Instituto Tecnológico de Costa Rica

School of Electronic Engineering

Optical system for measuring position of metallic colored pellets on a platform.

Graduation project report to qualify for the title of Electronic Engineer with the academic degree of Bachelor

Robert J. Barnes Pérez

Prague, Spring Semester 2016

INSTITUTO TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA ELECTRÓNICA

PROYECTO DE GRADUACIÓN

ACTA DE APROBACIÓN

Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado "Optical system for measuring position of metallic colored pellets on a platform." (Sistema óptico para medición de posición de balines metálicos coloreados en una plataforma), realizado por el señor Robert J. Barnes Pérez y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal Evaluador

[illegible]
Ing. William Marín Moreno
Profesor lector

[illegible]
Ing. Juan Carlos Jiménez Robles
Profesor lector

TEC | Tecnológico de Costa Rica
Ingeniería Electrónica

[illegible]
Ing. Eduardo Interiano Salguero
Profesor asesor

Cartago, 23 de Agosto de 2016

Declaration of Authenticity

I declare that this Graduation Project has been made, in full, by my person, using and applying literature on the subject and introducing my own knowledge.

Where I have used bibliographical material, I proceeded to indicate sources by citation.

Consequently, I take total responsibility for the work done and the contents of the graduation final report.

[illegible]

Robert Javier Barnes Pérez
Id: 1-1532-0549

Abstract

This thesis is the result of an investigation research on alternatives to develop a computerized vision system to measure the position of colored pellets on a planar noncontact magnetic manipulation platform (MagMan). This has been achieved before using MATLAB & Simulink environment, however, this thesis explores different options in order to improve the processing performance. First, the problem is going to be defined to set goals and objectives. Second, the research work is going to be discussed with the purpose of selecting and then describing the most adequate environment of the new computer vision system. The image processing methods considered for implementation are explained in the main body of the work, covering mainly color-detection techniques and object position measuring algorithms; which derived two productions: in Verilog Hardware Description Language and C code; both implemented on a Terasic tPad. The design processes in which the vision systems were implemented is included, both deviations are characterized in detail and then tested and compared against each other to finally conclude which is more effective to solve the given problem.

Keywords

FPGA, Computer vision system, digital image processing, image segmentation, color detection, position measuring, Terasic tPad, camera, Hardware description Language (HDL), Verilog, C code, Heterogeneous computer.

Resumen

Esta tesis es el resultado de un proceso de investigación sobre las alternativas para desarrollar un sistema de visión computarizado para medir la posición de balines coloreados en una plataforma plana de manipulación magnética sin contacto (MagMan). Esto se ha logrado antes usando el ambiente de desarrollo MATLAB y Simulink; sin embargo, esta tesis explora diferentes opciones con el fin de mejorar el rendimiento de procesamiento. En primer lugar, el problema será definido para establecer metas y objetivos. En segundo lugar, el trabajo de investigación será discutido con el propósito de seleccionar y luego describir el entorno más adecuado del nuevo sistema de visión computarizado. Los métodos de procesamiento de imágenes considerados para la aplicación se explican en el cuerpo principal de la tesis, que cubre principalmente técnicas de detección de color y algoritmos que miden la posición del objeto; los cuales derivaron en dos producciones: tanto en el lenguaje de descripción de hardware Verilog, como en código de lenguaje C; ambos implementados en un Terasic tPad. Los procesos de diseño en el que se aplicaron a los sistemas de visión están incluidos, ambas desviaciones se caracterizan en detalle y luego son probadas y comparadas entre sí para finalmente concluir cuál es más eficaz para resolver el problema dado.

Palabras Clave

Arreglo de Compuertas Programable (FPGA), sistema de visión computarizado, procesamiento digital de imágenes, segmentación de imágenes, detección de color, medición de posición, Terasic tPad, cámara, lenguaje de descripción de hardware (HDL), Verilog, código C, computador heterogéneo.

Dedication

To my parents, whose constant effort has resulted in the consummation of this work.

Acknowledgments

Foremost, I would like to acknowledge to my supervisor Ing. Jirí Zemánek, first for providing the opportunity to develop my undergraduate project abroad, which has been a major goal for me; and second for his valuable advices and support during its progress.

Also, I would like to thank Ing. Ivo Herman, Ing. Štefan Knotek and Ing. Martin Gurtner, the other members of the Control Engineering department with whom I shared the work environment, interesting discussions and other experiences that stretched my vision of life.

Last but not least, I would like to thank to my parents and my family for their biggest, inspiring and unconditional support during this studying process which made me grow personally and professionally, always guided to self-improvement while maintaining humility.

General Index

Index of Figures

Index of Tables

Chapter 1. Introduction

This project is based on an existing laboratory platform, which was developed by AA4CC group at the Department of Control Engineering, in the Faculty of Electrical Engineering at Czech Technical University in Prague. The overall project studies the problem of handling the non-contact manipulation on a flat surface using a magnetic force field [25].

1.1. Environment of the Project

The laboratory platform for flat magnetic manipulation (MagMan) has a flat touch foil that is used as a running floor for metal pellets (fig. 1.1). Under the foil there are four electromagnetic modules located adjacently, with dimensions of 50x50x75 mm. These sections are composed of four iron core coils; intended to create the magnetic field above. The modules allow independent PWM current control for each of their coils, so they can act simultaneously and independently to guide the pellets on the desired way (fig. 1.2).

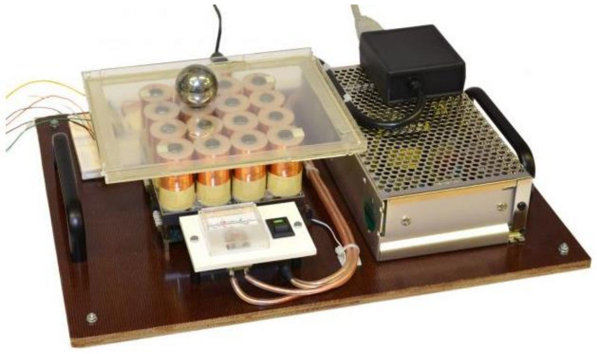


Figure 1.1. MagMan laboratory platform with a steel ball. **[11]**

The position of the balls on the surface can be set from two possible sources: either from a Matlab file or from an iPad app made especially for the MagMan [25]. The platform offers two options of position measurement of the steel ball in real time: the resistive touch sheet that is used as surface or a RGB camera set above [11]. The processing unit of the platform uses an ARM Cortex M3 processor at 72 MHz.

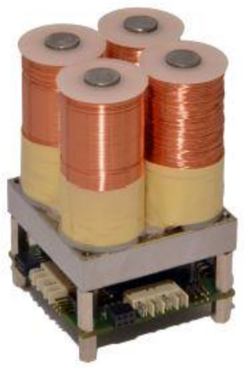


Figure 1.2. Photograph of a single magnetic module. **[11]**

From the point of view of control engineering, the platform is capable of operate with one or more pellets, which makes it a multiple-input multiple-output system (MIMO). As explained before, the board has four magnetic modules composed of four iron core coils. The magnetic field generated by each of these coils is continuous. Since they are located next to each other in a square matrix, the attractive energy created by one coil affects the surrounding coils and so on; inducing a complex force field in the platform. The project was conceived as a model and tool to investigate strategies to control this type of dynamic systems [26]. Fig. 2 shows a simplified modular diagram of the system

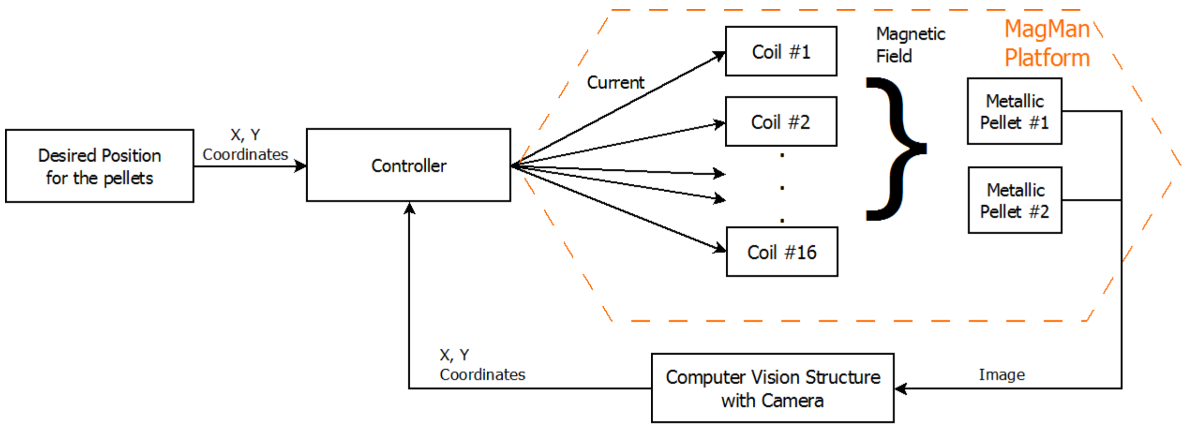


Figure 1.3. Simplified modular diagram of the MagMan platform.

The ultimate motivation for the project is to contribute to development of efficient distributed feedback manipulation schemes [25]. Although the configuration shown is in the centimeter scale, it can be extrapolated to show the essence of some problems occurring in micrometer scale; which could be useful in fields such as biotechnology and analytical chemistry and electrophoresis.

The experimental setup of the platform was awarded first place in the “Matlab & Simulink Student Design Challenge” event of 2013 [15].

1.2. Definition the problem

The main problem of the MagMan platform lies in the visual system of position measurement. Although there is the resistive foil, the visual method is needed because it can process multiple elements on the platform, provided that they are optically distinguishable. The key drawback of the optical method is that it has slower response than the resistive sheet. There are two stages for the visual method for measuring the ball's position: Image acquisition and Computer vision (Fig. 1.4) [11].

The first phase comprises of a camera Basler acA2000-340kc attached to a Bitflow Neon-CLB frame grabber using the Camera Link interface [18]. At this block, the lens's exposure time is set to 7.5 ms, while it takes approximately 14 ms to transfer the image from the frame grabber to Simulink environment via USB [11].

On the second stage, the Simulink tool converts the RGB image signal into a matrix of hue and color saturation, making segmentation of the balls on the platform easier [1].This algorithm requires on average 4.6 ms to determine the pellet's location from the image, according to [11]. Therefore, the place where the ball is cannot be known sooner than 26.1 ms after the start of the camera exposure. In summary, the time delay caused by the computer vision algorithm limits the performance of the control system.

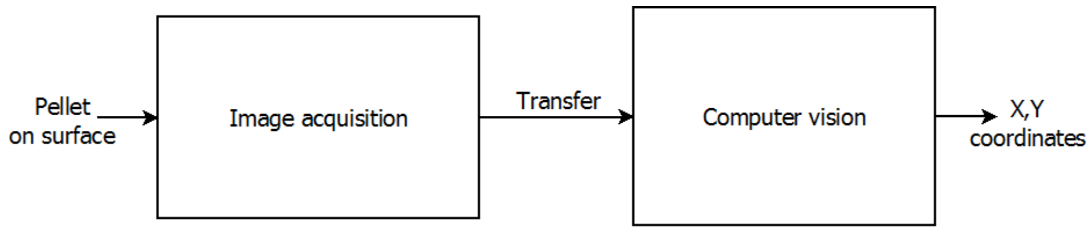


Figure 1.4 Scheme of camera model.

To advance in the dielectrophoresis research and the tests for control of distributed manipulation for which Magman was devised, the visual feedback system needs to be improved. This way investigators would have a more accurate placement of the pellets on the platform.

1.3. Synthesis of the problem

The observing system of metallic colored balls of the Magman platform is not capable of acquire and process information at the speed required by the controller, hence the overall performance is restricted.

1.4. Solution approach

As depicted in the previous section, the main problem is that the visual method used to measure position is too slow compared to the controller, limiting its precision.

In the first place, the project executor must learn how does the current system processes the image, based on the documents of the previous implementations. The next step is to gain insights into algorithms for detection of position of one (or more) steel balls in a video-sequence obtained from camera. After evaluation of different procedures, the most efficient has to be chosen, accordingly with the hardware to be used.

In parallel with the algorithms' investigation, different hardware options shall be evaluated. For example, hardware available options will be investigated to determine whether the existing camera is sufficient or if a new one is needed. The new proposal should be a specialized system capable to obtaining a real-time detection that would give a better speed of detected position, for applicability of the image based feedback. According with this, it will be considered to develop the new system using a Field Programmable Gate Array (FPGA).

As explained in the previous section, the current scheme transfers the image from the camera to the frame grabber via Camera Link interface, and then the data is sent via PCI to Simulink environment. According to [11], a specialized system such as FPGA would solve the inherent problems of serial transmission; given that using a high speed camera with a built-in FPGA board will allow a significant increase in the frequency of the position measurement not only by running the computer vision algorithms on the board, but also by avoiding the lengthy transfer of the whole image to the PC. The higher frequency achieved, the better the feedback. Consequently, one of the foremost options for developing the project will be the FPGA.

Nevertheless, for the solution of the problem there will be shown at least two different options, and the definitive solution will be chosen based on the comparison of the characteristics of the possibilities depicted.

Chapter 2. Goal and objectives

2.1. Goal

In the long term, it is intended that the magnetic field manipulator becomes extremely useful in a research of the response of objects exposed to an electric field, their polarization and subsequent interaction of their dipole (quadrupole, octopole) with the surrounding electric field [26]. This phenomenon is called dielectrophoresis, and its results can be used to steer, characterize and sort objects as delicate as cells; hence its appeal for fields such as biochemistry and bioanalytical instrumentation [26].

2.2. General objective

Implement a specialized computer vision system to measure the position of colored metallic balls, to improve the performance of the image based feedback of the laboratory platform for magnetic manipulation (MagMan).

2.3. Specific objectives

1. Implement an algorithm for detection of objects in a video stream obtained from a camera.

   Indicator: The object must highlight a proof that it has been recognized.

2. Develop a process capable of distinguish different colors in a video stream obtained from a camera.

   Indicator: After the image processing, the system must be able to differentiate colors and label them.

3. Implement a system capable of measure the position of one or more objects on a flat surface from an image.

   Indicator: The system must send via RS-232 the absolute position of the objects according with the dimensions of the surface.

Chapter 3. Computer Vision Systems

The initial phase of this project consisted on a survey over cameras and computer systems available in the market, in parallel with a research of related projects. The search was mainly focused on cameras compatible with FPGAs or microcontrollers, in order to come up with the device that fits best the requirements of the system, without ignoring the current implementation. The main requirements to consider are the processing time and the cost of the device to use.

Since the main body of the project, including all the designs of architectures that develop the operations will be performed on the processing unit, it will be discussed first. Then, the camera options will be reviewed based primarily in their compatibility with the selected processing unit.

3.1.Processing Unit

Before starting the engineering design for the new computer vision system, the options for developing such structure must be reviewed. This is a critical choice because it will directly impact on the rest of the design cycle. The main characteristic to consider is the complexity of the application, besides the cost, capacity and the flexibility the technology offers [9].

The current vision system transfers the image to a Personal Computer via USB, to Simulink environment where the processing occurs. This provides all the flexibility the workspace can give (libraries, algorithms, tools and commands), with the disadvantage that the operations will be handled mainly by software, which does not take full advantage of the hardware [14]. Also, the bottleneck caused by transferring the image (as explained in section 1.2) limits the performance of the platform, so it's necessary to find a new solution.

Given the nature of the project, image processing systems usually are computationally intensive but also structured. This type of applications can be mapped on FPGA or application specific integrated circuit (ASIC) (Khan, 2010). Both options differ in some key features as flexibility, cost, size and performance. According to Bailey (2011), dedicated hardware as an ASIC would be definitively the optimal option for several reasons: a custom made circuit will always be smaller, faster and more efficient in terms of power than an FPGA. The main drawbacks of an ASIC are the economic cost of manufacture a specific chip and its certainly null flexibility. Since the computer vision system may need to be changed in the future, the ASIC is discarded as option for the project. There is also the possibility to develop the image processor in a microcontroller.

After ruling out the ASIC, the remaining options are the FPGA and the microcontroller. There is a wide range of options in both markets including variety in prices and performance. For terms of this project, were considered microcontrollers that the department owns and some other options that will be described shortly.

With the microcontroller, the first option would be a Raspberry Pi 3 model B, which has a 1.2 GHz 64-bit quad-core ARM v8 CPU, Camera and display interfaces and VideoCore IV 3D graphics core [17]. This is way too advanced in single threading processing, and it offers the option of 3D image handling for further investigations. Another advantage of this device is its size, what makes it discrete.

There is also the Terasic Video Embedded Evaluation Kit (VEEK/tPad) has a Cyclone IV FPGA that works at a frequency of 50 MHz and provides 114480 logic elements, plus RS-232, USB and Flash memory ports. This kit features a 5 MP camera and a LCD touchscreen, and it is addressed to host embedded processing-based systems. It also offers different picture sizes, which varies inversely proportional the frame rate [23].

The Optomotive Velociraptor HS has an embedded Spartan-GLX FPGA and is designed for high speed image processing at JPEG format. This device features a gigabit Ethernet port and a 4 pin trigger connector, which would require to set up a small server to upload the measured position.

Last but not least, there is the industrial-aimed DSP-PCIe/104 board. This board features a Xilinx Virtex-5 FPGA which has 12 960 logic slices and supports PCI-Express interface which provides fast communication with a host computer. This device can also be connected with the Basler camera using an FG-400CL expansion package [19].

The most practical of all the previous options is the Optomotive camera because it also has the FPGA attached, all together in a small device. The main drawback of this camera is its few options of connectivity [16], given that the project is to be designed for continuous serial transmission. The next one is the powerful Virtex-5 FPGA because it would provide fewer processing limitations, which can be translated in more parallelism possibility in the design; besides it is compatible with the Basler camera previously used. Unfortunately, this board is not owned by the university; and its price is roughly 4000 USD [19] plus the adapter for the camera, which leaves it out of the question.

The Raspberry Pi, as a microcontroller is a good processor of general applications, being highly efficient executing instructions from software. However, it does not offer the possibility of parallel threading, which makes it inappropriate for the task of processing images. On the other hand, the tPad offers flexibility both in hardware (FPGA) and software processing. This is particularly useful in this kind of applications given that the image can be analyzed either way: the more convenient. Also, the tPad has its camera attached, it has a wide range of connectivity options (Ethernet, USB, RS-232, PS2, SMA, among others) [21] to transmit the measured position, what results in not needing more devices. Its main drawback are its size and the fact that it's phased out by a new model from Altera (VEEK-Multi Touch) [24].

3.2.Image Sensor Technology

After selecting the FPGA as optimal choice to host the processor, the next argument is about what camera is the most adequate for the image acquisition.

The Magman platform’s first vision system was composed by a Basler acA2000-340kc camera, attached to a Bitflow Neon-CLB frame grabber [18]. Both components will be portrayed in this chapter, and then other hardware options will be described for the new implementation , to finally conclude on which alternative will be used and why was it chosen.

The Basler camera is a high speed industrial camera based on a CMOS sensor, and is able to deliver up to 340 frames per second (fps) at 2 Megapixels resolution (2046 x 1086), providing images of 8, 10 or 12 bits wide. It supports Camera Link interface and its typical power consumption is 3.0 W. This camera worked on a frame rate of 112 fps because it was not possible to process the data at camera’s full rate without a specialized hardware such an FPGA or GPU (Simonian, 2014). It would be possible to attach this camera to an FPGA or a microcontroller but in most cases this would require an undercard that make the connection possible. For example, this camera can be attached to a PCIe 104 FPGA though an FG-400 CL daughter card, given that it is based on Camera Link interface [19].

The Optomotive Velociraptor HS is, by far, the fastest option considered for this project. It has an embedded Spartan-GLX FPGA and is able to achieve a frame rate of 178 fps at maximum resolution (2048x2048) [16]. As discussed in the previous section, it is also the most practical in terms of the size to functionality ratio but this artifact is not in the faculty; it would have to be bought.

The Terasic tPad may not have the fastest camera or the best image resolution but the truth is that the algorithms for image segmentation do not require the largest camera resolution to work properly; this will be discussed with more detail in the design stage. The main advantage of this device is that it can use the Nios II processor (described in section 4.1.3). According to [3], the Nios II is the ideal real-time mainframe to use with DSP (Digital Signal Processing) Builder-based hardware accelerators to provide deterministic, high performance real-time results. This particular feature makes it the best choice to develop the new computer vision system for the Magman platform.

Based on the previous argument, the selected device to host the computer vision system will be the Altera Terasic tPad due its practicality, flexibility and its high compatibility with several pheripherals.

Chapter 4. Digital Image Processing

In this chapter are described different algorithms used for image processing in order to detect diverse colors and shape from an array of pixels. The goal is to implement a system capable of measure the position of one or several colored steel balls in the Magman Platform in an empirically estimated frame rate of 50 Hz; while the position calculated has to be good enough to be used as feedback for the control system of the platform.

4.1.Problem Setting

For the detection problem we are aiming to solve, there are some considerations to be taken before describing digital processing algorithms, given their effectiveness can vary according to the environment they are going to be used. First, the distance from the camera to the platform is considered constant, even though there is a small difference between the distance to the camera is from the edges and from the center of the platform. Second, the illumination of the scene is stable, which means that long exposure periods before acquiring the image are not necessary. Third, the platform in the image is static, the only moving objects are going to be the steel balls. Fourth, the objects to be detected have known shape and are symmetric and one object cannot hide another, which simplifies the identification process.

Since the main constraint for the new computer vision system is speed, the algorithms to be implemented cannot be too complex because this would mean more response time as a result of dense computation. Another consideration in order to improve processing speed is that, once the objects are identified there is no need to scan the whole frame again because the movement of the ball in the platform cannot be more than a certain number of pixels. In this course of actions, the scanned section will be called region of interest (ROI); where its center is the point where the ball was located in the last frame.

One of the advantages the Altera Terasic tPad is its flexibility: it can process the image via hardware or software or both. The algorithms can be described and synthesized in Verilog by the Quartus II design software or can also be programmed by software using the Nios II

Software Build tools for Eclipse. The Design Software are described further in Chapter 5 in sections 5.2 and 5.3.

4.2. Color Detection

The human eye is capable to distinguish hundreds of shades of different colors, and this expanded perception of our reality encouraged the development and optimization of devices able to acquire, store and show full-colored frames. According to [13], the use of color in image processing is encouraged for its practicality in object identification from a frame.

An important portion of the image segmentation that requires this project is to discern between colored elements, so this branch is fundamental for the development. This section explains briefly the basic concepts of colored pictures to then explain some algorithms that can be used in order to detect the colored balls in the platform.

