2007.10.011

Murphy, K. (1998). Hidden Markov Model (HMM) Toolbox for Matlab. Retrieved September 3, 2016, from https://www.cs.ubc.ca/~murphyk/Software/HMM/hmm.html

Nguyen-Duc-Thanh, N., Lee, S., & Kim, D. (2012). Two-stage Hidden Markov Model in gesture recognition for human robot interaction. *International Journal of Advanced Robotic Systems*, *9*. http://doi.org/10.5772/50204

Nickel, K., Scemann, E., & Stiefelhagen, R. (2004). 3D-tracking of head and hands for pointing gesture recognition in a human-robot interaction scenario. *Automatic Face and Gesture Recognition, 2004. Proceedings. Sixth IEEE International Conference on*. http://doi.org/10.1109/AFGR.2004.1301593

Nickel, K., & Stiefelhagen, R. (2007). Visual recognition of pointing gestures for human-robot interaction. *Image and Vision Computing*, *25*(12), 1875–1884. http://doi.org/10.1016/j.imavis.2005.12.020

Ohmura, I., Mitamura, T., Takauji, H., & Kaneko, S. (2010). A real-time stereo vision sensor based on FPGA realization of orientation code matching. *Optomechatronic Technologies (ISOT), 2010 International Symposium on*. http://doi.org/10.1109/ISOT.2010.5687334

Oikonomidis, I., Kyriazis, N., & Argyros, A. A. (2011). Efficient model-based 3D tracking of hand articulations using Kinect. In *BMVC* (Vol. 1, p. 3).

Park, C. B., & Lee, S. W. (2011). Real-time 3D pointing gesture recognition for mobile robots with cascade HMM and particle filter. *Image and Vision Computing*, *29*(1), 51–63. http://doi.org/10.1016/j.imavis.2010.08.006

Park, S., & Jeong, H. (2007). Real-time Stereo Vision FPGA Chip with Low Error Rate. *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*. http://doi.org/10.1109/MUE.2007.180

Park, S., Yu, S., Kim, J., Kim, S., & Lee, S. (2012). 3D hand tracking using Kalman filter in depth space. *Eurasip Journal on Advances in Signal Processing*, *2012*(1). http://doi.org/10.1186/1687-6180-2012-36

Pointgrey. (2016). Bumblebee2 FireWire stereo vision camera systems. Retrieved August 18, 2016, from https://www.ptgrey.com/bumblebee2-firewire-stereo-vision-camera-

systems

Poppe, R. (2010). A survey on vision-based human action recognition. *Image and Vision Computing*, *28*(6), 976–990. http://doi.org/http://dx.doi.org/10.1016/j.imavis.2009.11.014

Premaratne, P. (2014). *Human Computer Interaction Using Hand Gestures*. *Cognitive Science and Technology*. Adelaide, South Australia, Australia: Springer. http://doi.org/10.1007/978-981-4585-69-9

Rabiner, L. R. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*. http://doi.org/10.1109/5.18626

Rahman, M. H., & Afrin, J. (2013). Hand Gesture Recognition using Multiclass Support Vector Machine. *International Journal of Computer Applications*, *74*(1), 39–43.

Refaeilzadeh, P., Tang, L., & Liu, H. (2009). Cross-validation. In *Encyclopedia of database systems* (pp. 532–538). Springer.

Ren, Z., Meng, J., & Yuan, J. (2011). Depth camera based hand gesture recognition and its applications in Human-Computer-Interaction. In *8th International Conference on Information, Communications and Signal Processing, ICICS 2011*. School of EEE, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798, Singapore. http://doi.org/10.1109/ICICS.2011.6173545

Reyes, M., Domínguez, G., & Escalera, S. (2011). Featureweighting in dynamic timewarping for gesture recognition in depth data (pp. 1182–1188). Barcelona. http://doi.org/10.1109/iccvw.2011.6130384

Scharstein, D., & Szeliski, R. (2002). A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *International Journal of Computer Vision*, *47*(1), 7–42. http://doi.org/10.1023/A:1014573219977

Scharstein, D., & Szeliski, R. (2003). High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on* (Vol. 1, pp. I–I). IEEE.

Scharstein, D., Ugbabe, P., & Szeliski, R. (2011). 2001 Stereo datasets with ground truth. Retrieved September 3, 2016, from http://vision.middlebury.edu/stereo/data/scenes2001/

Schwarz, L. A., Mkhitaryan, A., Mateus, D., & Navab, N. (2011). Estimating human 3D pose from Time-of-Flight images based on geodesic distances and optical flow. *Automatic Face & Gesture Recognition and Workshops (FG 2011), 2011 IEEE International Conference on*. http://doi.org/10.1109/FG.2011.5771333

Schwarz, L. A., Mkhitaryan, A., Mateus, D., & Navab, N. (2012). Human skeleton tracking from depth data using geodesic distances and optical flow. *Image and Vision Computing*, *30*(3), 217–226. http://doi.org/http://dx.doi.org/10.1016/j.imavis.2011.12.001