All colored displays from the Cathode Ray Tube (CRT) monitors and TVs, to the digital Plasma, LCD and LED screens used nowadays are based on producing all color variations with the three primary colors: Red, Green and Blue; where the characteristics used to distinguish one color from another are brightness, hue and saturation [13]. There are other protocols for color encoding such as Cyan, Magenta and Yellow (CMY); Hue, Saturation and Value (HSV); Luminance and Chrominance (YCbCr), among others; but cover them all goes beyond the scope of this document, so we will focus only in RGB and HSV color spaces.

4.2.1. RGB and HSV color models

The RGB color model is based on the three dimensional Cartesian coordinate system, where each axis represents the value of one of the three primary colors: Red, Green and Blue, hence its name. If all the values of these axes are normalized in the interval [0,1], the color space can be modeled as a cube that contains all achievable colors; furthermore, the primary colors are in three different corners, while the secondary colors are in other three corners. A straight line traced from the origin [0,0,0] to the point [1,1,1] runs the entire grayscale from black to white[13], as shown in fig.4.1.

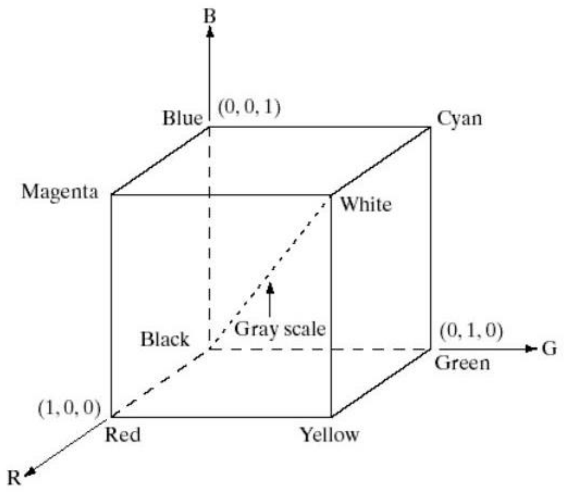


Figure 4.1. Schematic of the normalized RGB colorspace cube. **[13]**

According to González and Woods (2007), RGB color model is useful to build colors in hardware implementations, however it is not well suited to describe colors. Humans describe colors based on its hue, saturation and brightness; being *hue* the attribute that describes a pure color, *saturation* is the measure of white light and value is a measure of *brightness*. The HSV colorspace is mapped putting the vector of RGB grayscale in vertical position with the black vertex at the bottom and the white vertex above it, becoming the intensity axis (fig 4.2). The cube can then be extruded into a hexagonal shape where the vertices are the primary and secondary colors (fig 4.4). Henceforward, primary colors are separated by 120° each other and 60° to the secondary counterparts as shown in the fig. 4.3, where the dot is an arbitrary

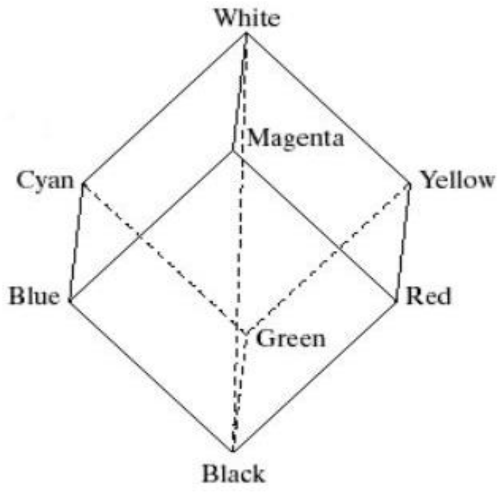


Figure 4.2. Conceptual relationship between RGB and HSV color models. **[13]**

color point, the saturation value is radial and the hue is the angle from the red vertex, incrementing in counter clockwise direction. If the values of R, G and B axes are normalized in the interval [0,1], the equations 4-1, 4-2 and 4-3 allow to calculate the Hue, Saturation and Intensity respectively [13].

$$H = \begin{cases} \theta & if\, B \leq G \\ 360 - \theta & \text{if } B > G \end{cases} \tag{4-1}$$

Where

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}\cdot[(R-G)+(R-B)]}{[(R-G)^2+(R-B)\cdot(G-B)]^{1/2}}\right\}$$

Saturation is given by

$$S = 1 - \frac{3}{(R+G+B)}\cdot[min(R,G,B)] \tag{4-2}$$

Value is given by

$$V = \frac{1}{3}\cdot(R+G+B) \tag{4-3}$$

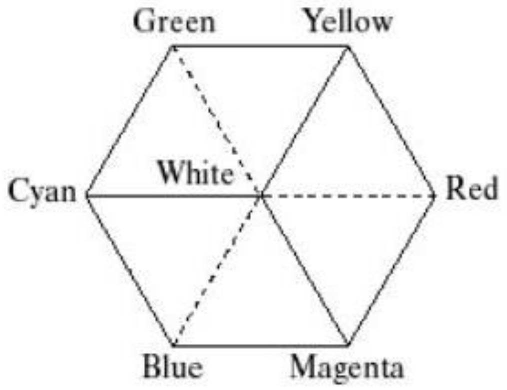


Figure 4.4. RGB cube extrusion to HSV color model. **[13]**

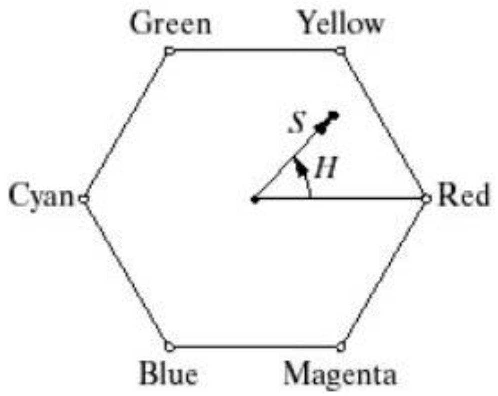


Figure 4.3. Hue and Saturation in the HSV color model. **[13]**

4.2.2. HSV Threshold in RGB colorspace

This method relies on the concepts described in the previous section but without the conversion from one space to another. In another words, it offers the effectiveness of thresholding in HSV colorspace without the expense of transforming each pixel. According to [18] hue, saturation and value isosurfaces are linear or piecewise linear in terms of RGB components. So, if the isosurfaces corresponding to the hue, saturation and value are described analytically using plane equations, we will be able to do the HSV thresholding by simply checking whether a pixel lays above or below a set of boundary planes in the RGB space [18]. Figures 4.5 through 4.7 show the HSV isosurfaces in RGB colorspace for the reader's understanding.

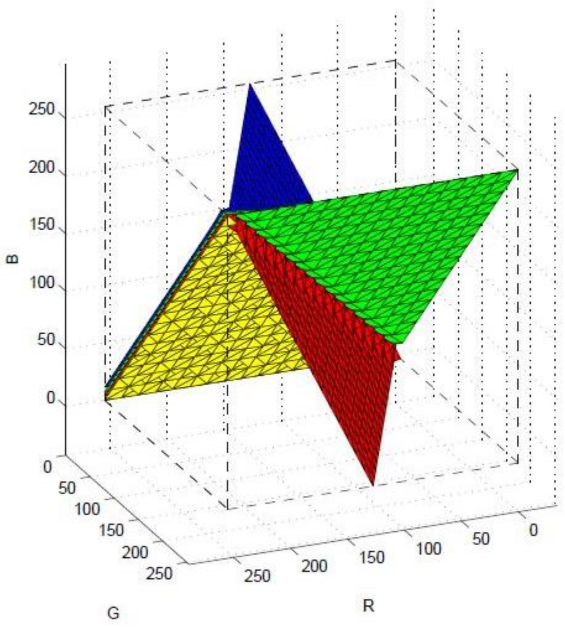


Figure 4.7. Hue isosurfaces in the RGB color space for $H = 0$ (yellow), $H = 0.25$ (red), $H = 0.5$ (green) and $H = 0.75$ (blue). **[18]**

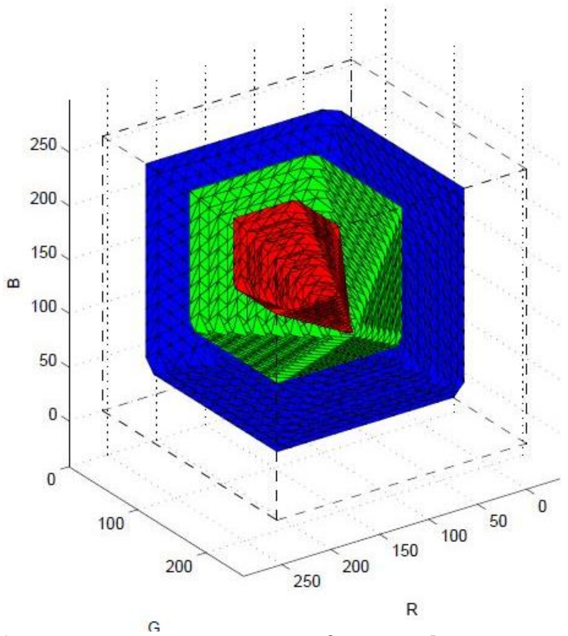


Figure 4.6. Saturation isosurfaces in the RGB color space for $S = 0.25$ (red), $S = 0.5$ (green) and $S = 0.75$ (blue). **[18]**

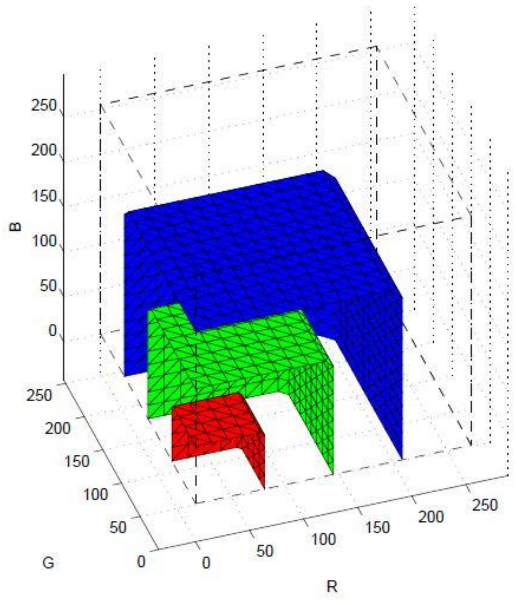


Figure 4.5. Value isosurfaces in the RGB color space for $V = 0.25$ (red), $V = 0.5$ (green) and $V = 0.75$ (blue). **[18]**

4.2.3. Pixel Threshold

This is a very simple method based in the merely concept of thresholding the pixels that meet certain value. For example: in an RGB image, by simply verifying the channels of each pixel, can we can examine if it belongs to a red object we need to track. Following this idea, in a pixel depth of 30 bits (just as used in the tPad) each channel has 10 bits, making the possible values in the interval [0,1023]. In order to detect a red object, the thresholding would be as simple to set an arbitrary value of red channel, and every pixel above this limit is considered foreground; while those don't as background.

It is almost superfluous to explain the fragility of a color detection algorithm based on this purpose. Here it is mentioned only for completeness but for a real implementation it would be less than useless. First, it would only allow (in the best case scenario) to detect red, green or blue colors in their purest versions, excluding similar shades. Second, a system based on linear threshold would also be highly susceptible to light changes. The only advantage this procedure can have among other algorithms is that it is the fastest: just needs three comparators; but does not meet the requirements of this project.

4.2.4. Segmentation in RGB Vector Space

This is an improved version of the previous procedure. It is also mapped in the RGB colorspace with the difference that the color identification is not based on a fixed value of R, G or B but on a *region* around the color we desire to detect. The objective of segmentation is to classify each RGB pixel as inside the region stated or not [13]. In order to determine the proximity of each pixel to the average of the region selected, the simplest way is to calculate the Euclidean distance. Depending on how the distance is calculated, the tridimensional region can be either a cube, a sphere or an ellipsoid. For example, equation 4-4 is used to calculate the Euclidean distance between the average color we want to discriminate (vector **a**) and an arbitrary point in RGB color space (vector ***z***). Therefore, the color denoted by **z** is considered as *similar* to ***a*** if the distance between them is less than the specified threshold [13].

$$D(\boldsymbol{z},\boldsymbol{a}) = \|\boldsymbol{z}-\boldsymbol{a}\| = [(\boldsymbol{z}-\boldsymbol{a})^T \cdot (\boldsymbol{z}-\boldsymbol{a})]^{\frac{1}{2}} \tag{4-4}$$

If a maximum distance D_0 is set, the similar points in the color space will look like a sphere of radius D_0. Since calculating the square root for images of useful sizes takes a heavy processing because of the number of pixels, it’s simpler to perform this threshold in a cubical shape centered in **a.** This can be done comparing separately the distance in the three directions (R.G.B) between **z** and **a,** which, according to [13] is simpler computationally than calculating the spherical enclosure.

4.3.Image Segmentation [13]

According to [13] image segmentation in computer vision systems is the process to subdivide the image into regions or objects whose pixels share a characteristic feature, in order to obtain information from it. In our case, the goal is to measure the position of the object of interest, with the advantages that it has known shape and if there is more than one, they cannot occlude each other; which simplifies the segmentation.

4.3.1. Basic line and edge detection

In a digital image, typically an *edge pixel* is found when there is a discontinuity in the intensity function, while an *edge segment* is a set of connected edge pixels. The detection of edges in digital image processing is based on derivatives guided by the following rules for the first derivative:

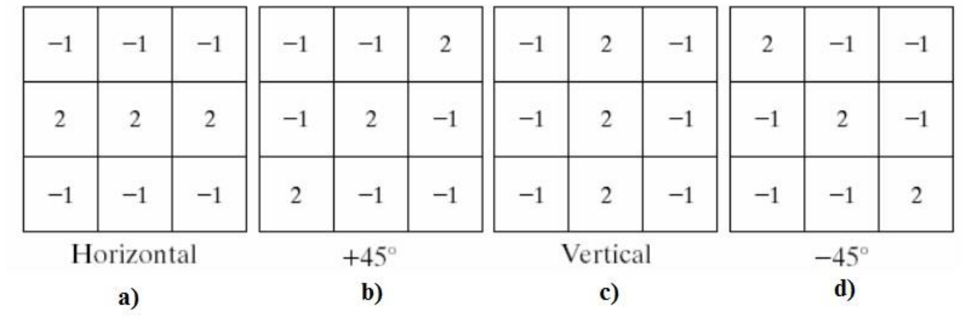
a) Must be zero in areas of constant intensity;
b) Must be nonzero at the onset of an intensity step or ramp;
c) Must be nonzero at points along an intensity ramp.

For the second derivative, the following rules apply:

a) Must be zero in areas of constant intensity;
b) Must be nonzero at the onset and end of an intensity step or ramp;
c) Must be zero along intensity ramps.

Based on these statements, an expression for the first order derivative is obtained by expanding a linear function in Taylor series, being the function $f(x+\Delta x)$ and $\Delta x=1$.

Equations 4.5 and 4.6 show the result of this approximation of one-dimension first and second order differentials, respectively.



a) Horizontal			b) +45°			c) Vertical			d) −45°		
−1	−1	−1	−1	−1	2	−1	2	−1	2	−1	−1
2	2	2	−1	2	−1	−1	2	−1	−1	2	−1
−1	−1	−1	2	−1	−1	−1	2	−1	−1	−1	2

Figure 4.8. Line detection masks for different orientations. **[13]**

$$\frac{df}{dx} = f'(x) = f(x+1) - f(x) \tag{4-5}$$

$$\frac{d^2f}{dx^2} = f''(x) = f'(x+1) - f'(x)$$

$$f''(x) = f(x+1) + f(x-1) + f(x) \tag{4-6}$$

An interesting example where an 8-bit image is subjected to the previous equations is shown in [13], pp 693-695. For our interest, the conclusions of such experiment are: (1) First-order derivatives produce thicker edges in an image. (2) Second-order derivatives have stronger response to fine detail. (3) Second-order derivatives produce a double-edge response at ramp and step transitions in intensity. (4) The sign of the second-order derivatives can be used to determine whether a transition into edge is from dark to light or vice-versa.

The previous explanation is useful as an introduction to the general process of segmentation, but is still in one direction. For *real* image processing applications, the common procedure is to encompass the frame using a filter mask or a kernel, which is usually a matrix of different values depending of the target. Some example of masks are shown in fig. 4.8, being a) a mask optimized to find horizontal lines through d)-45° lines.

4.3.2. The Marr-Hildreth edge detector [13]

David Marr and Ellen Hildreth proposed in 1980 two premises in which they based their model of edge detection: a) intensity changes are not independent of image scale, and b) a sudden intensity change will give raise to a peak in the first derivative, or a zero crossing at the second derivative, as proven in the previous section. To fulfill this requirements, Marr and Hildreth suggested the Laplacian operator was the one who fulfilled best their expectations, applied to a 2-Dimensional Gaussian function. (eq. 4-7). The final expression is called *Laplacian of Gaussian* (LoG), and it is depicted in eq. 4.8.

$$G(x,y) = e^{-\frac{x^2+y^2}{2\cdot\sigma^2}} \tag{4-7}$$

$$\nabla^2 G(x,y) = \frac{\partial^2 G(x,y)}{\partial x^2} + \frac{\partial^2 G(x,y)}{\partial y^2}$$

$$\nabla^2 G(x,y) = \left[\frac{x^2+y^2-2\cdot\sigma^2}{\sigma^4}\right]\cdot e^{-\frac{x^2+y^2}{2\cdot\sigma^2}} \tag{4-8}$$

The Laplacian of Gaussian can then be converted into a low-pass filter mask of variable size (for an example, see fig. 4.9.). As a rule, the size of an $n\times n$ LoG filter should be that n is the smallest odd integer greater than or equal to 6σ; being σ the standard deviation. Since the mask is isotropic, its convolution with the frame yields a blurred picture where the intensity of the structures is decreased in any direction, thus avoiding to use several masks depending of the direction of the edges of interest. The algorithm by itself consists of convolving the LoG filter with an image *f(x,y)* (eq. 4-9) and then finding zero crossings in *g(x,y)* to determine the locations of edges in *f(x,y).*

$$g(x,y) = \nabla^2[G(x,y) * g(x,y)] \tag{4-9}$$

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Figure 4.9. Laplacian of Gaussian 5× 5 mask. **[13]**

The principal feature of the Marr-Hildreth edge detection algorithm is the zero detection of *g(x,y)* because its simplicity and its good results. Its main drawback is the treatment of negative numbers in hardware which may imply denser processing and slower performance.

4.3.3. Center of Mass

This method is a simple procedure to calculate the center of a shape in a binary image. It is based upon accumulating the weight of each pixel times the indexed position in one dimension (x for demonstration) for each row, divided by the sum of the weights of each pixel for each row, as shown in eq. 4-10.

$$X_{cm} = \frac{\sum\sum x \cdot g(x,y)}{\sum\sum g(x,y)} \qquad (4\text{-}10)$$

One disadvantage of this method is that it calculates the center of the concentrations of the pixels for virtually any shape, and for the purposes of this project it is required that the method does so for only circular shapes. For this reason this method was discarded for the final implementation.

4.3.4. Foreground Model

This method was proposed by [18] as a procedure to detect the position of the balls from a background subtracted image. This image is considered as a two-dimensional matrix of the same size of a region of interest *D(i,j)*, where each pixel is classified between two threshold values t_1 and t_2 to have three different classes, where $t_1 < t_2$. (eq. 4-11)

$$C(D_{ij}) = \begin{cases} 0 & \text{iff } \ 0 < D_{ij} \le t_1 \\ 1 & \text{iff } \ t_1 < D_{ij} \le t_2 \\ 2 & \quad\quad \text{otherwise} \end{cases} \qquad (4\text{-}11)$$

According to eq. 4-11, the pixels with classification $C(D_{ij})=0$ belong to the background, $C(D_{ij})=2$ belongs to foreground and $C(D_{ij})=1$ means “not determined” tier. The higher threshold t_2 is chosen empirically as a quantile of the difference image pixel values, while t_1 is chosen as one half of the threshold t_2. After the pixel classification we form two vectors whose lengths are equal to the width *M* and the height *N* on the image. These vectors, v_1 and v_2 contain sums of rows and columns respectively (see eqs. 4-12)

$$v_1 = \sum_{j=1}^{N} C(D_{ij})$$

$$(4\text{-}12)$$

$$v_2 = \sum_{i=1}^{M} C(D_{ij})$$

After having the rows and columns in vectors, the next step is to create a digital filter to detect the shape we are interested in. Simonian proposes an analysis based on the projections of the secant lines of the circle in a column sum vector, which provides the filter shown in eq. 4-13 for any fixed circle radius. The evaluation for this filter in a circle with radius of 70 pixels can be seen in fig. 4.10.

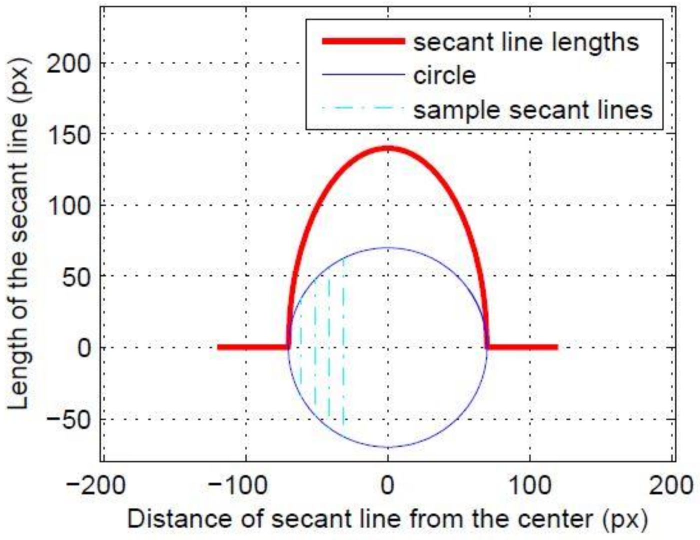


Figure 4.10. Length of parallel secant lines of a circle with radius 70 px as a function of their distance from the circle's center (red line), i.e. the coefficients of filter fc. The blue circle represents projection of a ball to the image plane. A few sample secant lines are shown as well, plotted with cyan color **[18]**

$$f_c(x) = \begin{cases} 2 \cdot \sqrt{r^2 - x^2} & \text{iff } x \in [-r, r] \\ 0 & \text{otherwise} \end{cases} \tag{4-13}$$

Finally, the procedure to know the center of the ball is as simple as the location of the maximum result of the convolution between image vectors (v_1 and v_2) and the digital filter shown in fig 4.10. The mathematical expression for this operation is described in eq. 4-14.

$$\begin{aligned} x_c &= \arg\max_i\{(\boldsymbol{v}_1 * \boldsymbol{f}_c)(i)\} \\ y_c &= \arg\max_j\{(\boldsymbol{v}_2 * \boldsymbol{f}_c)(j)\} \end{aligned} \tag{4-14}$$

4.4. Alternatives Assessment

In this section we discuss briefly which methods are the most appropriated for the main parts of the computer vision system, being color detection and circle detection in the image.

For color detection, the pixel threshold based only in R, G or B channels at the same time the easiest and less reliable approach, given that the hue cannot be judged based on the value of only one channel. The optimal solution for color detection would be to transform each pixel to the HSV colorspace to evaluate if it belongs to a color of interest, but this requires dense calculations that would slow the performance of the system. The same thing happens with segmentation in RGB colorspace because the Euclidean distance for each pixel would need to be calculated and compared to different values of the different colors to identify. The best option is the HSV threshold in RGB colorspace because it only needs the parametrization of the thresholds needed and two comparators for hue and for saturation.

For image segmentation, the pure edge detection based on differentials leads to the Marr-Hildreth edge detection algorithm, but this turns out to be overly complicated to handle because of the negative coefficients in the kernel. Besides, edge detection is not the main goal but detection of the *whole* shape; so despite its effectiveness is discarded. For the center of mass calculation, as mentioned in section 4.3.3, it can calculate the center of any shape so it provides no filtering at all. Finally, the Foreground model proposed by [18] holds a balance between effectiveness and difficulty, so that if it's synthesizable in Verilog, the pixels can be first classified as background or foreground in different levels or colors; and then this method could be used to locate the circles in the image.

Chapter 5. System Analysis and Design

This chapter presents the constraints analysis according to the selected FPGA's processing time and data consistency. It also presents the FPGA development board used for the new computer vision system, the Quartus II design software and the Nios II Build Tools for Eclipse. All considered options are described in terms of programming language, FPGA development board, CMOS sensor used, embedded operating system and the soft-core mainframe

5.1. Terasic tPad overview

The Terasic tPad is a design environment capable to perform processing based systems. It is composed by the DE2-115 development board attached to a multimedia daughter card via HSMC port. The core of the DE2-115 board, where the Nios II processor is installed and all logic is performed, is the Cyclone IV FPGA. The tPad and Cyclone IV FPGA technical specifications are presented in table 5.1, and table 5.2 respectively.

Table 5.1 Technical specification of tPad board. **[23]**

Feature	**Value [unit]**
Operation Voltage DC	12 [V]
Flash Memory	8 [MB]
EEPROM	32 [Kbit]
SRAM	2 [MB]
SDRAM	128 [MB]
Clock frequency	50 [MHz]
Width	160 [mm]
Large	223 [mm]
Depth	47 [mm]

The board is to be configured via USB blaster, but it supports JTAG AS configuration as well. As it can be seen in fig.5.1, the device counts also with many peripheral connectors, which signifies a wide range of possibilities for processing digital signals including audio and video in real time. The board has on its back side a 5 Mega Pixel camera with an LCD 8” touch screen (fig. 5.2).

Table 5.2. Resources of Cyclone IV EP4CE115 FPGA. **[20, 21]**

Logic Elements [LEs]	114480
Embedded Memory [kbits]	3888
Embedded 18x18 multipliers	266
General Purpose PLLs	4
Global Clock Networks	20
User I/O Banks	8

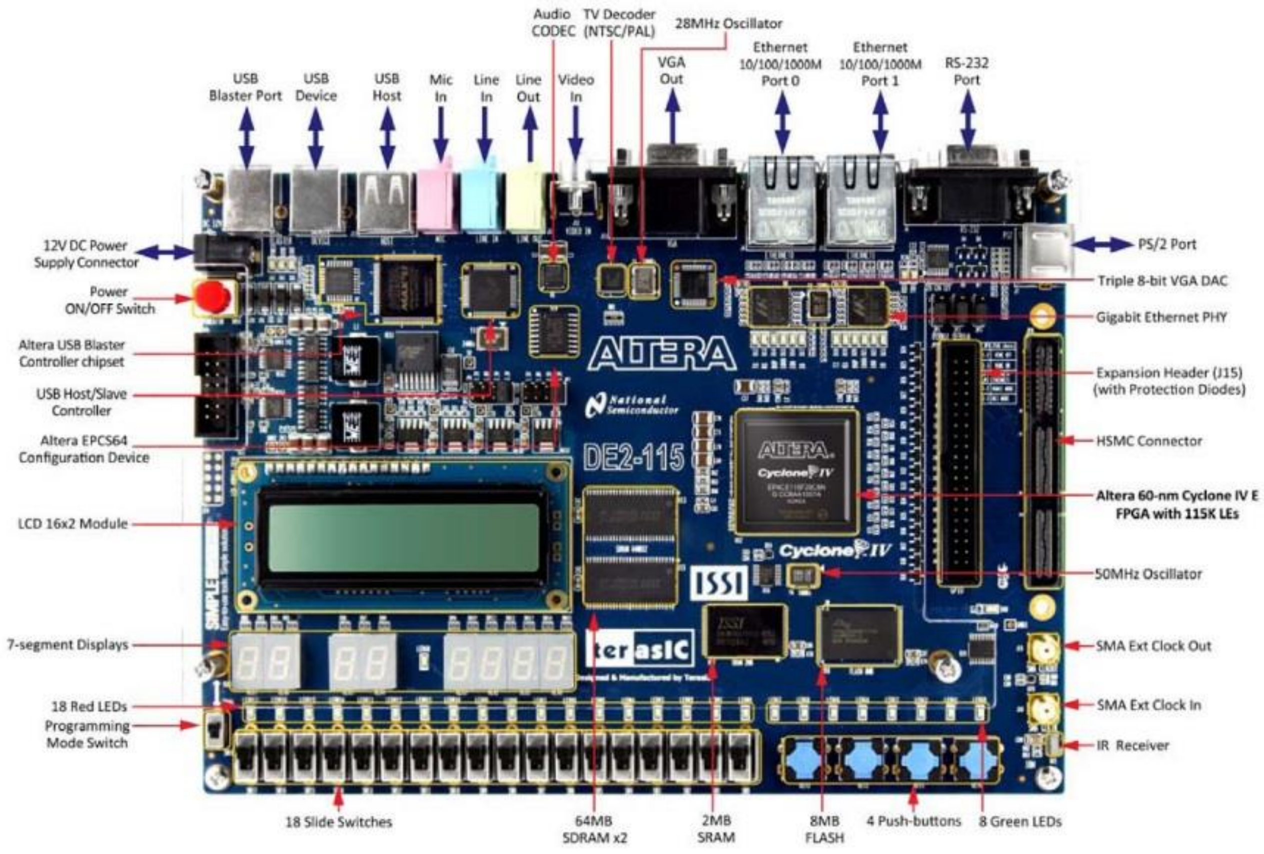


Figure 5.1. Altera Terasic tPad (bottom view). **[21]**

On the top side, the board has 9 green and 18 red leds for output or display, it also has 18 slide switches and 4 push buttons for inputs. It also shows a 16x2 LCD screen where can be shown messages as desired. It also has various types of memories: SDRAM, SRAM, FLASH and EEPROM. Among its connectors features an IR receiver, a TV decoder, a CD-quality audio CODEC with line-in, out, and microphone-in jacks; VGA DAC with VGA-out connector, USB ports type A and B, 2 SMA connectors, 2 Gigabit Ethernet PHY ports and a RS-232 transceiver.

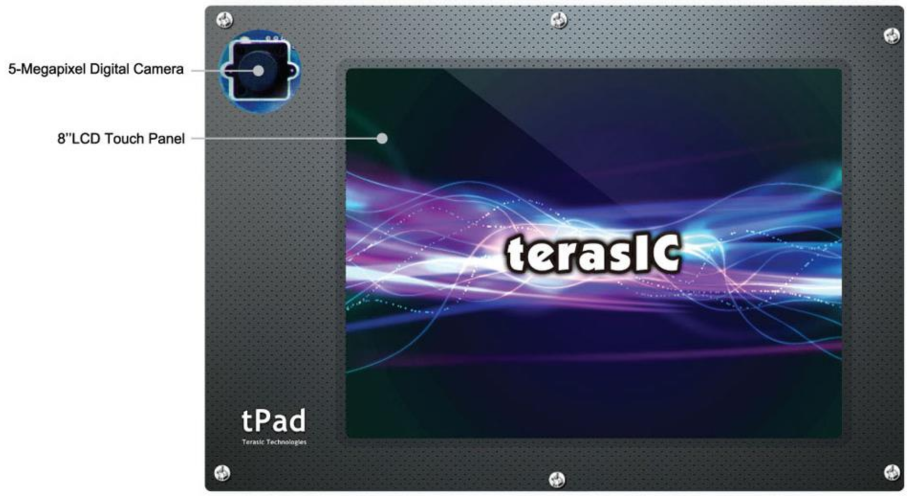


Figure 5.2. Altera Terasic tPad (top view). **[23]**

For the development of the computer vision system, the tPad offers a set of IP cores that can be selected, connected and synthesized using the Qsys tool for building architectures. This tool is also useful to instantiate the softcore Nios II Processor, which can be built in the FPGA; operated through a RISC instructions set (see section **5.2**).

5.1.1. CMOS camera sensor [22, 23]

The tPad is equipped with a 5-Megapixel digital image sensor module (D5M) that provides an active imaging array of 2,592 x 1,944 pixels as maximum resolution. It features low-noise CMOS technology that achieves CCD quality. The sensor can be operated in its default mode (maximum resolution) or programmed by the user through I2C serial interface for frame size. Other sizes supported by the 5DM are listed in table 5.3 with the corresponding frame rate achievable with an input clock of 96 MHz.

The CMOS sensor acquires the image in Bayer pixel format plus a blank region regardless the frame size (see fig. 5.3). This implies that the blank area of the acquired array of pixels need to be cropped, and the active image must be converted to RGB format before any further changes. Further specifications of the camera's readout sequence can be found in [22]

Table 5.3. Table of picture sizes with maximum frame rate achievable. **[22]**

Resolution H x V [pixels]	**Frame Rate [fps]**
2592 x 1944	15.15
2048 x 1536	23
1600 x 1200	35.2
1280 x 1024	48
1024 x 768	73.4
800 x 600	107.7
640 x 480	150

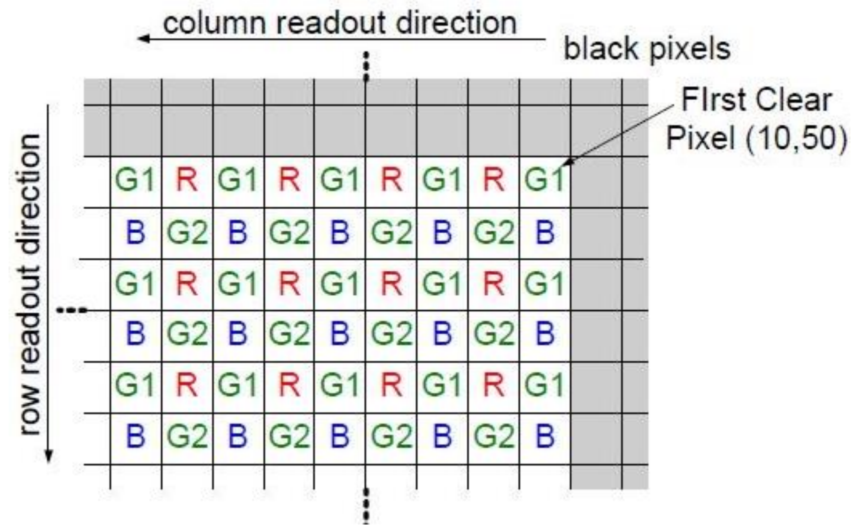


Figure 5.3. Bayer pixel format and readout direction of the D5M sensor. **[22]**

5.1.2. LCD touch screen [7, 23]

The tPad features an 8-inch LCD panel that supports resolution of (800x600) pixels with backlight. The LCD panel works with an 18-bit parallel RGB data interface and an input clock of 40 MHz. The tPad is also equipped with an Analog Devices AD7843 touch screen chip that digitizes analog x and y coordinates of touch points to 12-bit coordinates, which can be read in the FPGA through the serial port interface on the AD7843. Table 4.4 shows some technical details of the screen (note the touch characteristics were ignored due to its futility in this project). Further specifications of the LCD can be found in [7]

Table 5.4. Technical details of the LCD touchscreen. **[7]**

Feature	**Value [unit]**
Screen size	8 inches (diagonal)
Power supply	3.3 [V]
Digital operating current (max)	120 (132) [mA]
Power consumption (max)	396 (436) [mW]
Backlight LED voltage (max)	9.9 (10.5) [V]
Backlight LED current (max)	180 (200) [mA]
Resolution	800 x 600 pixels @ RGB
Active area	162.0(W) x 121.5(H) [mm]*
Module size	183.0(W) x 141.0(H) x 7.2(D) [mm]*
Color arrangement	18 bits RGB
Interface	Digital

***(W):** width; **(H):** height.

5.1.3. Nios II processor

There are two kinds of mainframes in FPGAs according to Antunes (2010): hard and soft core. The Hardware based processors (*hard-core*) are located in a specific area of the integrated circuit, while the *soft-core* processors are installed in the general-purpose FPGA logic cells [8]. Altera's Nios II is a soft-core general-purpose RISC processor that can be instantiated on an Altera FPGA device. In the Quartus II development environment are three different alternatives: economic, standard and fast [3].

Economic: Ideal for microcontroller applications. It uses the fewest FPGA logic and memory resources [4].

Standard: is designed to implement a small processor core without a significant trade off in performance This core is optimal for cost-sensitive, medium-performance applications, such as computers running a full-featured operating system [6].

Fast: designed for high performance. Optimal for performance-critical applications as well as applications with large amounts of code and data, such as running a full-featured operating. The Nios II fast core can use a memory management unit (MMU) to run embedded Linux [5].

Figure 5.4 shows a simplified modular diagram of the Nios II structure. Some other features of the Nios II processor are listed below:

- Full 32-bit instruction set, data path, and address space
- 32 general-purpose registers
- 32 interrupt sources
- External interrupt controller interface for more interrupt sources
- Single-instruction 32 × 32 multiply and divide producing a 32-bit result

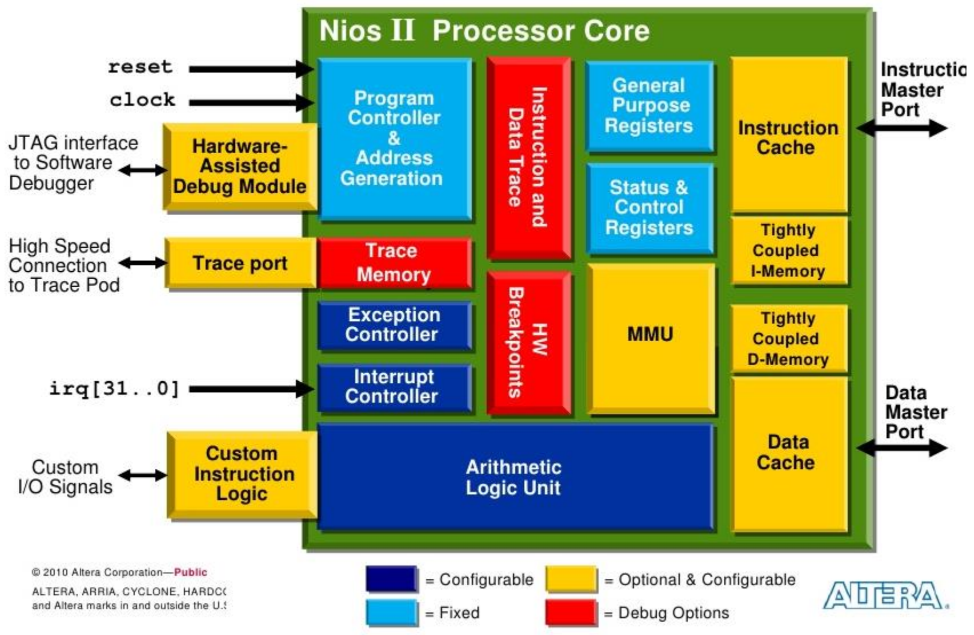

Nios II Processor Core
reset
clock
JTAG interface to Software Debugger
Hardware-Assisted Debug Module
High Speed Connection to Trace Pod
Trace port
irq[31..0]
Custom I/O Signals
Custom Instruction Logic
Program Controller & Address Generation
Trace Memory
Exception Controller
Interrupt Controller
Instruction and Data Trace
HW Breakpoints
General Purpose Registers
Status & Control Registers
MMU
Arithmetic Logic Unit
Instruction Cache
Tightly Coupled I-Memory
Tightly Coupled D-Memory
Data Cache
Instructio Master Port
Data Master Port
= Configurable
= Optional & Configurable
= Fixed
= Debug Options
© 2010 Altera Corporation—Public
ALTERA, ARRIA, CYCLONE, HARDC
and Altera marks in and outside the U.
ALTERA

Figure 5.4. Nios II processor block diagram. **[3]**

- Dedicated instructions for computing 64-bit and 128-bit products of multiplication

- Optional floating-point instructions for single-precision floating-point operations
- Single-instruction barrel shifter
- Access to a variety of on-chip peripherals, and interfaces to off-chip memories and peripherals
- Hardware-assisted debug module enabling processor start, stop, step, and trace under control of the Nios II software development tools

5.1.4. RS-232 serial transmission port [21, 27]

The DE2-115 board uses a ZT3232 transceiver chip and a 9-pin DB9 connector for RS-232 communications, with flexible power supply from 3V to 5.5V. The chip supports both EIA/TIA-232 and V.28/V.24 communication interfaces, consisting of two line drivers, two line receivers and the proprietary switch-capacitor regulated voltage converters.

The ZT3232E chip supports NSOIC, WSOIC, SSOP and TSSOP package types for transmission and reception, at a data rate of 250 kbps. Image 5.5 shows the connections between the FPGA and the ZT3232E chip.

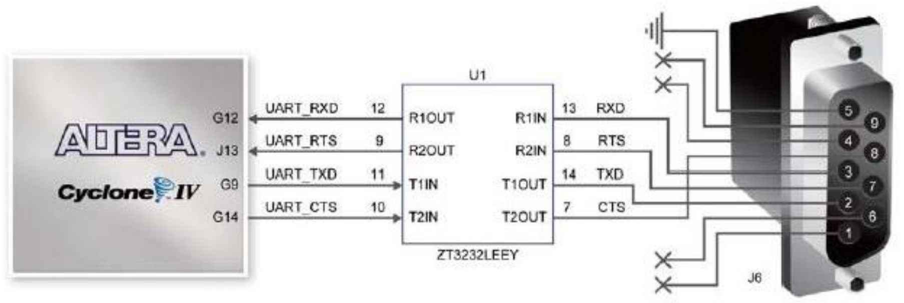


Figure 5.5. Pin connections between Cyclone IV FPGA and ZT3232E chip. **[21]**

5.2. Design Software

In order to work with the Terasic tPad, the software development environment that this board requires must be briefly introduced. The main tools to be used will be the Altera Quartus II System Development Software for hardware and the Eclipse Nios II for software.

5.2.1. Quartus II

The Quartus II development environment is useful for analysis and synthesis of the structures based on hardware description languages like Verilog or VHDL. The Altera Quartus II design software provides a complete, multiplatform design environment for system-on-a-programmable-chip (SOPC) designs. This environment synthesizes the scheme implemented and fits it on the tPad. It also performs various tests including connectivity checks, timing constraints and cell occupancy, all this in order to evidence the (in) validity of the proposed schemes.

The Altera Quartus II design software also includes the Qsys tool, which is a powerful environment where the core of the Nios II processor can be instantiated and connected with the peripheral modules as required by the design. It also includes templates of hardware described modules compatible with the built-in architecture of the tPad, which will simplify the implementation of the systems.

Another useful tool included in the Quartus II software is the TimeQuest Timing Analyzer. As its name says, its main function is to review the described system searching for any kind of errors related to timing, such as clocks, fan out, delay, slack, among others. Other tools used are the pin planner, that allows to assign the FPGA pins where the input and output signals will be connected and the Programmer, which downloads the compiled design into the board.

5.2.2. Nios II Software Build Tools for Eclipse.

This tool allows to create and modify the software code run by the Nios II soft core processor, if included in the design. The programming language is C/C++, and after compilation this tool transforms the code into the binary file executed by Nios II. The Eclipse environment also allows access to the libraries ran by the hardware described modules created with the Qsys tool, which expands its utility in heterogeneous systems. Fig. 5.6 shows the development flow required to build an ASIC controlled by the Nios II processor. Note that this image includes the workflow integrating all the three main tools of the environment (Quartus II, Qsys and Eclipse)

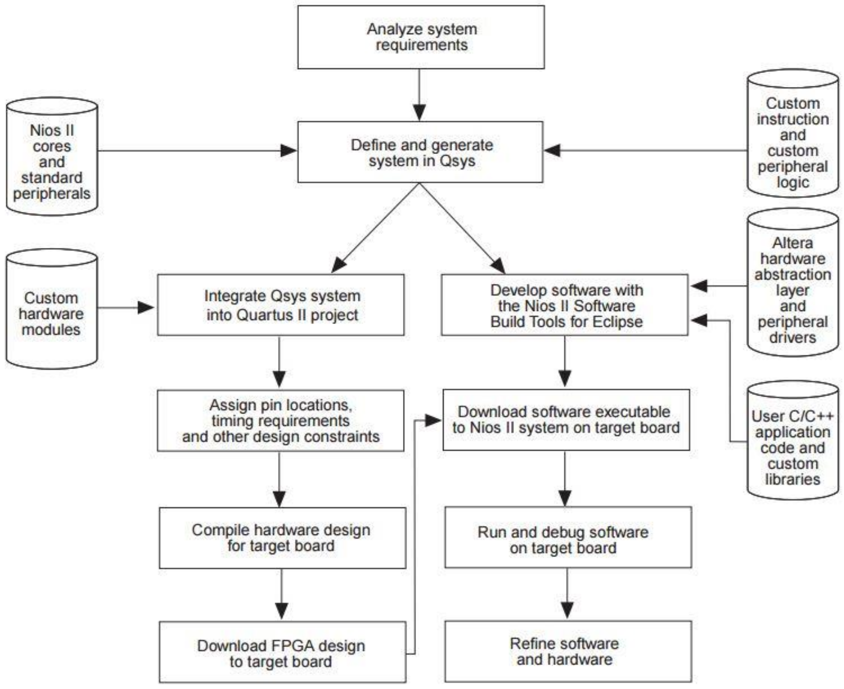

Analyze system requirements
Nios II cores and standard peripherals
Define and generate system in Qsys
Custom instruction and custom peripheral logic
Custom hardware modules
Integrate Qsys system into Quartus II project
Develop software with the Nios II Software Build Tools for Eclipse
Altera hardware abstraction layer and peripheral drivers
Assign pin locations, timing requirements and other design constraints
Download software executable to Nios II system on target board
User C/C++ application code and custom libraries
Compile hardware design for target board
Run and debug software on target board
Download FPGA design to target board
Refine software and hardware

Figure 5.6. Nios II development flow. **[2]**

5.3.Programming Languages

This section briefly describes the programming languages used in this project, including the purposes of each one. Given that the system has both hardware and software components, a hardware description language and a high level language are going to be used respectively to develop each domain.