Sempena, S., Maulidevi, N. U., & Aryan, P. R. (2011). Human action recognition using Dynamic Time Warping. *Electrical Engineering and Informatics (ICEEI), 2011 International Conference on*. http://doi.org/10.1109/ICEEI.2011.6021605

Shan, Y., Wang, Z., Wang, W., Hao, Y., Wang, Y., Tsoi, K., … Yang, H. (2012). FPGA based memory efficient high resolution stereo vision system for video tolling. *Field-*

Programmable Technology (FPT), 2012 International Conference on. http://doi.org/10.1109/FPT.2012.6412106

Shi, Y., Taib, R., & Lichman, S. (2006). GestureCam: A smart camera for gesture recognition and gesture-controlled web navigation. Singapore. http://doi.org/10.1109/icarcv.2006.345267

Shi, Y., & Tsui, T. (2007). An FPGA-based smart camera for gesture recognition in HCI applications. Tokyo.

Siriboon, K., Jirayusakul, A., & Kruatrachue, B. (2002). HMM topology selection for on-line Thai handwriting recognition. *First International Symposium on Cyber Worlds, 2002. Proceedings.* http://doi.org/10.1109/CW.2002.1180872

Suarez, J., & Murphy, R. R. (2012). Hand gesture recognition with depth images: A review. In *2012 21st IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2012* (pp. 411–417). Center for Robot-Assisted Search and Rescue, Texas AandM University, College Station, TX 77840, United States. http://doi.org/10.1109/ROMAN.2012.6343787

Suryanarayan, P., Subramanian, A., & Mandalapu, D. (2010). Dynamic Hand Pose Recognition Using Depth Data. *Pattern Recognition (ICPR), 2010 20th International Conference on*. http://doi.org/10.1109/ICPR.2010.760

Tang, M. (2011). Recognizing hand gestures with microsoft's kinect. Retrieved from http://www.masters.dgtu.donetsk.ua/2012/fknt/sobolev/library/article5.pdf

Vasicek, Z., & Sekanina, L. (2008). Novel Hardware Implementation of Adaptive Median Filters. *2008 11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems*. http://doi.org/10.1109/DDECS.2008.4538766

Wang, X., Xia, M., Cai, H., Gao, Y., & Cattani, C. (2012). Hidden-Markov-Models-based dynamic hand gesture recognition. *Mathematical Problems in Engineering*, *2012*. http://doi.org/10.1155/2012/986134

Weinland, D., Ronfard, R., & Boyer, E. (2011). A survey of vision-based methods for action representation, segmentation and recognition. *Computer Vision and Image Understanding*, *115*(2), 224–241. http://doi.org/http://dx.doi.org/10.1016/j.cviu.2010.10.002

Wilson, A. D., & Bobick, A. F. (1999). Parametric hidden Markov models for gesture recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*(9), 884–900. http://doi.org/10.1109/34.790429

Woodfill, J. I., Gordon, G., Jurasek, D., Brown, T., & Buck, R. (2006). The Tyzx DeepSea G2 Vision System, ATaskable, Embedded Stereo Camera. *2006 Conference on Computer Vision and Pattern Recognition Workshop (CVPRW'06)*. http://doi.org/10.1109/CVPRW.2006.202

Xu, W., & Lee, E. J. (2012). Continuous gesture recognition system using improved HMM algorithm based on 2D and 3D space. *International Journal of Multimedia and Ubiquitous Engineering*, *7*(2), 335–340. Retrieved from http://www.scopus.com/inward/record.url?eid=2-s2.0-84865619564&partnerID=40&md5=45533bc3145cfb311a8fc8ebc3d55fbb

Yang, C., Jang, Y., Beh, J., Han, D., & Ko, H. (2012). Gesture recognition using depth-based hand tracking for contactless controller application. *2012 IEEE International*

Conference on Consumer Electronics (ICCE). http://doi.org/10.1109/ICCE.2012.6161876

Yin, X., & Xie, M. (2003). Estimation of the fundamental matrix from uncalibrated stereo hand images for 3D hand gesture recognition. *Pattern Recognition*, *36*(3), 567–584. http://doi.org/http://dx.doi.org/10.1016/S0031-3203(02)00072-9

Zabih, R., & Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence. In J.-O. Eklundh (Ed.), *Computer Vision --- ECCV '94: Third European Conference on Computer Vision Stockholm, Sweden, May 2--6 1994 Proceedings, Volume II* (pp. 151–158). Berlin, Heidelberg: Springer Berlin Heidelberg. http://doi.org/10.1007/BFb0028345

Zhang, C., & Tian, Y. (2013). Edge enhanced depth motion map for dynamic hand gesture recognition. In *2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPRW 2013* (pp. 500–505). Department of Electrical Engineering, City College of New York, United States. http://doi.org/10.1109/CVPRW.2013.80