5.3.1. Hardware Description Language

At the lowest level, the digital systems are composed by logical gates. To design and compile a complex structure at this level would be unmanageably tedious and error prone. According to Bailey (2011), it is possible to program an FPGA at this level, but it would be as programming a microprocessor in assembly language; while using a Hardware Description Language (HDL) allows to describe the operation of a circuit in a "human readable" form.

“A Verilog design consists of a hierarchy of modules. Modules communicate with each other through a set of declared input, output, and bidirectional ports.”[8]. Since the design in code actually represents hardware, the main difference it has with software programming is that the diverse blocks can work and communicate simultaneously when instantiated in the FPGA or even in simulations.

For this project, the chosen HDL is Verilog, given that it is the most common for digital designs in the data-transfer level of abstraction. Besides, it also has a lot of support in the internet via forums, which is a helpful advantage.

5.3.2. Software Programming Language

Given the complexity of the new computer vision system, a portion of the processing of the images could be done via software. C is a procedural structured programming language and it is supported by a wide number of software compilers.

As mentioned before, the Nios II microprocessor can be programmed in C programming language via the Eclipse Software Building Tools. Furthermore, each one of the Altera University Program’s modules have a set of functions defined in Hardware Abstraction Layers (HALs), which are compiled in C libraries. This particularity can facilitate the treatment of basic operations as memory addressing or interruption handling.

Chapter 6. Design of the solution

Even on the same device, there are several ways to implement a system that executes a given function. Is in this stage when the selection of the architecture, the algorithms and the procedure become variables that can play on favor or against the performance of the final circuit. In the chapter 3 was discussed how the Terasic tPad was selected as the background hardware to host the new computer vision system; and its features have been described in chapter 5. In this chapter the outline of the structure will be depicted from the acquisition of the image through two possible ways of processing it.

6.1. Acquisition of the image and Heterogeneous Computer.

Acquisition of the image is a key feature in this project. A very important part of the final performance of the new vision system depends on the rate at which it can collect and store one complete frame. Since the main restriction is speed, the pipeline was designed to classify the pixels as they were input instead of storing the image first and then execute the classification. In the other hand, a Heterogeneous computer provides the advantages of hardware parallelism in the Video-In pipeline combined with the ease of algorithm instantiation via C code as described in section 5.3.2.

The architecture of this version of the new computer vision system is a mixture of two different examples that use the tPad Camera. The first part of the framegrabber is based on the camera example provided with the tPad's CD. The second part is an adaptation of the VEEK media computer. Its core is the Nios II processor and it is instantiated with its peripherals using the Qsys tool (see section 5.2.1 for further details). The Video-In pipeline is depicted in the image 6.1 inside the orange square while the tasks for the mainframe are inside the blue square. The different modules that compose the whole diagram in fig 6.1 are described below.

CMOS Configuration: Originally developed by Altera. This module was designed to communicate with the CCD via I2C port to set up several operational parameters such as row and column size, analog channel gains, starting exposure, among others. These

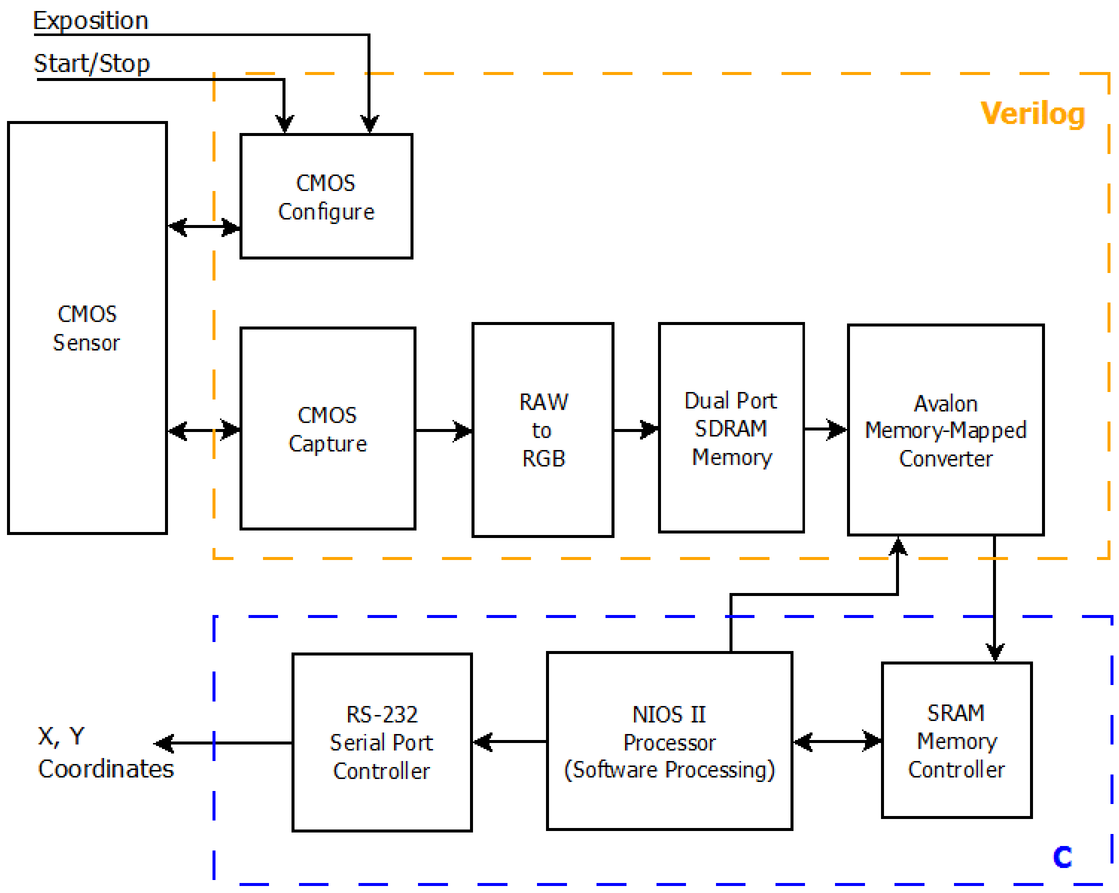


Figure 6.1. Modular diagram of the heterogeneous computer used as first solution.

were changed in order to fit the system requirements. Table 6-1 shows the parameters that were modified. Note that gain values are unit less because they represent multiplication coefficients.

<u>CCD Capture:</u> Originally developed by Altera, this module was not modified. Its function is to receive the incoming data and signals from the CMOS sensor. The data is then converted to the frame captured using two counters that give the address of the pixels received. These are switched to the system clock domain and sent to the Bayer Pattern Resampler.

<u>Bayer Pattern Resampler:</u> (Raw to RGB). This module converts a video stream from the Bayer Pattern format to the 30-bit RGB format combining four adjacent pixels from the incoming stream into one. Since for each four pixels are two green, one red and one blue, the green values are averaged in the final array. It is important to highlight that this module diminishes the image to half of its original resolution. Originally developed by Altera, this module was not modified.

Dual Port SDRAM: this is a dual clock FIFO memory that stores the frames. It is composed by two parallel blocks of 240 kb. Originally developed by Altera, this module was not modified.

Avalon Memory-Mapped Converter: this module was originally developed by [10], and it was not modified. The purpose of this block is to allow Nios II processor to read the frames temporarily stored in the SDRAM and convert them to the Avalon Memory-Mapped bus format.

Table 6.1. Configuration parameters for the CMOS sensor.

Parameter	**Value [unit]**
PLL output freq.	144 [MHz]
Exposure time	16,567 [ms]
Row size	599 [pixels]
Column size	799 [pixels]
G_1 gain	4,469
R gain	6,781
B gain	5,328
G_2 gain	1,625

NIOS II: the processor is instantiated in Qsys and it is used to move the pixels read by the CMOS controller and store them in the SRAM memory. For this case the obvious choice is the fast core, given it is the most powerful. This core also offers hardware divide and multiplication, which are both useful to perform complex operations if needed. The Nios II processor is also useful to control the RS-232 serial port to transmit the coordinates of the found objects.

SRAM Memory Controller: This block represents both a 2 MB memory and its controller. This module is used to store the pixels and is controlled by both the processor and the Pixel Buffer DMA Controller.

Once the image is successfully converted to the Avalon Memory-Mapped bus format and then stored in the SRAM memory, we consider it has been successfully acquired. Image display is not part of the objectives of this project, however, is a powerful tool in order to control if the frame is being acquired correctly. It also allows to practically check other parameters as noise, colors and object detection. The rest of the pipeline is described below for completeness, but it is not part of the final design.

Pixel Buffer DMA Controller: It uses its Avalon memory-mapped master interface to read video frames from an external memory (a SRAM), and then sends them out via its Avalon streaming interface. This module can be programmed from software using the Nios II processor to read and modify the image stored in the memory.

Dual Clock FIFO: buffers video data and help transfer a stream between two clock domains. It is necessary due to the difference between the clocks of the system and the LCD touchscreen.

VGA Controller: The VGA controller IP core takes the incoming data, and then sends that information to either the on-board VGA DAC or the LCD with touchscreen daughtercard synchronously either with an external clock (for the LTM touchscreen) or a clock generated by the module (for the VGA DAC).

LCD Screen: Shows the image acquired. This screen, the FIFO memory and the VGA controller are included in the design even though they are not necessary for the ball detection, but they are useful in order to control the procedures applied.

The Heterogeneous computer was also useful to develop the Dynamic Region of Interest logic. This procedure places the center of the ROI of the next iteration in the coordinates found for the ball, and it is explained in section 6.3 for System Initialization.

6.2. Image Processing via Dedicated Hardware

The definitive version of the computer vision system uses some elements from the previous design and adds new components (see fig. 6.2). The first modules: CCD configuration through Raw to RGB converter are still used, but the SDRAM FIFO was replaced by a dual port RAM memory. The pixels are received by the Saturation Filter, where their RGB values are evaluated in accordance with the HSV thresholding in RGB colorspace described in chapter 4 in two saturation levels (0.25 and 0.50) and seven possible colors plus black denoting background. Each pixel is then decoded in three bits according to color and stored in the dual port RAM memory.

In the output, the read signals and output from Frame Buffer are multiplexed in order to fulfill both operation modes of the computer vision system: screen or position measure. The idea is to be able to align the visual space of the camera using the screen mode and then to turn it to Position Measuring Mode. The screen mode also allows to show the pixels that are understood as foreground, which can be taken as reference to adjust either hue or saturation in the circuit, or lightning in the scene.

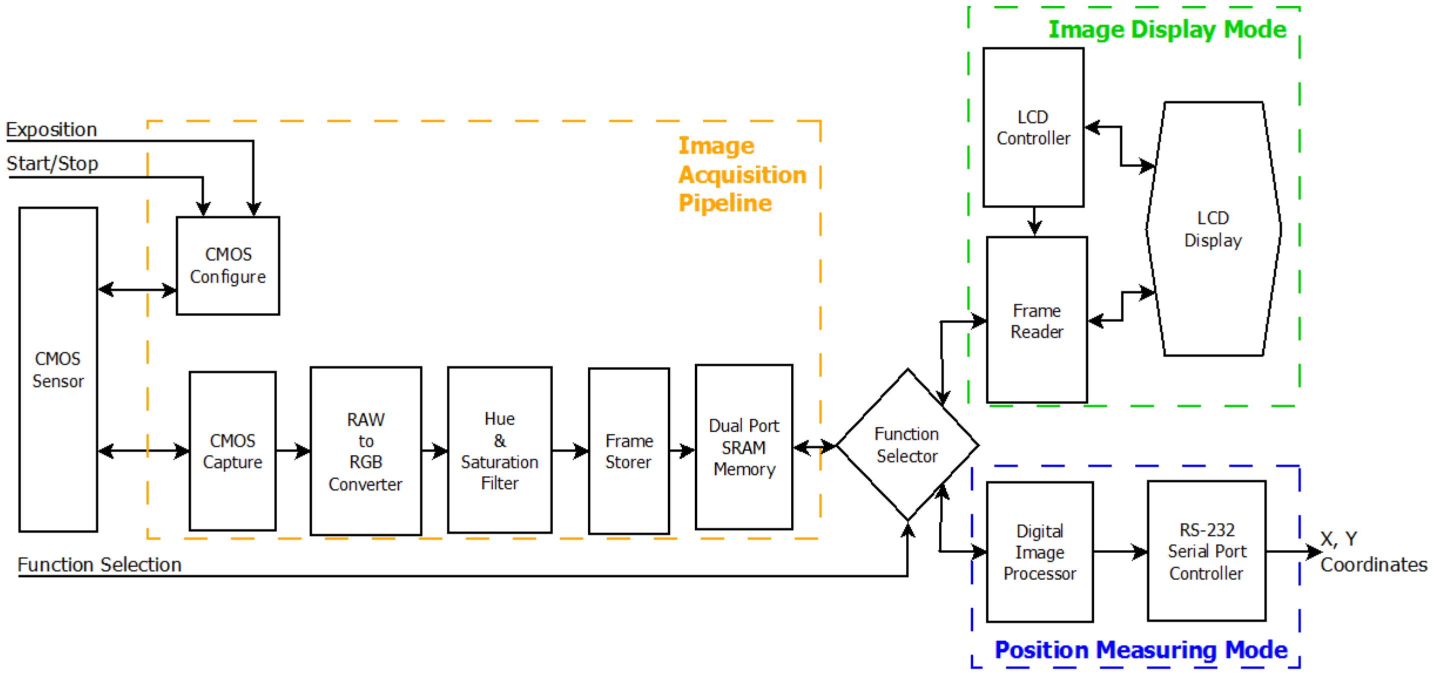


Figure 6.2. Modular diagram of the Computer Vision System designed as second solution.

The processing stage is depicted in detail in fig. 6.4.This section is controlled by a Master FSM that globally controls the main blocks as Read & Sum, Convolutioners and UART Transmission. This last block is controlled by a slave FSM that controls the protocol for transmission of two bytes at a time. The core of the transmission code was developed by [12] and suffered no changes. A slave FSM was designed to control this core and to make it send two bytes per command from Master FSM.

6.2.1. Hue and Saturation Filter

This is one of the key modules of the system because it executes the whole color detection. Its design is based on the color filtering proposed by [18] explained briefly in section 4.2.2 with several modifications in the hue channel, while the saturation level was kept as proposed.

The system was devised to work with up to seven different colored balls, so for the color filtering in hue the basic six colors (red, yellow, green, cyan, blue, and magenta) plus orange were selected (see fig. 6.3). The saturation filter is set to restrict two levels: S=0.50 and

S=0.25 as depicted previously in image 4.6. Based on the conditions of table 6.2, the regions for the saturation threshold are shown in eq. 6-1 and 6-2, and the whole regions are shown in fig. 6.3.

$$S_{50} = \sim(A|B|C|D|E|F|Q) \quad (6\text{-}1)$$

$$S_{25} = \sim(A|B|C|D|E|F|Q) \quad (6\text{-}2)$$

To be able to make the comparison of each pixel against this strict threshold in RGB color space, the values are compared according to the plane equations shown in fig. 6.3. Note that the planes shown in image a) work for both red and cyan colors. This happens since these colors are opposite in the symmetric RGB color space; so the inequation of the pixel against the planes is just inverted. The same situation happens between green and magenta, and blue and yellow.

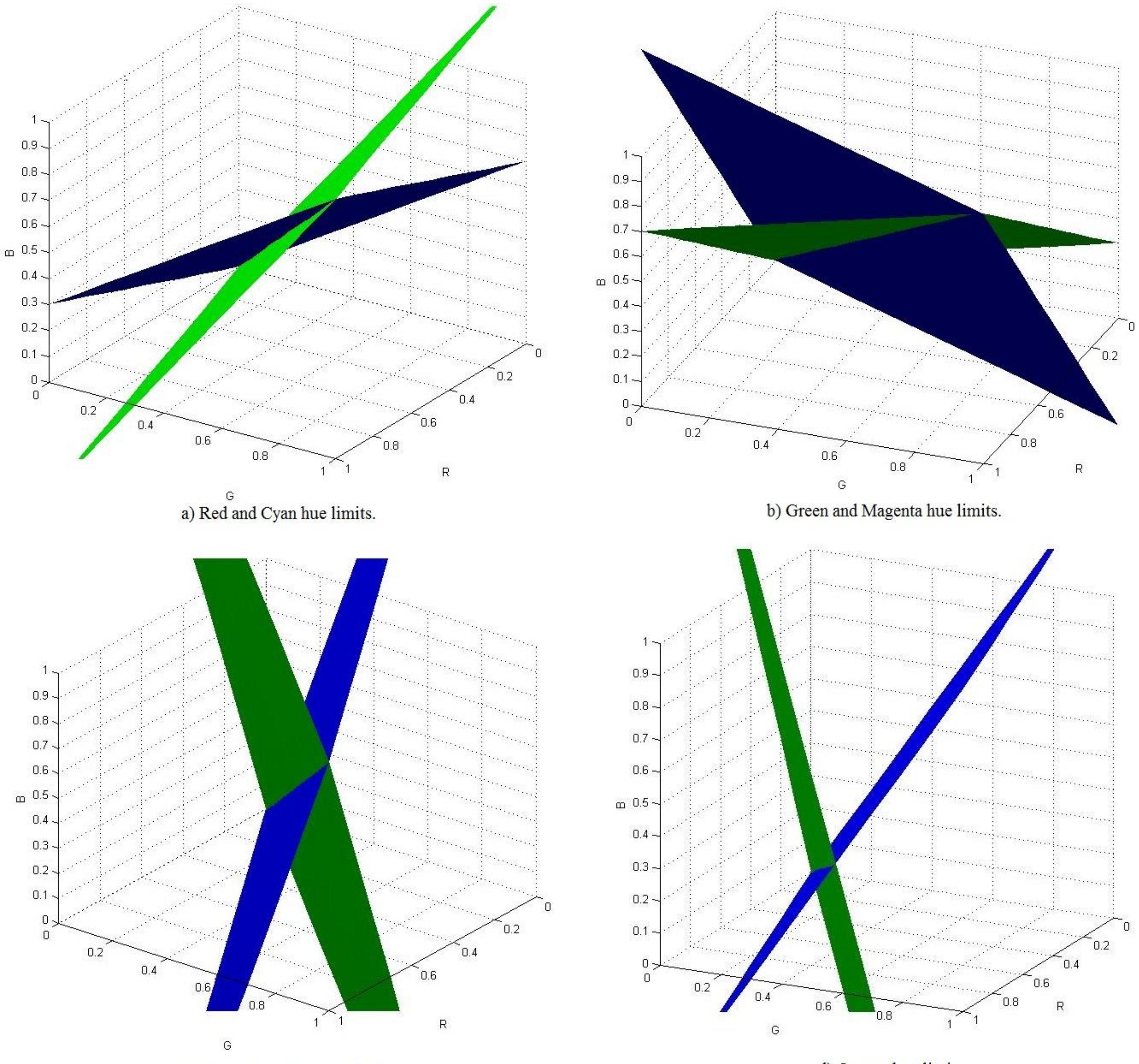
Table 6.2. List of pixel channels constraints in order to comply with different levels of saturation.

N	**S>0.50**	**S>0.25**
A	(R-B<B) & (2*G-R<R)	(3*B-3*G<G)& (4*R-3*B<B)
B	(G-B<B) & (2*R-G<G)	(3*R-3*B<B)& (4*G-3*R<R)
C	(G-R<R) & (2*B-G<G)	(3*G-3*B<B)& (4*R-3*G<G)
D	(B-R<R) & (2*G-B<B)	(3*G-3*R<R)& (4*B-3*G<G)
E	(B-G<G) & (2*R-B<B)	(3*R-3*G<G)& (4*B-3*R<R)
F	(R-G<G) & (2*B-R<R)	(3*B-3*R<R)& (4*G-3*B<B)
Q	(R>512)&(G>512) &(B>512)	(R>768)&(G>768) &(B>768)

The final stage of the hue and saturation filter assigns a 3-bits code to each pixel instead of their original RGB values if it meets any of the hue and saturation filters for the configured colors. If the pixel does not belong to any of the predefined colors, the assigned code will be zero. The codes assigned for the set conditions are shown in table 6.3. The hue conditions have been scaled to avoid negative decimal numbers handling; while the saturation constraints are shown as they were mapped.

6.2.2. Frame Storer

As its name says, takes the pixels and address from CCD Capture to decode them through the Hue and Saturation Filter according to the colors we desire to use.



a) Red and Cyan hue limits.

b) Green and Magenta hue limits.

c) Blue and Yellow hue limits.

d) Orange hue limits.

Figure 6.3. Hue filters used to detect colors. **a)** Red and cyan threshold planes. **b)** Green and magenta threshold planes. **c)** Blue and yellow threshold planes. **d)** Orange threshold planes (lower plot).

6.2.3. Frame Buffer

Is a 32 bit wide memory that has two ports: one read and one write port. It covers two necessities at the same time: it stores the image captured to make possible its analysis and it is also useful to separate the 144 MHz clock domain (PIXCLK) from the 50 MHz clock (CLOCK_50) when position measuring mode is selected, and the 40 MHz clock when image display mode is selected (CLK_LCD).

Table 6.3. Hue and Saturation conditions and the code to be assigned if the evaluated pixel meets the requirements.

Color	Hue Constraint	Saturation Constraint	Assigned Code [bin]
Red	$3 \cdot R + 7 \cdot G > 10 \cdot B$ & $10 \cdot G < 3 \cdot R + 7 \cdot B$	S > 0.50	001
Green	$3 \cdot G + 7 \cdot B > 10 \cdot R$ & $10 \cdot B < 3 \cdot G + 7 \cdot R$	S > 0.25	010
Blue	$3 \cdot B + 7 \cdot R > 10 \cdot G$ & $10 \cdot R < 3 \cdot B + 7 \cdot G$	S > 0.25	011
Yellow	$3 \cdot B + 7 \cdot R < 10 \cdot G$ & $10 \cdot R > 3 \cdot B + 7 \cdot G$	S > 0.25	100
Cyan	$3 \cdot R + 7 \cdot G < 10 \cdot B$ & $10 \cdot G > 3 \cdot R + 7 \cdot B$	S > 0.25	101
Magenta	$3 \cdot G + 7 \cdot B < 10 \cdot R$ & $10 \cdot B > 3 \cdot G + 7 \cdot R$	S > 0.50	110
Orange	$3 \cdot R + 7 \cdot B < 10 \cdot G$ & $10 \cdot G < 3 \cdot B + 7 \cdot R$	S > 0.25	111
Default (none)	-	-	000

6.2.4. Function Selector

This module is basically a multiplexor for the inputs basically a multiplexor for the inputs and outputs of the read port of the Frame Buffer. The main purpose for this module is to alternate between Image Display mode or Position Measuring mode as the user desires.

6.2.5. FSM Cerebro

This is the Master Finite State Machine (FSM) that controls all the process to perform the position measurement task. It has been designed with 16 states that control the principal modules of the computer Vision system, besides the sum, convolution and transmission FSMs. The state diagram is depicted in Appendix A, including the inputs and outputs.

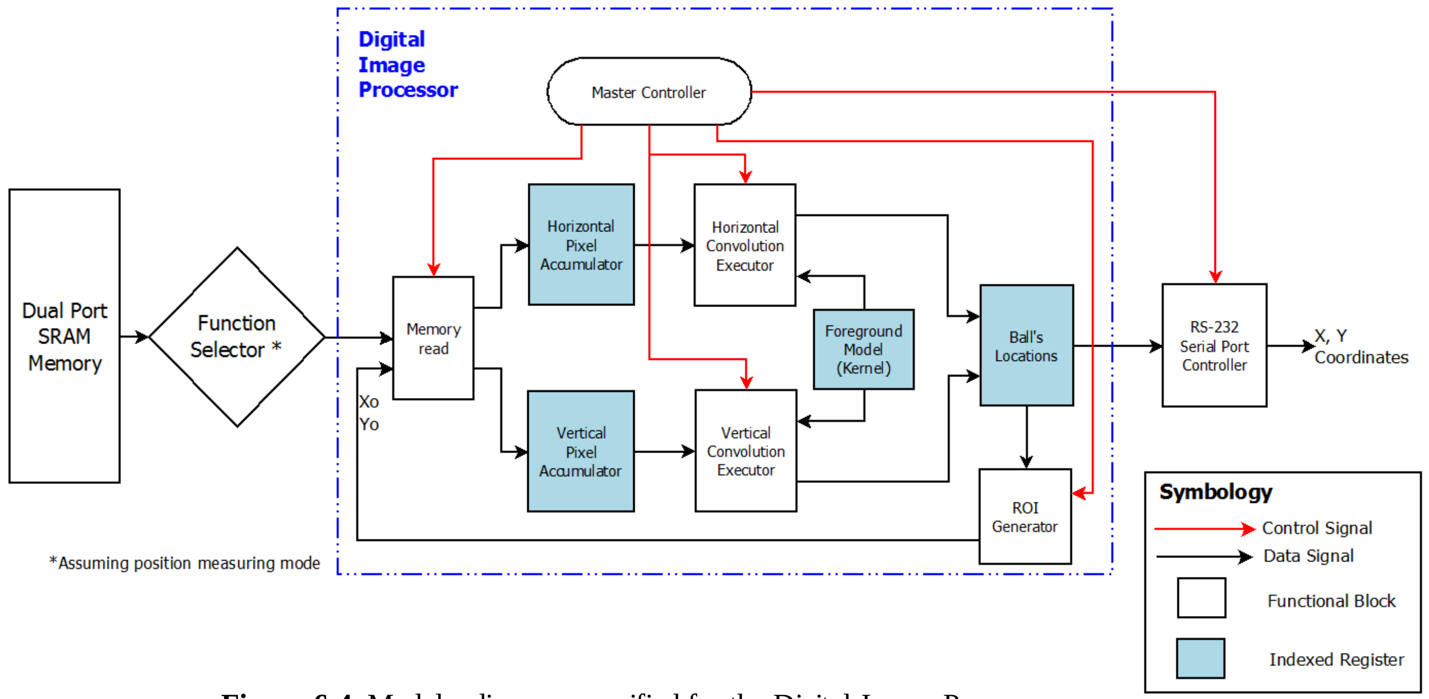


Figure 6.4. Modular diagram specified for the Digital Image Processor.

6.2.6. Memory Read

This module performs the double task to read the pixels from memory and count the quantity of the same color inside of each row and column. These results are stored in two 8-bits wide RAM memories modeled as array registers of dimensions 150x6 (Pixel accumulators in image 6.4). These memories are the center of the process because after run the ROI, these results are used in the convolution modules to identify the round shape we're looking for. This module is controlled by a slave FSM, whose state diagram is depicted in Appendix C, including the inputs and outputs.

6.2.7. Convolution Executor

This module is used twice in the design, and it is controlled by a slave FSM, whose state diagram is depicted in Appendix B. It is devised to apply the method described in chapter 4 for Foreground Modelling based on linear convolution. The kernel used for convolution is 101 pixels long and the values stored were established on a ball radius of 50 pixels (fig 6.5), which according to previous tests was an approximation of the size in the frame of the ball at the fixed distance of 64 cm from the CCD sensor to the surface of the platform. As labeled in the method, the maximum result of the convolution is stored with the position where the kernel is centered in the row or column which have the most pixels of each color. However, there is a requirement of detection of at least 88% of the ball area for the data to be recognized as a ball in the next stage.

6.2.8. ROI Generator

When Convolution has been performed, this modules prepares the new locations for the ROIs to be used in the next iteration. This values are used for both reading the memory and also for adjust the position of the found centers according to the initial coordinates.

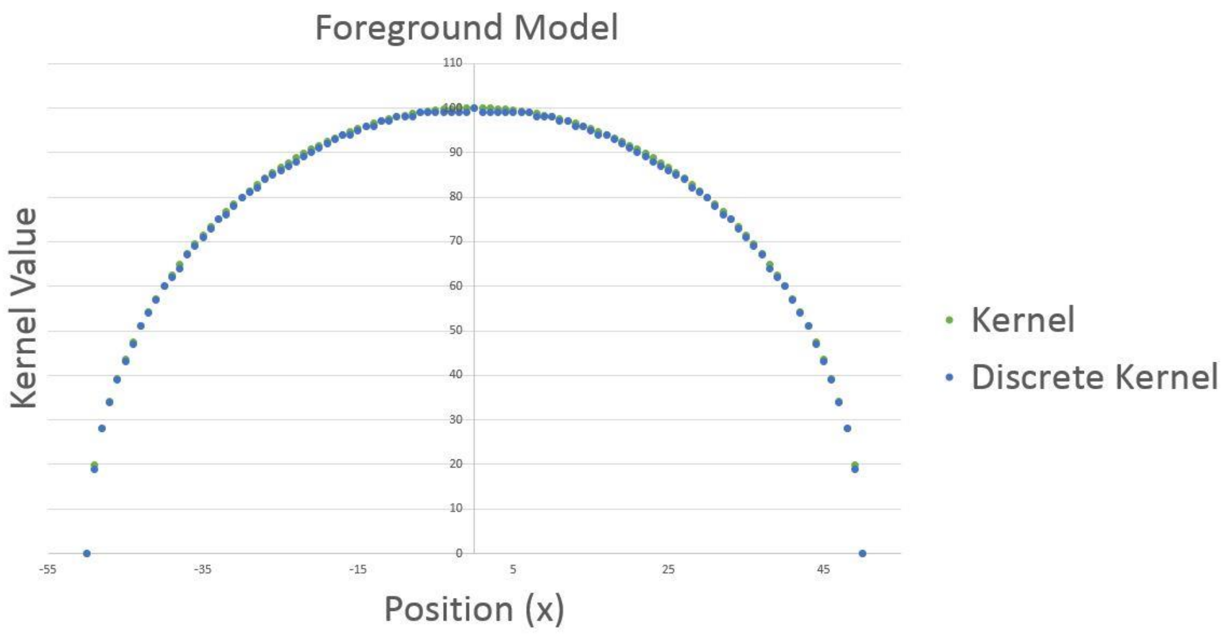


Figure 6.5. Kernel used as foreground model in the Convolution operation. Values according to equation 4-13 (green) and discrete approximation for practical purposes (blue).

6.2.9. UART Serial Port Controller

This module simply receives the data from Offset to be sent and transmits to the platform via RS-232 serial port at a baud rate of 115200 bps. The data is sent in on package of twelve bytes as maximum and two as minimum. The distribution of the location bytes are shown in table 6.4 in arrival order. The center coordinates are sent only when a ball has been successfully detected.

Table 6.4. Format of the data sent by the computer vision system.

Number	Name	Description	Data Format
0	Initialization	Its value is always 0x0FF. This value indicates that the system has read the memory and is starting to perform convolution	Init. Value [7:0]
1	ROI indicator	Specifies the Region of Interest that is being processed at the time	ROI [7:4] Zero [3:0]
2	Horizontal position	Contains the 8 LSB of the found position of the ball.	X_{LSB}[7:0]
3	Horizontal position Information	Includes the information of the Region of Interest and the color of the found ball. Also the MSB of the found position of the ball is in this byte.	Active ROI [7] ROI [6:4] Color[3:1] X_{MSB}[0]
4	Vertical position	Contains the 8 LSB of the found position of the ball.	Y_{LSB}[7:0]
5	Vertical position Information	Includes the information of the Region of Interest and the color of the found ball. Also the MSB of the found position of the ball is in this byte.	Active ROI [7] ROI [6:4] Color[3:1] Y_{MSB}[0]

6.3. System Initialization: Dynamic Regions of Interest

The system is programmed to follow two routines: the initialization routine and the regular routine. The initialization procedure is configured to analyze twelve Regions of Interest overlapped between them with the goal to find the first reference for the position of the circles in the acquired frame.

Since the diameter of the balls is between 80 and 90 pixels, the size of the ROIs is 150 x 150 pixels separated 75 pixels from one another, so is not possible for the ball to be completely in two ROIs at the same time. However, in the experimental results, the same ball could be detected twice during First Run due to this ROI overlap; so an additional restriction had to be made to prevent double detection. If a ball's center is closer than 30 pixels to the end of the ROI, it is better to be detected properly in another one only during First Run. Thus, the range where the results are considered as effective if the detected center is between 30 and 120pixels into the ROI. Image 6.6 illustrates this concept.

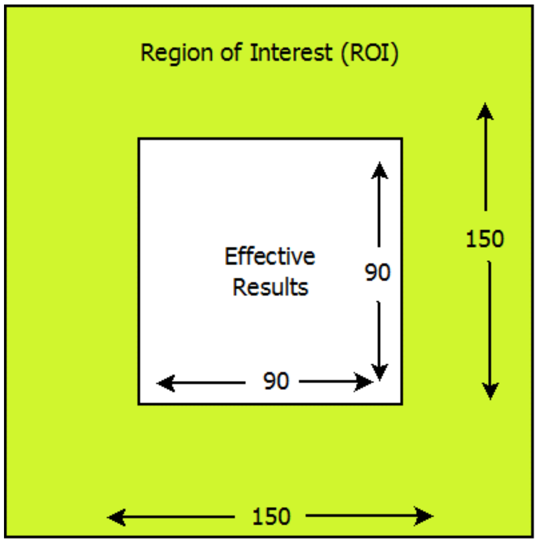


Figure 6.6. Illustration of one region of interest (yellow square) and the section where found centers are valid (white square).

The initial regions of interest are listed in the table 6.5. Every time the system finishes to analyze one of this sections of the image, if a ball is detected a new dynamic ROI with the code of the color is activated; being the initial location 75 pixels left and 75 up from the detected center. This ROI will be used after the calibration process. If the digital image processor finishes analyzing the twelve ROIs and no ball is found, it would need a manual reset.

After this first run, the system only works with the regions where a ball was located during the initialization routine. All regions found can move independently and their starting position will be always 75 pixels left and 75 up from the detected center in the previous iteration. Therefore, the analyzed section of the image varies with the position of the ball, resulting in a time-saving procedure given that it allows not to scan the whole frame each iteration.

Table 6.5. Initial coordinates for the different regions of interest used to initialize the computer vision system.

N	Initial X	Initial Y
0	0	0
1	75	0
2	150	0
3	225	0
4	0	75
5	75	75
6	150	75
7	225	75
8	0	150
9	75	150
A	150	150
B	225	150

Chapter 7. Result Analysis

After explaining extensively how the two versions of the computer vision system work, is logical to measure the obtained performance for each type to see which one is better. In this chapter are going to be presented first the results separately for the characteristics that cannot be compared and then a final comparison.

7.1. Heterogeneous computer

In the constant search for maximum performance, the heterogeneous computer developed to measure the position of the balls has several versions itself. This happened in an iterative process, until it was decided that this instantiation was obsolete and had no further practical use. However, it was a powerful tool to understand some properties of the tPad camera as the scalability of the frame, the effect of changing the exposure time and the values of the RGB channels gain for the CMOS sensor. The flexibility provided by the Nios II processor also allowed to test different ways of image segmentation programmed in C language, which meant an important influence to the selection of the algorithm used in the definitive design of the dedicated hardware.

The first tests done with the tPad were to apply the procedures depicted in the user manuals such as [7, 21, 22 and 23] with the goal of acquire the image and display it on the LCD screen. After the instantiation and set up of the whole system described in the section 6.1, one of the first obstacles was to adjust correctly the snap to the screen. The camera was configured to row size of 600 and column size of 800 due to the time it takes to acquire the image; but after the raw to RGB resampling, the resolution is reduced to 400x300. The tPad screen has a frame size of 800x600, so each acquired pixel needs to be quadruplicated to fit the picture in the screen without flickering. Besides, the Pixel Buffer DMA controller works independently from the Avalon Memory-Mapped Converter, so if the processor is not continuously moving the frame from the Converter to the Pixel Buffer, the screen would always show either the same picture or the uninitialized memory (usually randomly colored dots).

Image 7.1 illustrates the evolution of the adjustment of the screen: a) shows the result of programming the camera to resolution different than standard (800x600); b), c) and d) shows the frame when the camera is programmed to standard resolution with the difference that b) has an incorrect location assigned by the processor; c) shows correct location without pixel quadruplicating and d) shows the correct adjustment to fit the image on the screen. For the next pictures, the used adjustment is the same that delivered the result shown in fig 7.1-d).

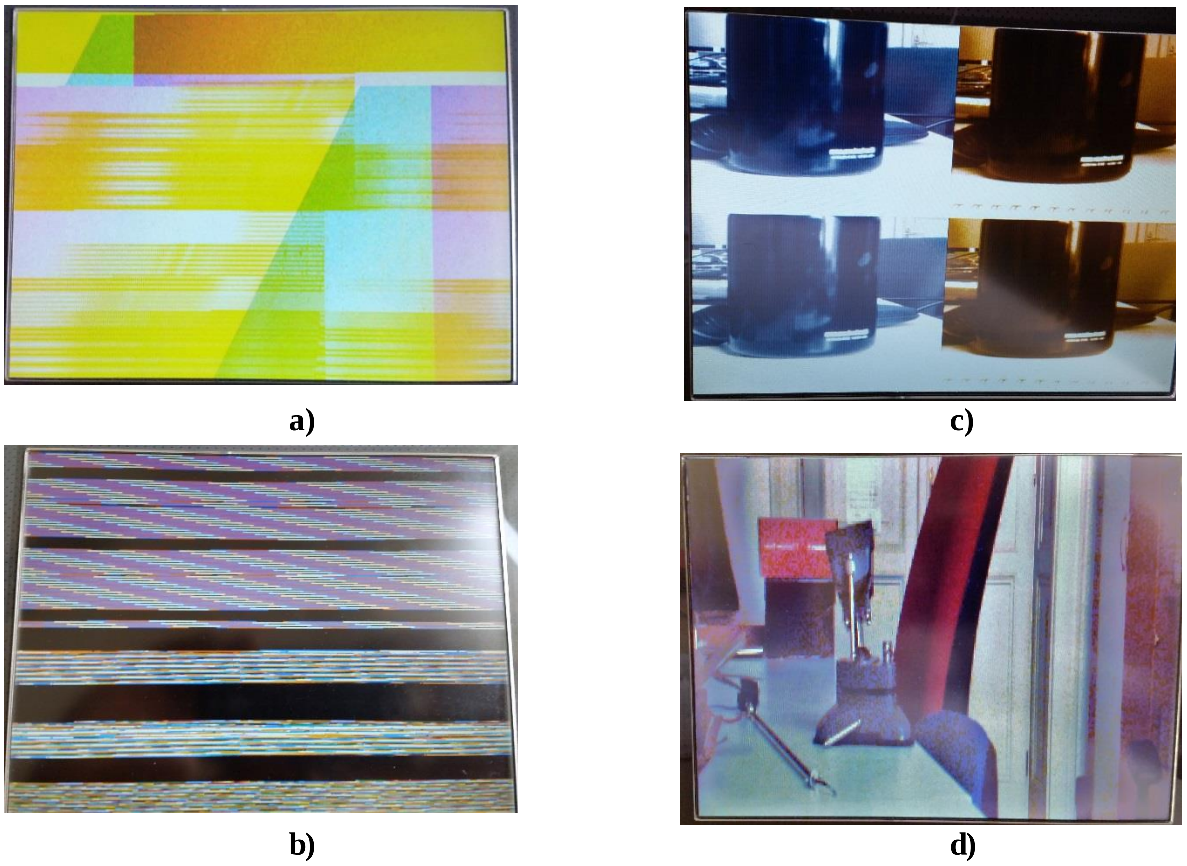


Figure 7.1. Evolution of location adjustment from Avalon Memory-Mapped Converter to Pixel Buffer memory with **a)** resolution different than 800x600; **b)** standard frame size with an incorrect location assigned by the processor; **c)** correct location without pixel quadruplicate; and **d)** shows the correct adjustment to fit the image on the screen

After being able to acquire and display correctly the image, the one issue that was remarkably notorious was the response time of the system. For example, when an object was moved in front of the camera, there was a delay in the LCD display that was notorious even to the naked eye. So, in order to have a real measure of this delay, the next step was to adjust the hue and saturation filters explained in section 6.2.1, implement them via software using the Software build tools for Eclipse, to finally calculate the center of mass as first approach. The second version of the heterogeneous computer, the hue and saturation filtering was done via hardware, and the position was still measured as the center of mass. The third version of this system featured simple thresholding color filtering (as explained in section 4.2.3) via software with center of mass calculation.

With the three different versions of the system, it was time to prove which one was the best for the assigned task. Preliminarily the purpose of the first two versions was to compare the effectiveness of the image segmentation via hardware and via software, and the third was aimed to be the fastest but the least robust. The test was simple: take 500 similar pictures and calculate the average time it takes the computer to process the entire frame. Additionally, the system wouldn't show the picture in screen in order to save instructions and therefore, time. The results are shown in table 7.1. It is important to highlight the fact that the results only include processing time, they don't include the camera exposition time nor the storage in Dual Clock FIFO memory.

Table 7.1. Time needed in average to process one frame using different versions of the Heterogeneous Computer.

Version	Description	Number of Clock Cycles	Time [ms]	Avg. Freq. [Hz]
First	Software HSV color filtering with center of mass calculation	6033318	120,66	8,29
Second	Software RGB color thresholding with center of mass calculation	3097705	61,95	16,14
Third	Hardware HSV color filtering with center of mass calculation	2970132	59,40	16,83

As the reader can infer from table 7.1, even in the best case scenario, the time to process one frame is not enough to achieve 50 fps. So, at this point it was possible to compress the image as it was received in order to reduce the instructions needed to move one frame from the Avalon Converter to the Pixel buffer, but it would carry the problem of the number of cycles it takes to Nios II processor to execute one instruction. Another option was to increase the input clock frequency but the Quartus II tool prevented strongly against this because of safety and results repeatability. So the final decision was to declare this approach as obsolete, keep the hue and saturation filters in hardware and develop from scratch the processing stage that would detect the circles in a stored frame and calculate their centers. The new system would most likely to be controlled by one or several Finite State Machines (FSMs) instantiated in Verilog HDL.

7.2. Dedicated Hardware

After implementation and instantiation, the computer vision system was having some functionality problems that were not perceptible in the simulations. This meant some extra-dense debugging time that required evaluations in implementation, scenery and timing constraints in the design. In the following sections the behavior of the system will be exposed and the corrections done will be justified. The original artificial illumination was provided by two fluorescent lights placed at the same distance as the camera, powered by the regular electric supply.

7.2.1. Functionality and operating conditions

Most of the slack and routing problems were corrected by pipelining the different modules, but the system still failed to start properly in Position Measuring Mode. Fortunately, the Image Display Mode worked as expected almost from the beginning of the implementation, so this situation was helpful in the troubleshooting process. Some features were disabled to diagnose which ones were failing, and as a result of this procedure, the first tests were done under natural lightning with only one ball. Another obstacle was the compiling time: due to the size of the project, it would take at least 20 minutes to Quartus II to synthesize, place and route the whole project.

As mentioned before, the first tests regarding ball detection were aimed to find a single ball in both stationary and real-time frames. However, there were initial problems with the early stages: for some reason, the system was able to detect the ball in the first run, then it switched to the Dynamic ROIs and then, after a few iterations the track of the ball was lost. This happened with both balls, even when applying different illumination. At first this problem was partially solved lowering the minimum convolution value to consider valid a circle, but the issue persisted when finding the red ball. This subject was completely solved with the replacement of the deficient illumination, as will be explained later.

After single-ball tests, started the multiple balls experiments in real time, and then another problem popped out: detection under natural light was possible to track only the blue ball, while artificial light allowed to detect only the red one. After making lots of modifications to the gain in RGB channels of the camera in order to adapt it to this illumination, it was still not possible to detect both balls at the same time. Furthermore, the tracking of the red ball was lost after a few iterations because the lighting used at the time (fluorescent light bulbs) was powered by the European standard AC current, which works at 50 Hz. In summary, the small difference between the image acquisition and scene lighting frequencies was the cause for the monitoring loss.

Once this relationship was fund, it was undeniable the need to change the lighting to a system under a frequency that did not match the image acquisition rate; therefore white LEDs were chosen for their accessibility and because they are powered by DC. The new set of lights was mounted using Merkur Robotic Kit as depicted in figure 7.2 at a distance of 20 cm from the surface. As a support, natural light was blocked using window blinds. The Camera's RGB

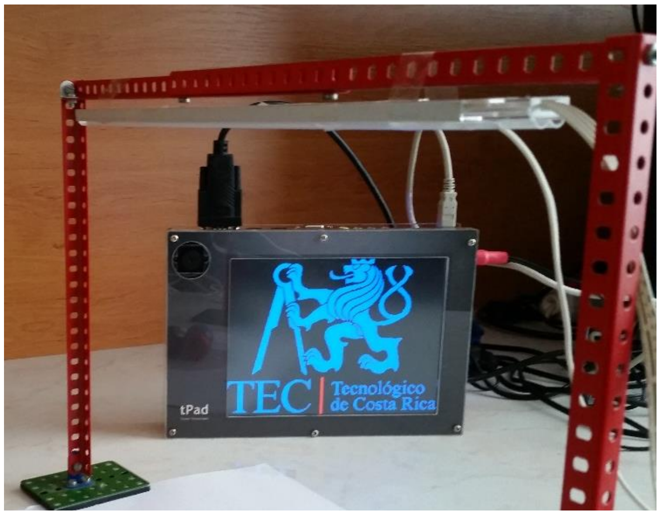


Figure 7.2. Illumination structure for the scene and welcome screen for the Computer Vision System.

channel gains were adjusted one more to this lighting and then the system was capable to detect both balls continuously at 50.25 [Hz].

7.2.2. Accuraccy and Response Time

After proving functionality and setting the definitive operating conditions, it was time to test the accuraccy and performance of the digital image processor based on Verilog hardware description language. This version proved to be a better implementation from the beginning because, unlike the Heterogeneous computer, there was no perceptible delay between the movement of the objects and its reaction in the screen when the system was operated in Image Display Mode.

A significant problem found in single-ball detection was an uncanny detection uncertainty even when the system was working with a static frame. As it can be seen in image 7.3, the found locations were different even for a motionless picture. Furthermore, this issue worsened when the image was acquired continuously because of the noise makes vary the quantity of identified pixels; therefore the error was propagated to both axes (fig. 7.4.). The source couldn't be the Memory Reader module because it has proved functionality, given it is commanded by a slave FSM and it has no slack glitches. The only problematic with signal

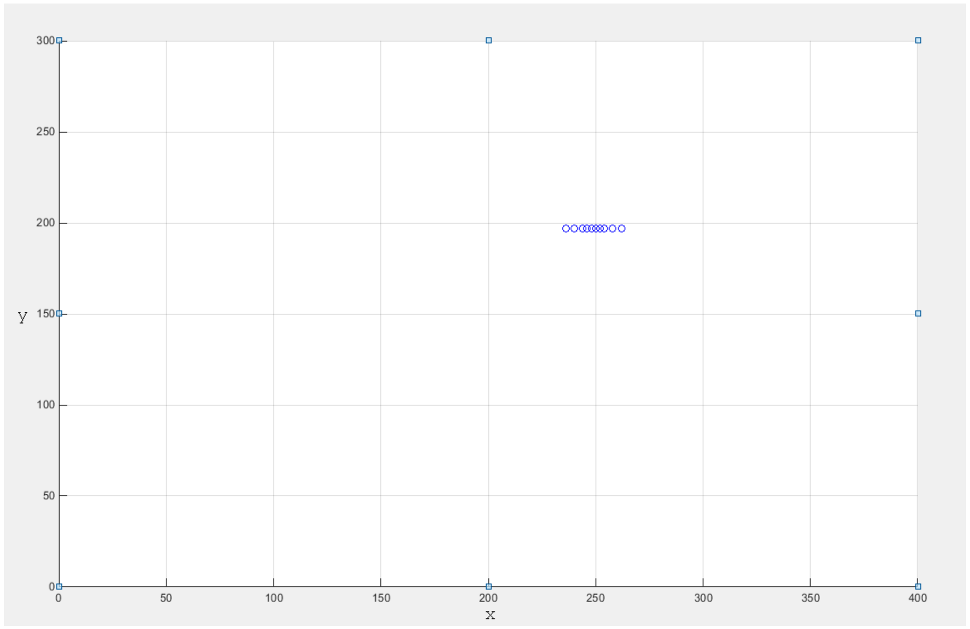


Figure 7.3. Received coordinates for a single-ball test using a static frame.

delay in the whole design can indirectly cause this uncertainty: slack between two nodes inside the Frame Buffer. This path cannot be segmented because it is inside one of the IP cores provided by Altera, and there wasn't any other slack issue on the paths connected to this memory, so there was not an actual tool to mend this problem.

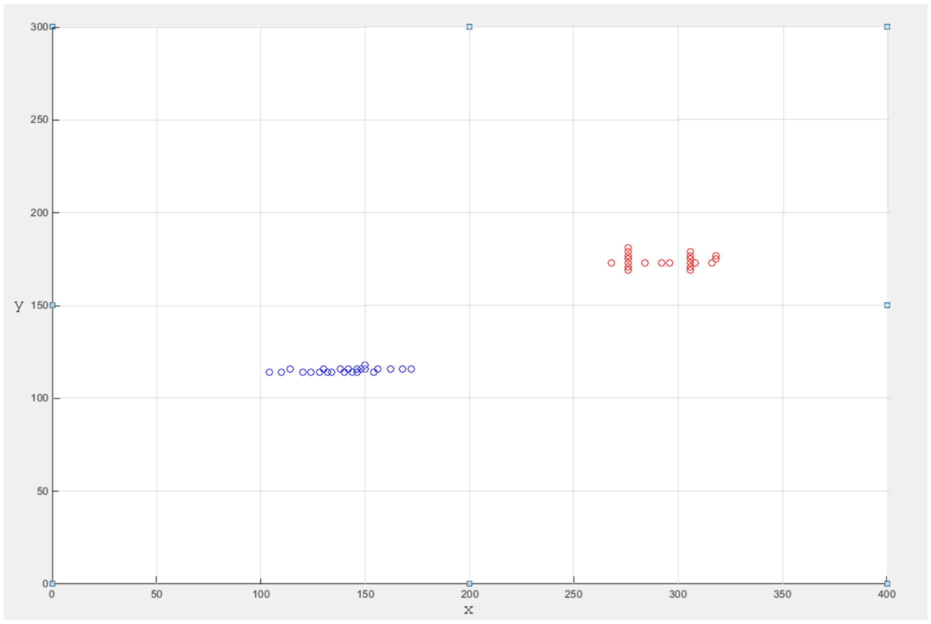
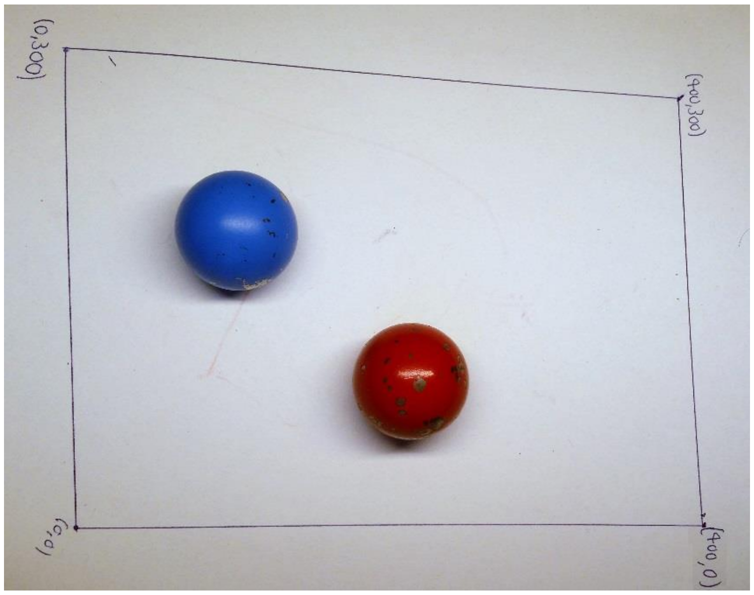


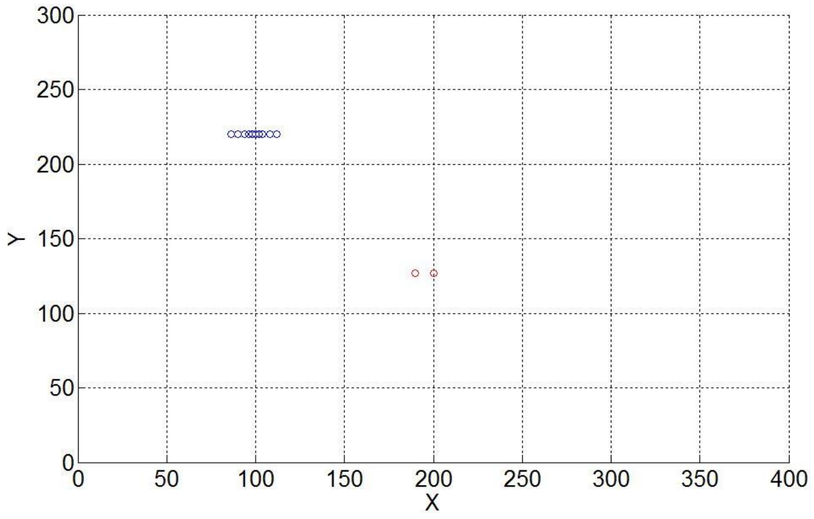
Figure 7.4. Received coordinates for a multiple-ball test using continuously acquired frames.

The Magman Platform was not available to do tests when the computer vision system was ready to work, however, in order to test roughly the validity of the acquired coordinates was devised: A sheet of paper was put under the camera, and the boundaries of the acquired frames were found using the screen mode (result is the background in image 7.5 a)). Two types of tests were done: with static and dynamic frames. Figure 7.5 shows the set-up for the test with a static frame (a)) and the results obtained (b)). Likewise, figure 7.6 shows the set-up for the test with a dynamic frame (a)) and the results obtained (b)). Looking at both results the accuracy difference between both modes is manifest. Unfortunately, to provide a statistic measure of the uncertainty the surface would have to be mapped to a rectangular shape using Homography; and the circumstances for these tests are not the same as the definitive scenario with the MagMan Platform, so this can be done as continuance for the project.

The performance of the computer vision system is measured in two parameters: the frequency of image acquisition and the frequency data is sent. As explained in section 6.1, the exposure time is set to roughly 16,6 ms; while the time to collect all pixels, filter them by color and store them in memory is around 3 ms. This is the same configuration for the Heterogeneous Computer, thus the image gathering rate is experimental result is 50,25 [Hz]. The picture acquisition rate is shown in real time in the tPad in hexadecimal format in the leftmost 7-segment displays. In the other hand, image segmentation time is shown in the following two displays, being typically at 182 packages per second. The typical time to perform this task is shown in table 7.2.

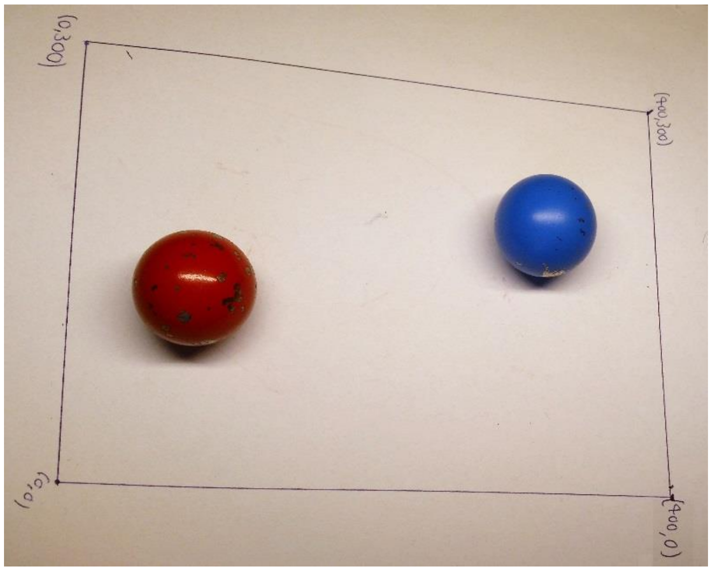


a) Scene with two balls for testing.

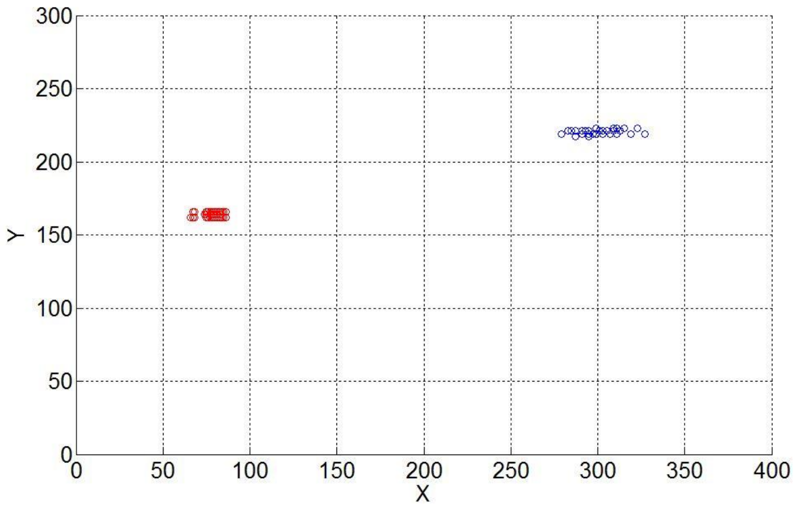


b) Coordinates received as locations for the scenario shown in sub-figure **a)**.

Figure 7.5. Multiple-ball test using a static frame: **a)** Set-up; **b)** Results obtained



a) Scene with two balls for testing.



b) Coordinates received as locations for the scenario shown in sub-figure **a)**.

Figure 7.6. Multiple-ball test using continuously acquired frames: **a)** Set-up; **b)** Results obtained

Since camera and processing stages work independently, and the second is almost four times faster than the first, there are some considerations to make. First: the system will send one location per colored ball in one frame within the time constraints only if the number of balls is equal to three. Second: if the number of balls is two or one, the computer will transmit more than one coordinate for the same ball in the same frame, so these values could be averaged in order to compensate the uncertainty explained previously. Third: if the number of balls is higher than three, the system will work at the same processing time but the frequency of data arrival for each color will be inversely proportional to the number of balls in the scene e.g. if there are six balls it will take two frames to transmit the locations of the six pellets, which could cause tracking loss. In conclusion, to ensure functionality including tracking and data arrival within requirements, the number of balls must not be higher than three.

Table 7.2. Typical execution time and frequency for the computer vision system based on dedicated hardware.

Stage	Tasks	Time [ms]	Frequency [Hz]
Img. Acquisition	Exposure Time Image capture Color filtering Storage in memory	19.900	50.25
Img. Segmentation	Read from memory Measure ball's position Adjust ROIs Transmit measured position	5.502	181.75

Despite the restriction explained in the preceding paragraph, performance in processing stage could be improved by modifying both Cerebro and Convolution FSMs and the circuitry they control. The Convolution Executor completes its task for every color in all active ROIs, indistinctly of First Run. This is necessary in the initialization process but not in the image segmentation because every active ROI is assigned to the code of the color it is supposed to track. Consequently, if the Convolution Executor can be modified to only track one color per ROI scan after First Run, the whole process of the image will have a significant boost in performance, given that convolution phase is the part that takes most of the time.

In general terms the project complies with the main objective, which was to implement a specialized computer vision system to measure the position of colored metallic balls. The main constraint was to make it work above 50 Hz because that's the rate the MagMan controller needs to maintain a stable control over the position of the balls. Regarding the specific objectives, the system does indeed recognize different colors (which can be modified), recognizes the circular shape of the ball and measures its center. The system can of course be improved, but considering this was supposed a short project of 16 weeks, and two versions of a computer vision system were designed, implemented and tested, is an important progress in terms of the MagMan Platform project.

Chapter 8. Conclusions and Recommendations

8.1. Conclusions

- Color detection is more effective in HSV colorspace than in RGB, at the expense of more processing.
- White LEDs as artificial illumination provides no stroboscopic effects.
- Digital image processing using a heterogeneous computer eases the implementation of algorithms at the expense of more execution time.
- It is not possible to provide a statistic measure of the system's uncertainty.
- The number of balls in the scene must not be higher than three.

8.2. Recommendations and Future work

- The Convolution Executor module can be modified to improve the computer's processing time.
- Detection's accuracy can be improved by averaging the received data.
- DC powered illumination is recommended due to the AC frequency matches the image capture frequency.
- The balls are recommended to be repainted to improve the accuracy of the computer.
- A Matlab program can be developed in order to acquire the transmitted position in real time.
- The dimensions of the surface can be mapped using Homography.
- The tests done can be repeated with the balls on the MagMan Platform.

Bibliography

[1] Advanced Algorithms for Control and Communications [aa4cc] (2013-may-28) Magnetic manipulator Magman and Matlab [Video file] Consulted at: https://www.youtube.com/watch?v=AhS_2gU1qW0

[2] Altera Co. (2011). *Nios II Hardware development tutorial.* Consulted at: https://www.altera.com/content/dam/alterawww/global/en_US/pdfs/-literature/tt/tt_nios2_hardware_tutorial.pdf

[3] Altera Co. (no date) *Nios II processor: the world's most versatile embedded processor.* Consulted at: http://wl.altera.com/devices/processor/nios2/ni2-index.html

[4] Altera Co. (no date) *Nios II/e core: Economy.* Consulted at: http://wl.altera.com/devices/processor/nios2/cores/economy/ni2-economy-core.html

[5] Altera Co. (no date) *Nios II/f core: Fast for performance-critical applications.* Consulted at: http://wl.altera.com/devices/processor/nios2/cores/fast/ni2-fast-core.html

[6] Altera Co. (no date) *Nios II/s core: Standard.* Consulted at: http://wl.altera.com/devices/processor/nios2/cores/standard/ni2-standard-core.html

[7] Ampire Co., LTD. (7, 2009) *Specifications for LCD module.* Consulted at: http://mail.terasic.com.cn/~wyzhou/tPad_v2.0.0_CDROM.rar

[8] Antunes, F. (2010) *IP camera on FPGA with a web server.* (Master Thesis) Universidade do Minho, Portugal.

[9] Bailey, D. G. (2011). Design for Embedded Image Processing on FPGAs. Hoboken, SG: Wiley-IEEE Press. Retrieved from http://www.ebrary.com

[10] CN blogs. "How to read CMOS from the Nios II's image on the SDRAM?" Consulted at: http://www.cnblogs.com/oomusou/archive/2008/08/31/de2_70_cmos_-controller.html

[11] Filip, J. (2015) *Extension of the control system for the magnetic manipulator with a non-flat surface.* (Bachelor's thesis) Czech Technical University in Prague, Czech Republic.

[12] Gomez, G. (no date) *UART transmission on the de2-115 board*. University of Nevada, Las Vegas, United States of America. Retrieved from: http://cmosedu.com/-jbaker/students/gerardo/Documents/UARTonFPGA.pdf

[13] González, R. and Woods, R. (2007) *Digital Image Processing*. (Third edition) New Jersey, U.S.A: Prentice Hall.

[14] Khan, S. A. (2011). Digital Design of Signal Processing Systems. GB: Wiley. Retrieved from http://www.ebrary.com.

[15] Mathworks (2013) *"Matlab and Simulink student design challenge"* Consulted at: http://www.mathworks.com/academia/student-challenge/spring-2013/

[16] OptoMotive, Mechatronics Ltd. (2011) *Velociraptor HS user manual*. Consulted at: http://www.optomotive.com/products/velociraptor-hs

[17] Raspberry Pi Foundation. (2016). *Raspberry pi 3 model b*. Consulted at: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[18] Simonian, A. (2014) *Feedback control for planar parallel magnetic manipulation*. (Diploma thesis) Czech Technical University in Prague, Czech Republic.

[19] Sundance DSP INC. (2013) *DSP-PCIe/104, Virtex-5 FPGA module with expansion card*. Consulted at: http://www.sundancedsp.com/carriers-/platforms/dsp-pcei104

[20] Terasic Inc. (2010) *Cyclone IV device handbook, Volume 1*. Consulted at: http://mail.terasic.com.cn/~wyzhou/tPad_v2.0.0_CDROM.rar

[21] Terasic Inc. (2010) *DE2-115 user manual*. Consulted at: http://www.terasic.com.tw/cgi/bin/page/archive.pl?Language=English&CategoryNo=165&No=502&PartNo=4

[22] Terasic Inc. (2010) *Terasic D5M hardware specification* Consulted at: http://mail.terasic.com.cn/~wyzhou/tPad_v2.0.0_CDROM.rar

[23] Terasic Inc. (2010) *Tpad user manual*. Consulted at: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=183&No=637

[24] Terasic Inc. (2013) *Video and Embedded Evaluation Kit - Multi-touch*. Consulted at: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&No=670

[25] Zemánek, J., and Hurák, Z. "*Feedback linearization approach to distributed feedback manipulation*." In *2012 American Control Conference (ACC)*, 991-996.Montréal, Canada: American Automatic Control Council (AACC), 2012 Consulted at: http://aa4cc.dce.fel.cvut.cz/content/distributed-manipulation-shaping-magnetic-field-magman-platform

[26] Zemánek, J., Tomáŝ M., and Hurák, Z. "Feedback control for noise-aided parallel micromanipulation of several particles using dielectrophoresis." *Electrophoresis* 36, no. 13 (2015): 1451-1458. Consulted at: http://aa4cc.dce.fel.cvut.cz/content/distributed-feedback-micromanipulation shaping-electric-field-dielectrophoresis

[27] Zywyn Co. (5, 2005) *ZT32xxE low power +3V to +5.5V, 250kbps RS232 transceivers.* Consulted at: bin/page/archive.pl?Language=English&Category No=165&No=502&PartNo=4

Appendices

Appendix A: Glossary

AC: Alternate Current

ASIC: Application Specific Integrated Circuit.

CCD: Charged Coupled Device.

CMOS: Complementary Metal Oxide Semiconductor.

CPU: Central Processing Unit.

DAC: Digital to Analog Converter

DC: Direct Current

DMA: Direct Memory Access.

FIFO: First Input-First Output.

FPGA: Field Programmable Gate Array.

FSM: Finite State Machine.

HDL: Hardware Description Language.

HSV: Hue Saturation and Value color space.

Img.: Image. Depending on the context it can mean either the pixel matrix on the system or the scene to be acquired by the camera. (also fig.)

LCD: Liquid Crystal Display.

LED: Light Emitting Diode.

LSB: Least Significant Bit.

MagMan: Planar Non-contact Magnetic Manipulation Platform. The device is described in chapter 1.

MSB: Most Significant Bit.

RAM: Random Access Memory.

RGB: Red, Green and Blue color space.

ROI: Region of Interest for scanning.

SDRAM: Synchronous Dynamic Random Access Memory.

tPad: Terasic Prototype of board that includes an FPGA, LCD screen and 5-Mega Pixel camera. The device is described in chapter 5.

USB: Universal Serial Bus.

VGA: Video Graphics Array.

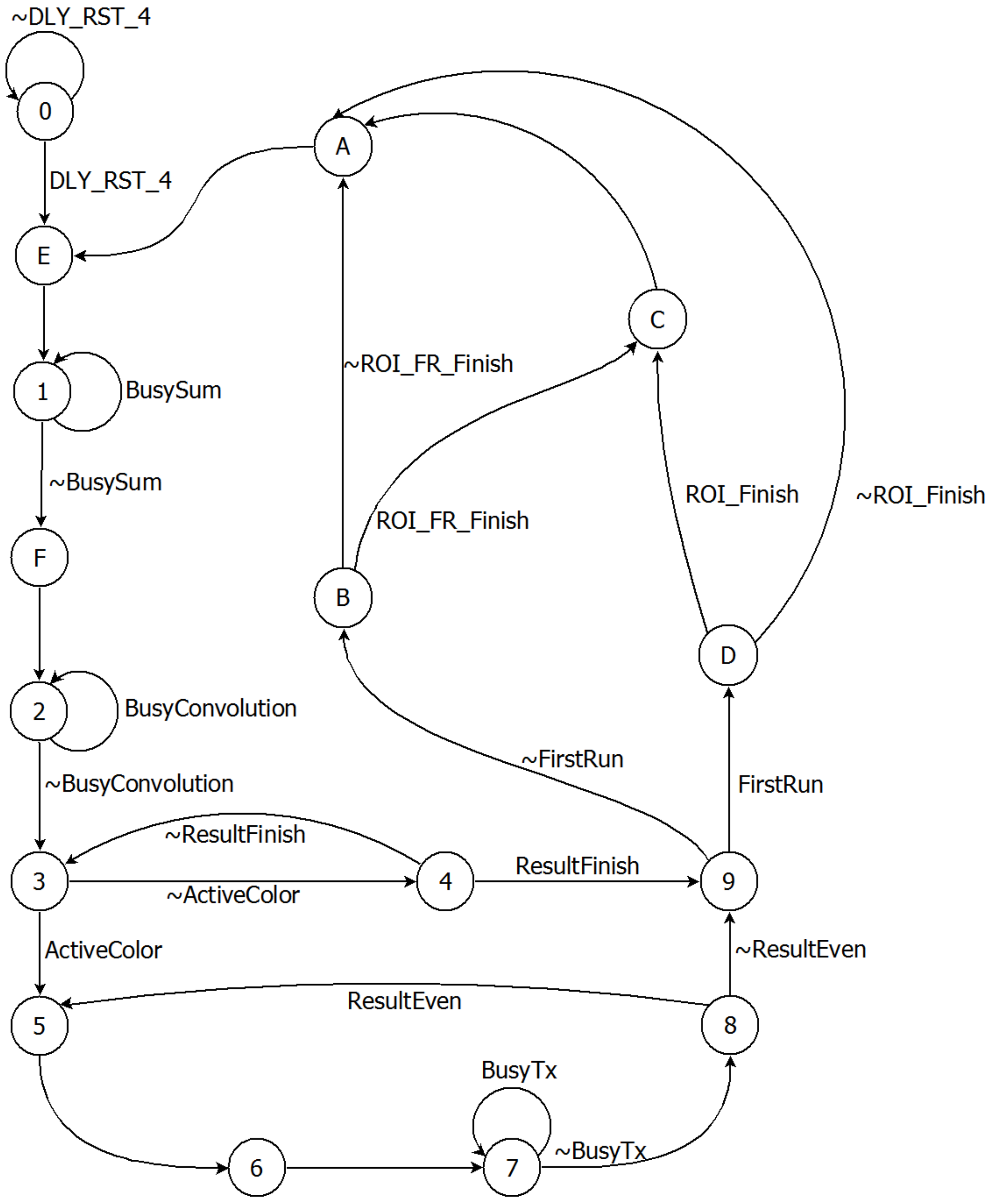
Appendix B: FSM Cerebro

Table B.1. Inputs, outputs and states for the FSM Cerebro

Current State	Condition (Input)	Next State	Output
0	~DLY_RST_4	0	ResetSum, ResetSumRegisters, ResetConvolution, ResetResultCount, ResetROICount.
	DLY_RST_4	E	
1	~BusySum	F	ResetConvolution, ResetResultCount.
	BusySum	1	
2	~BusyConvolution	3	ResetResultCount,
	BusyConvolution	2	
3	~ActiveColor	4	None
	ActiveColor	5	
4	~ResultFinish	3	EnableResultCount.
	ResultFinish	9	
5	none	6	ResetSumRegisters, EnableWriteMemories.
6	none	7	ResetSumRegisters, StartTx.
7	~BusyTx	8	ResetSumRegisters.
	BusyTx	7	
8	~ResultEven	9	ResetSumRegisters, EnableResultCount.
	ResultEven	5	
9	~FirstRun*	B	ResetSumRegisters.
	FirstRun*	D	
A	none	E	ResetSum, ResetSumRegisters.
B	~ROI_FR_Finish	A	ResetSumRegisters ResetConvolution, EnableROICount.
	ROI_FR_Finish	C	
C	none	A	ResetSumRegisters ResetConvolution, DisableFirstRun.
D	~ROI_Finish	A	ResetSumRegisters EnableROICount.
	ROI_Finish	C	
E	~(ValidROI|FirstRun)	9	StartSum ResetConvolution, ResetResultCount.
	(ValidROI|FirstRun)	1	
F	none	2	StartConvolution, ResetResultCount.

*FirstRun condition is set to negative logic.

Figure B.1. State Diagram for the FSM Cerebro.

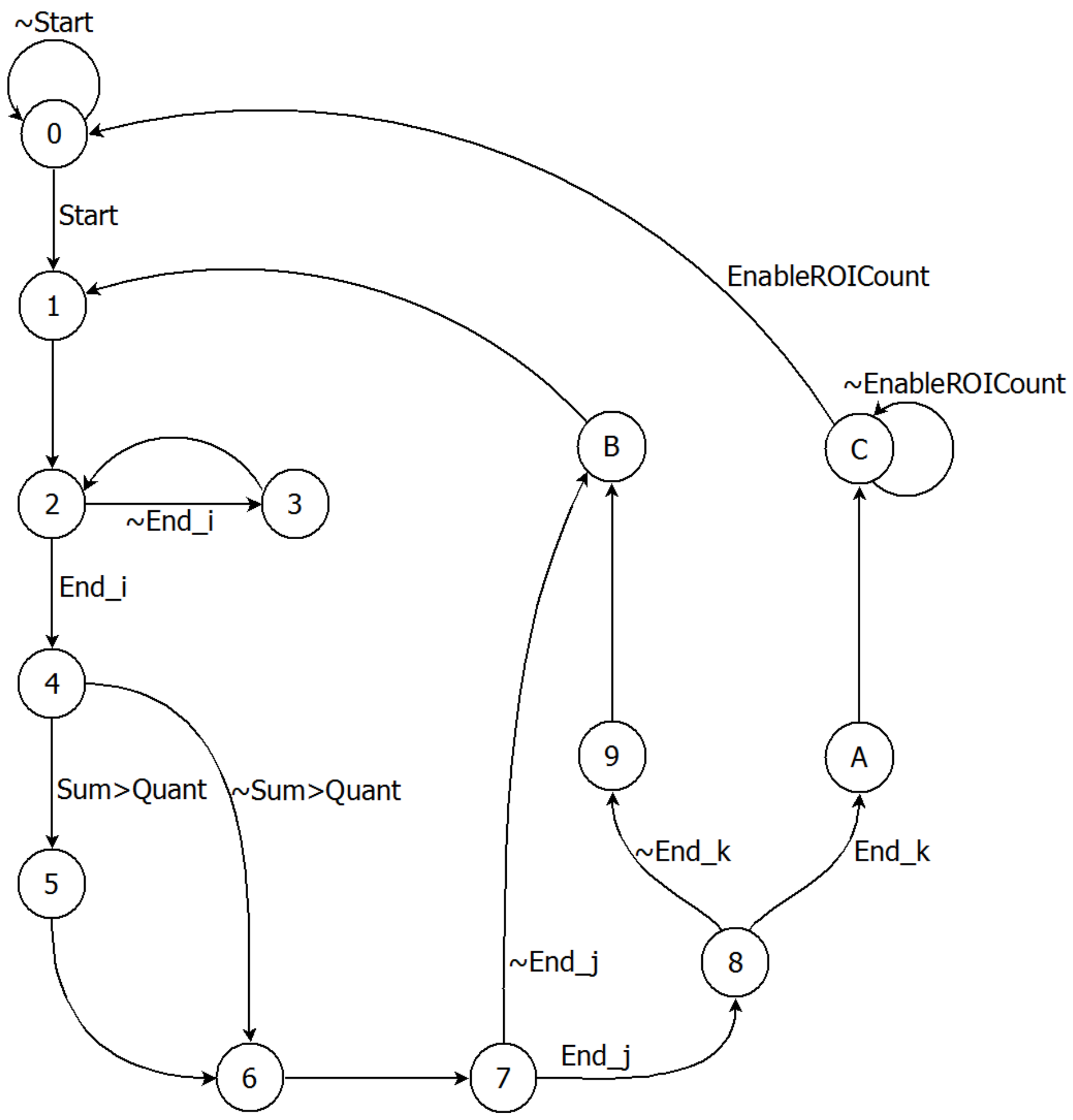


Appendix C: FSM Convolutioner

Table C.1. Inputs, outputs and states for the FSM Convolutioner.

Current State	Condition (Input)	Next State	Output
0	~Start	0	ResetSum Reset_i_Count Reset_j_Count Reset_k_Count ResetRegisters Enable_ij_adjust
	Start	1	
1	None	2	Reset_i_Count
2	~End_i	3	EnableSum
	End_i	4	
3	None	2	Enable_i_Count
4	~Sum>Quant	6	None
	Sum>Quant	5	
5	None	6	EnableRegisters
6	None	7	ResetSum
7	~End_j	B	Enable_j_Count
	End_j	8	
8	~End_k	9	None
	End_k	A	
9	None	B	Reset_j_Count Enable_k_Count Enable_ij_adjust
A	None	C	Reset_j_Count End
B	None	1	Enable_ij_adjust
C	~EnableROICount	C	None
	EnableROICount	0	

Figure C.1. State Diagram for the FSM Convolutioner.

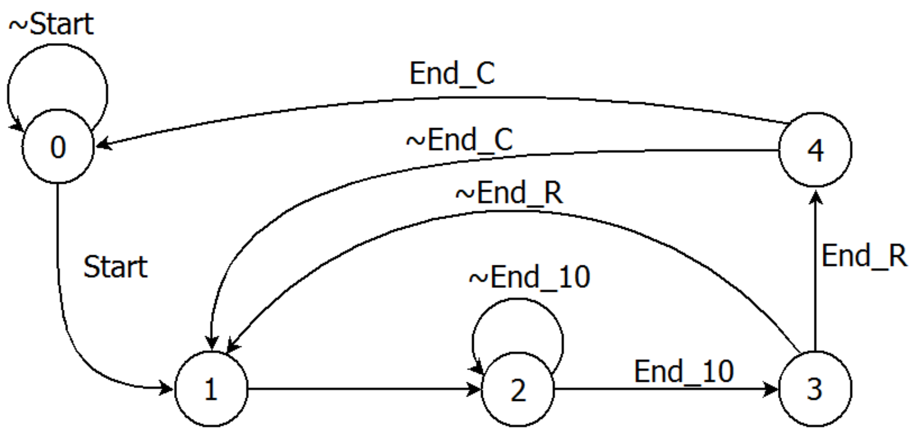


Appendix D: FSM Read & Sum

Table D.1. Inputs, outputs and states for the FSM Read & Sum

Current State	Condition	Next State	Output
0	~Start	0	Reset_10_Count Reset_R_Count Reset_C_Count
	Start	1	
1	None	2	EnableMemoryRead Busy
2	~End_10	2	Enable_10_Count Busy
	End_10	3	
3	~EndR	1	Reset_10_Count Enable_R_Count Busy
	EndR	4	
4	~EndC	1	Reset_R_Count Enable_C_Count Busy
	EndC	0	

Figure D.1. State Diagram for the FSM Read & Sum



Appendix E: Computer Vision System User Manual.

The document starts on the next page

Instituto Tecnológico de Costa Rica

České Vysoké Učení Technické v Praze

Department of Control Engineering,
Faculty of Electrical Engineering

Optical system for measuring position of metallic colored pellets on a platform.

User Manual

Robert J. Barnes Pérez

Prague, Spring Semester 2016

Abstract

This document is the user manual for a computer vision system for measuring the position of colored balls instantiated in a Terasic tPad. First, an overview of the device's interface will be given and then will be a description of how to set it up on its traditional purpose. Second, there is a guide through the hue filter if the user wants to change the colors to identify. Third, overviews the camera configuration, in order for the user to be aware of the conditions in which the image is continuously acquired.

Keywords

FPGA, Computer vision system, digital image processing, Terasic tPad, camera, Hardware description Language (HDL), Verilog.

Resumen

Este documento es el manual de usuario para un sistema de visión computarizado para medir la posición de balines coloreados, instanciado en un Terasic tPad. En primer lugar, se da una descripción general del dispositivo para luego explicar cómo configurarlo para su propósito tradicional. En segundo lugar, se da una guía a través del filtro de matices si el usuario quiere cambiar los colores a identificar. En tercer lugar, se describe la configuración de la cámara, con el fin de que el usuario sea consciente de las condiciones en las que la imagen se adquiere de forma continua.

Palabras Clave

Arreglo de Compuertas Programables (FPGA), Sistema de visión computarizado, Procesamiento digital de imágenes, Terasic tPad, Cámara, Lenguajes de descripción de Hardware, Verilog.

General Index

Chapter 1. System Overview

The Terasic tPad is a design environment capable to perform processing based systems. It is composed by the DE2-115 development board attached to a multimedia daughter card via HSMC port. The core of the DE2-115 board is the Cyclone IV FPGA, where all logic is performed. The tPad and Cyclone IV FPGA technical specifications are presented in table 1.1, and table 1.2 respectively.

Table 1.1 Technical specification of tPad board. **[6]**

Feature	**Value [unit]**
Operation Voltage DC	12 [V]
Flash Memory	8 [MB]
EEPROM	32 [Kbit]
SRAM	2 [MB]
SDRAM	128 [MB]
Clock frequency	50 [MHz]
Width	160 [mm]
Large	223 [mm]
Depth	47 [mm]

The board is to be configured via USB blaster, but it supports JTAG AS configuration as well. The board is shown on the top side with the FPGA and its peripherals in figure 1.1; and fig. 1.2 shows its back side, including a 5 Mega Pixel camera with an LCD 8" touch screen.

Table 1.2. Resources of Cyclone IV EP4CE115 FPGA. **[3, 4]**

Logic Elements [LEs]	114480
Embedded Memory [kbits]	3888
Embedded 18x18 multipliers	266
General Purpose PLLs	4
Global Clock Networks	20
User I/O Banks	8

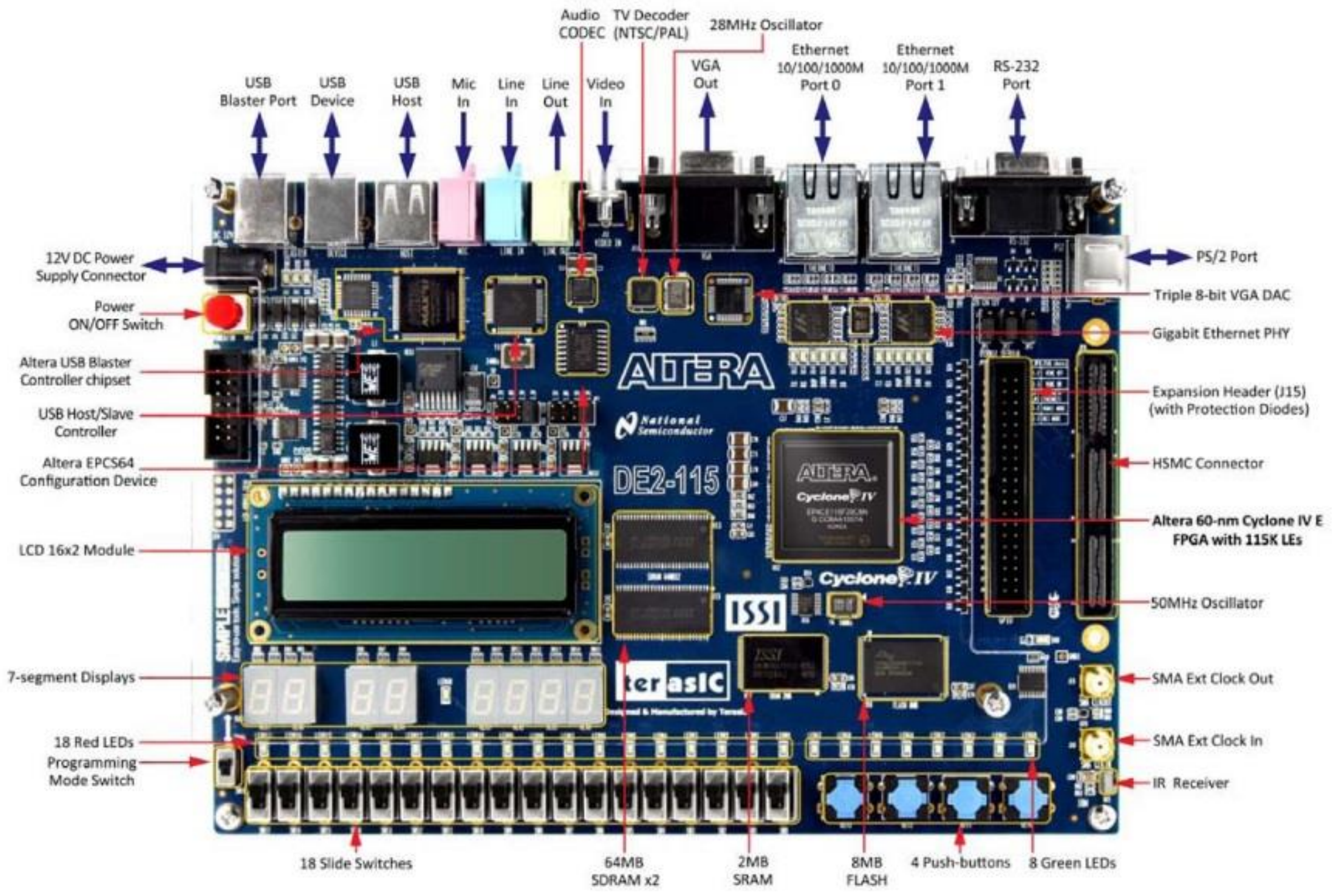


Figure 1.1. Altera Terasic tPad (bottom view). [4]

On the top side, the board has 9 green and 18 red leds for output or display; it also has 18 slide switches (SW), 4 push buttons (KEY) for inputs and 8 7-segment displays (HEX). Other peripherals are available but are not used for the application described in this document. For full device description please refer to [6].

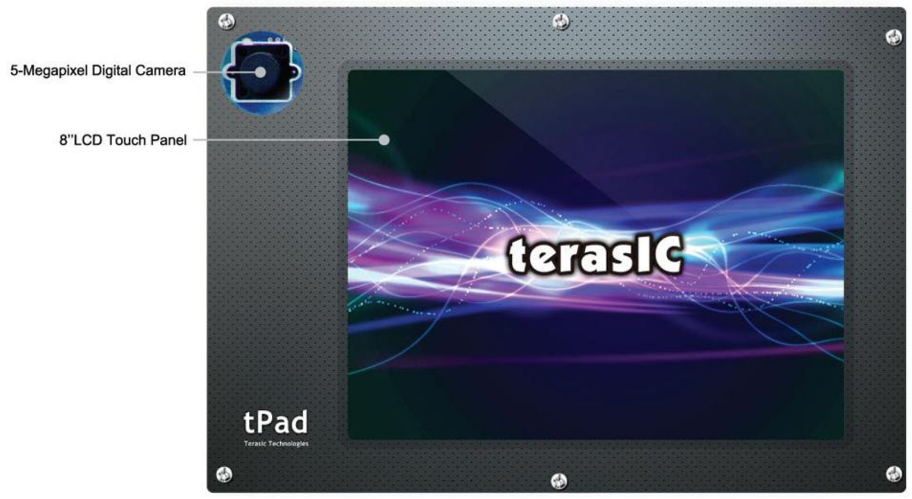


Figure 1.2. Altera Terasic tPad (top view). **[6]**

1.1. **Computer Vision System**

This section describes briefly how the whole system works for the user. The Video-In pipeline is depicted in the image 1.3 inside the orange square while the tasks for the mainframe are inside the blue square. The different modules that compose the whole diagram in fig 1.3 are fully described in chapter 6 of [1].

The camera is programmed by the CMOS Configure module, and the incoming pixels are arranged in CMOS Capture. Since the image is required to be in RGB format, the conversion is done in Raw to RGB module. Transformed pixels are received by the Saturation Filter, where their RGB values are evaluated in accordance with the HSV thresholding in RGB colorspace described in chapter 4 of [1] in two saturation levels (0.25 and 0.50) and seven possible colors plus black denoting background. Each pixel is then decoded in three bits according to color and stored in the dual port RAM memory.

In the output, the read signals and output from Frame Buffer are multiplexed in order to fulfill both operation modes of the computer vision system: screen or position measure. The idea is to be able to align the visual space of the camera using the screen mode and then to turn it to Position Measuring Mode with the sliding switch SW[2]. The screen mode also allows to show the pixels that are understood as foreground, which can be taken as reference

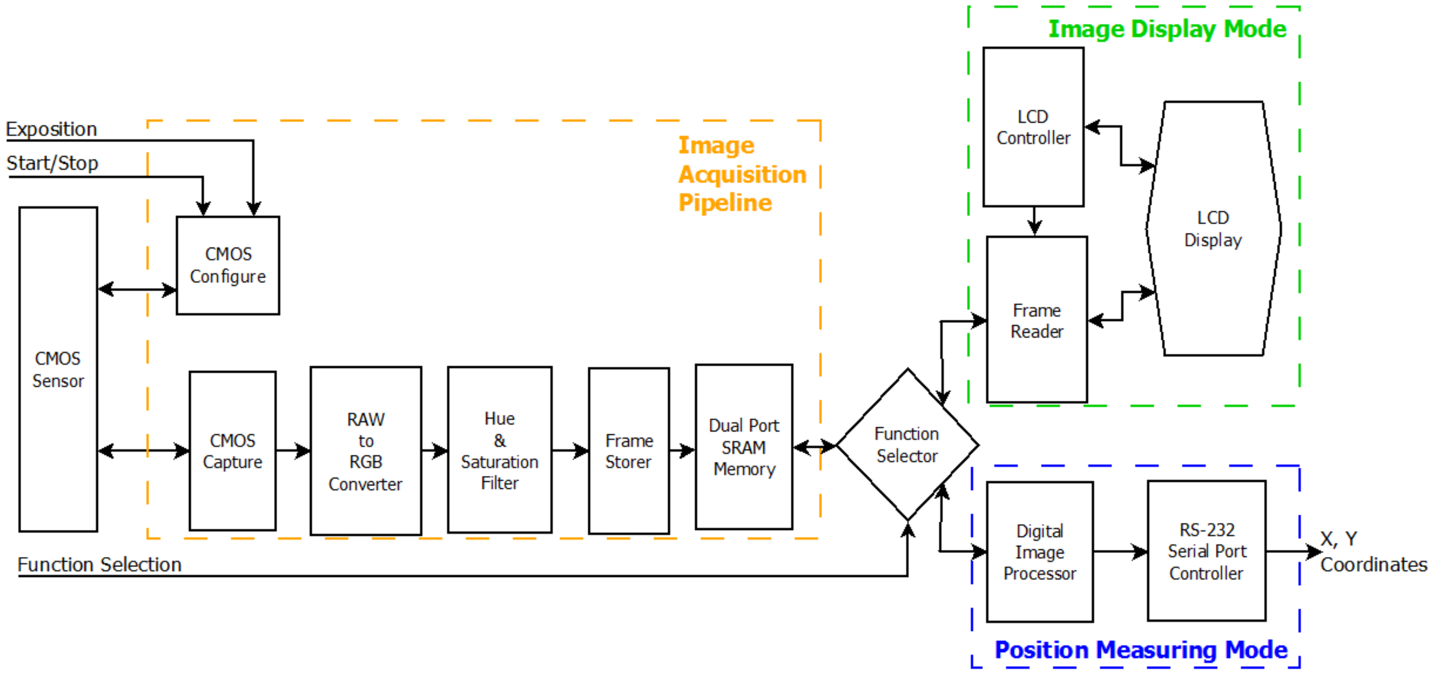


Figure 1.3. Modular diagram of the Computer Vision System designed as second solution.

to adjust either hue or saturation in the circuit, or lightning in the scene. The whole circle detection and position measure is performed in the Digital Image Processor, which also controls the RS-232 Serial port in order to transmit the found coordinates for the balls in the platform.

1.2. **Set Up**

The computer vision system on the tPad board needs no additional programming if the functionality is not going to be changed. Functional peripherals to control the device are described with their respective function in table 1.3. The steps to set up the board to work in normal operation are listed below:

1. Connect the 12 V DC Power Supply.
2. Connect an RS-232 male cable to the corresponding port.
3. Arrange the Switches as shown in table 1.1.
4. Set the switches as described below:
   a. SW[0]: On.
   b. SW[1]: Off.
   c. SW[2]: On.
   d. SW[17]: On.

Table 1.3. Functional peripherals to control the computer vision system on tPad board.

Component	**Function description**
KEY[0]	Master Reset
KEY[1]	Set the new exposure time (use with SW[0])
KEY[2]	Trigger the Image Capture (take a shot)
KEY[3]	Switch to Free Run mode
SW[0]	Off: Extend the exposure time
	On: Shorten the exposure time
SW[1]	Off: Image Display Mode
	On: Position Measuring Mode
SW[2]	Enable Memory Write (On/Off)
SW[17]	Mirror Image (On/Off)
HEX[7:6]	Image Acquisition rate per second (Display only in Hexadecimal format)
HEX[5:4]	Location Packages sent per second (Display only in Hexadecimal format)
HEX[3:0]	Number of acquired frames (Display only in Hexadecimal format)

5. Turn on the device. The Welcome Screen (fig. 1.4) should appear for about half a second while the system starts, and then the scene caught by the camera will be updated in real time in the screen.

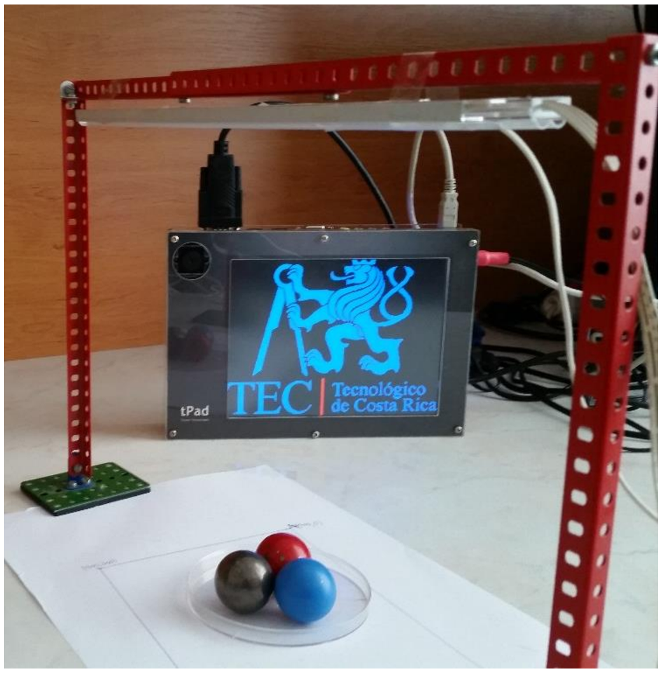


Figure 1.4. Illumination structure for the scene and welcome screen for the Computer Vision System.

6. On the back side of the tPad, the two leftmost 7-segment displays (HEX[7] and HEX[6]) will be showing the camera's rate to acquire the image in real time on hexadecimal format. If the Exposure time needs to be adjusted to achieve at least 50 (0x032) frames per second, press just once KEY[1]. Furthermore, exposure time can be modified using different combinations of KEY [1] and SW[0].
7. Align the frame acquired by the camera with the desired scene to measure the position of the balls. For aligning purposes a straight, colored, identifiable object can be used. The camera is recommended to be around 64 cm above the surface to scan.
8. Turn on the selected illumination system, and check the balls to identify are shown in the screen with a circular shape and with a reasonable number of pixels. An example is provided in figure 1.5.

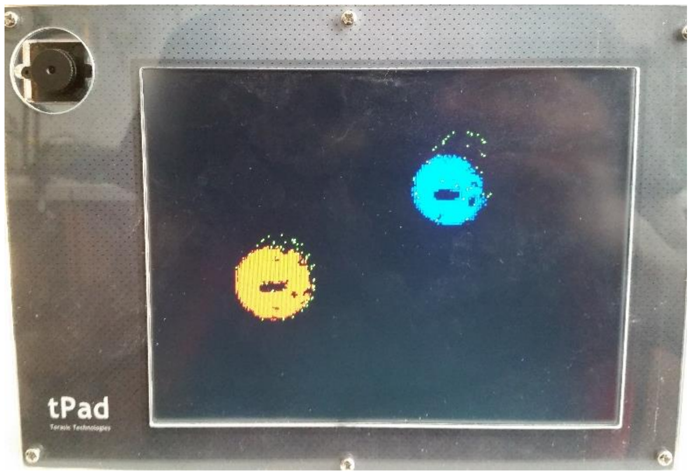


Figure 1.5. Example of reflection on tPad screen of identifiable balls.

9. Your system is now ready to start measuring the ball's position. To do so, just switch SW[1] to On position, and the system will start the stream of locations. If by any reason the dataflow stops, the system can be reset by just pressing KEY[0]. Please note that to return to this point, Camera exposure will need to be adjusted again.
10. The rate at which the data is being sent is now shown at the two 7-segment displays (HEX[5] and HEX[4]) in real time on hexadecimal format.

Chapter 2. Color Filtering

As mentioned in the previous chapter, the color identification occurs in the module named as Hue and Saturation Filter. In this chapter the method it uses to discern between colors will be explained, so it can be modified by the user in order to trace different colors of balls. Saturation levels can be modified as well, but the principle used to threshold this property is far more complicated in terms of implementation than Hue. For this reason Saturation has been set to two levels: S>0.25 and S>0.5.

2.1. **Hue Filter**

This module evaluates each pixel using the HSV threshold in RGB colorspace method, which is theoretically described in [2] and its implementation is outlined in [1]. The module was devised to work with up to seven different levels of hue: the basic six colors (red, yellow,

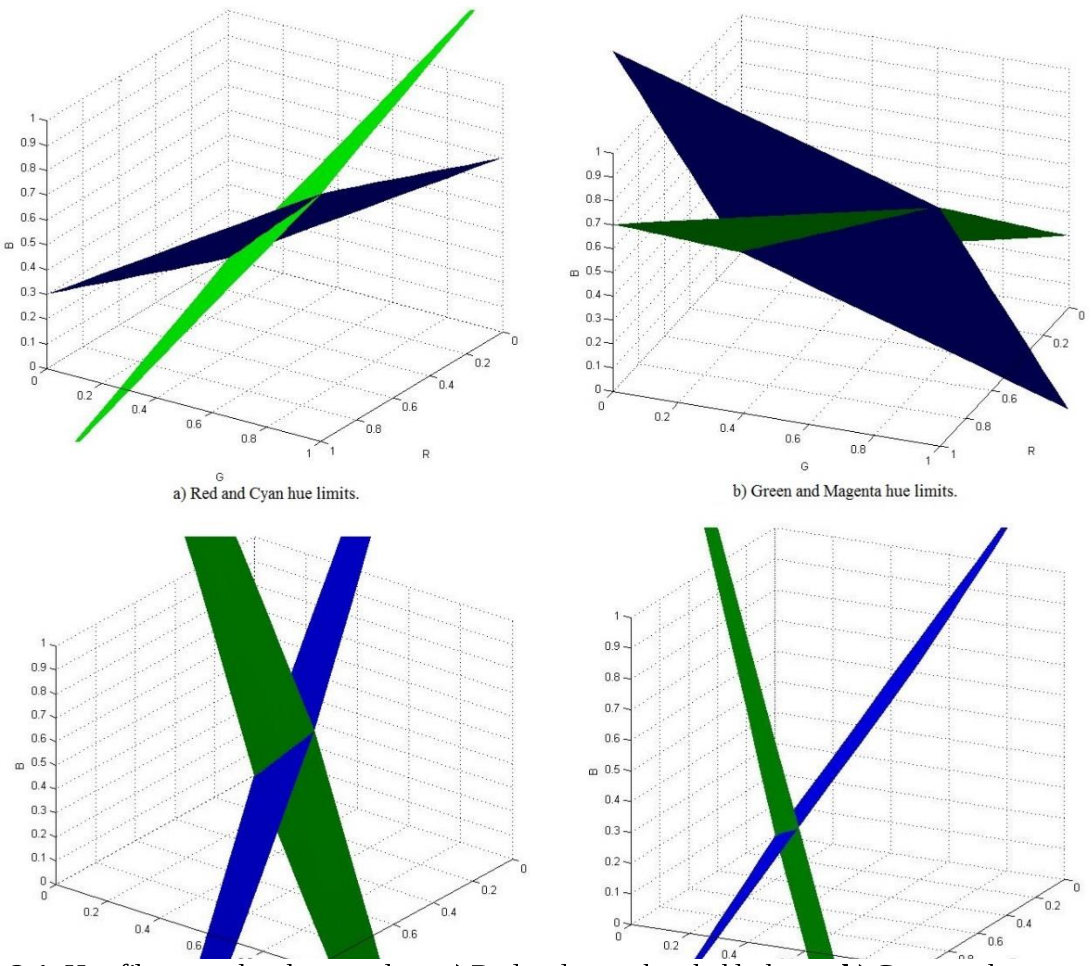


Figure 2.1. Hue filters used to detect colors. **a)** Red and cyan threshold planes. **b)** Green and magenta threshold planes. **c)** Blue and yellow threshold planes. **d)** Orange threshold planes (lower plot).

green, cyan, blue, and magenta) plus orange were selected for standard operation (see fig 2.1).

As Simonian explains the Hue threshold in RGB color space is done using the identity vector, which goes from (0,0,0) to (1,1,1), and a point that represents the color we aim to recognize (note from figure 2.1 that all planes include this line segment). The hue filter implemented is based on this principle with the difference that these elements were used as base to define *regions* in the color space to be recognized as one pigment. For example, in figure 2.1-a) the hue for Red (left bottom) and Cyan (upper right) are shown. These plots are the result of the process of selecting a range of hue, mapping it to RGB color space and find the plane equations that boundary the region of the desired color. Hence, each color has an upper threshold and a lower threshold, being both of them planes in the RGB color space. The plane boundaries for the colors shown in figure 2.1 are shown in table 2.1.

In the module seen in Verilog, the equations are scaled by a factor of 10 to avoid decimal-number handling. The part of the code where the equations are defined has been transcribed in Listing 2.1 to remark some features: First, R, G and B factors are registers inside the module that contain the pixel's values of Red, Green and Blue channels respectively. Second, as the reader can see, the bits to indicate the detection of each color are assigned as a AND function of the equations depicted in table 2.1. These signals are evaluated in order to assign a color code to the bit as shown in table 2.1 using the code depicted in Listing 2.2.

Listing 2.1. Color planes implementation and pixel evaluation.
Module: tPad_Camera/FrameStore/SatFilter

```
assign rHue=(((3*R+7*G)>(10*B))&&((10*G)<(3*R+7*B))),//red
       gHue=(((3*G+7*B)>(10*R))&&((10*B)<(3*G+7*R))),//green
       bHue=(((3*B+7*R)>(10*G))&&((10*R)<(3*B+7*G))),//blue
       yHue=(((3*B+7*R)<(10*G))&&((10*R)>(3*B+7*G))),//yellow
       cHue=(((3*R+7*G)<(10*B))&&((10*G)>(3*R+7*B))),//cyan
       mHue=(((3*G+7*B)<(10*R))&&((10*B)>(3*G+7*R))),//magenta
       oHue=(((3*R+7*B)<(10*G))&&((10*G)<(3*B+7*R)));//orange
```

Table 2.1. Hue conditions and the code to be assigned if the evaluated pixel meets the requirements.

Color	Hue Constraint	Assigned Code [bin]
Red	$3 \cdot R + 7 \cdot G > 10 \cdot B$ & $10 \cdot G < 3 \cdot R + 7 \cdot B$	001
Green	$3 \cdot G + 7 \cdot B > 10 \cdot R$ & $10 \cdot B < 3 \cdot G + 7 \cdot R$	010
Blue	$3 \cdot B + 7 \cdot R > 10 \cdot G$ & $10 \cdot R < 3 \cdot B + 7 \cdot G$	011
Yellow	$3 \cdot B + 7 \cdot R < 10 \cdot G$ & $10 \cdot R > 3 \cdot B + 7 \cdot G$	100
Cyan	$3 \cdot R + 7 \cdot G < 10 \cdot B$ & $10 \cdot G > 3 \cdot R + 7 \cdot B$	101
Magenta	$3 \cdot G + 7 \cdot B < 10 \cdot R$ & $10 \cdot B > 3 \cdot G + 7 \cdot R$	110
Orange	$3 \cdot R + 7 \cdot B < 10 \cdot G$ & $10 \cdot G < 3 \cdot B + 7 \cdot R$	111
Default (none)	-	000

Code from both Listing 2.1 and 2.2 are part of the same module (Hue & Saturation Filter) but they are presented separately to facilitate the explanation process. Note from Listing 2.2 that for the *case* statement are considered both hue and saturation conditions before assigning the color code that denotes the pixel as foreground or background.

Listing 2.2. Hue and Saturation evaluation and color code assignment.

Module: tPad_Camera/FrameStore/SatFilter

```
always@(posedge CLK)
begin
case({rHue&sat50,gHue&sat25,bHue&sat25,yHue&sat25,
     cHue&sat25,mHue&sat50,oHue&sat25})//input bus
     //with Hue&Sat conditions
     7'b0000001:     oSat<=3'h7;// orange  code 7
```

```
          7'b0000010:     oSat<=3'h6;// magenta code 6
          7'b0000100:     oSat<=3'h5;// gray    code 5
          7'b0001000:     oSat<=3'h4;// yellow  code 4
          7'b0010000:     oSat<=3'h3;// blue    code 3
          7'b0100000:     oSat<=3'h2;// green   code 2
          7'b1000000:     oSat<=3'h1;// red     code 1
          default:        oSat<=3'h0;//background code 0
               endcase
     end
```

2.2. **Color Adjustment**

If the user wants to change one of the colors to be detected, he or she must first identify where the color is located in the RGB color space, then set the planes that delimit the desired color to replace them for one of the colors defined in Listing 2.1. Pay special attention to the name of the wire whose equations are being changed in Listing 2.1 because of this wire is summoned in the *case* statement in Listing 2.2; where it is assigned the desired saturation level (0.50 or 0.25). Wire position on the list represents the code it will be assigned, which is reflected in the display and for transmission purposes.

Of course, the past changes only affect the functional part; and now the system will identify the new colors if the equations were established correctly. For display purposes, the user must select an RGB value to represent this new color in the display and replace it in 6-bits Hex format to the color channel outputs *oR*, *oG* and *oB* in the code in Listing 2.3; which belongs to FrameRead module. *Case* section in Listing 2.3 is based on evaluating Qsel register, which happens to contain the color code of the current pixel to be shown in the screen.

Listing 2.3. Color value assignment for display purposes according to code.

Module: tPad_Camera/FrameRead

```
always@(Qsel) //color out decoder
begin
     case(Qsel)
          0:begin//background
               oR   <=   6'h00;
               oG   <=   6'h00;
               oB   <=   6'h00;
```

```
        end
        1:begin//red
             oR   <=   6'h3f;
             oG   <=   6'h00;
             oB   <=   6'h00;
        end
        2:begin//green
             oR   <=   6'h00;
             oG   <=   6'h3f;
             oB   <=   6'h00;
        end
        3:begin//blue
             oR   <=   6'h00;
             oG   <=   6'h00;
             oB   <=   6'h3f;
        end
        4:begin//yellow
             oR   <=   6'h3f;
             oG   <=   6'h3f;
             oB   <=   6'h00;
        end
        5:begin//gray
             oR   <=   6'h1f;
             oG   <=   6'h1f;
             oB   <=   6'h1f;
        end
        6:begin//magenta
             oR   <=   6'h3f;
             oG   <=   6'h00;
             oB   <=   6'h3f;
        end
        7:begin//orange
             oR   <=   6'h3f;
             oG   <=   6'h08;
             oB   <=   6'h00;
        end

        default:begin
             oR   <=   6'h00;
             oG   <=   6'h00;
             oB   <=   6'h00;
        end
    endcase
    end
```

Please note that color in Hex format is usually in eight bits per channel and tPad's screen uses 6-bits wide RGB channels; so, each R, G and B values must be scaled from a maximum value of 255 to 63. For example: Red (color code 1) in its purest shade is represented as *#FF0000*, the values scaled are *#3F0000*, hence in Listing 2.3 case 1 are represented as *oR=3F*, *oG=00* and *oB=00*.

After these modifications, the whole project must be compiled, built and programmed again in the tPad board to check if the color is recognized as desired. If not, the selected region must be verified and (if needed) redefine the plane equations to repeat the process.

Chapter 3. Camera Configuration

The camera is the very beginning of the computer vision system, so it needs to be programmed properly in order to acquire the image according to the specifications. This chapter shows how it is done through Hardware Description Language.

3.1. Frame Properties Modification

The CMOS sensor is programmed from the FPGA to the camera's registers using I2C communication. This port is controlled by an FSM provided by Altera, so to change the configuration, the parameters of the FSM must be modified. For example, Listing 3.1 represents a section of the CMOS Configuration module, where the main parameters values are declared. In this section, the names are usually self-descriptive, except the sensor_row_mode and sensor_column_mode variables. For the user it is recommended not to change these variables before further reading D5M hardware specification [5].

Listing 3.1. Color value assignment for display purposes according to code.

Module: tPad_Camera/I2C_CCD_Config

```
parameter    default_exposure    = 16'h0351;
parameter    exposure_change_value    = 16'd100;

(…)

assign sensor_start_row         = 24'h010036;//
assign sensor_start_column      = 24'h020010;//
assign sensor_row_size          = 24'h030257;//300*2-1
assign sensor_column_size       = 24'h04031f;//400*2-1
assign sensor_row_mode          = 24'h220011;
assign sensor_column_mode       = 24'h230011;
assign Mirror_d     = iMIRROR_SW ?  24'h20C000 : 24'h208000;
```

As the reader can see, in this section some of the parameters for the camera are defined here. The start row and column are just for frame adjusting purposes, while the row and column sizes are set to twice because the CMOS sensor receives the image in Raw format. If the user wants to change image resolution the exposure time must be changed accordingly to achieve the desired frame rate. Initial exposure time can be approximated using equation 3-1, using the value defined as the parameter default_exposure. Equation 3-2 and table 3.1

describe how to calculate the PLL output frequency that drives PIXCLK, which means the clock signal that drives the whole image acquisition pipeline (orange dashed rectangle in fig. 1.3)

$$Exposure\ time = default\_exposure \cdot 22.119 \times 10^{-6} \qquad (3\text{-}1)$$

Table 3.1. PLL output frequency according to bit distribution. **[5]**

Bits	**Name**	**Description**
15:8	Multiplication Factor (MF)	PLL output frequency multiplier. Legal values: [16, 255]
7:6	X	Reserved
5:0	Division Factor (DF)	PLL output frequency divider minus 1. Legal values: [0, 63]

$$PLL\ output\ freq. = input\ freq. \cdot \frac{MF}{DF} \qquad (3\text{-}2)$$

In Listing 3.2 the *case* statement includes all the steps in the Camera configuration routine, executed by the FSM. In each case, the register LUT_DATA takes a different value in which the FSM interprets the eight MSB as the number of register, and the 16 LSB as the value to write given registers are 16 bits wide (some of these values were declared in Listing 3.1). The computer vision system does not modify all the camera registers shown; actually most of them remain on their default value. The complete list of the registers with its values and descriptions can be found in [5], while the initial conditions for the system using the configuration provided in both listings are presented in table 3.2.

Listing 3.2. Color value assignment for display purposes according to code.
Module: tPad_Camera/I2C_CCD_Config

```
/////////////////      Config Data LUT////////////////////
always
begin
     case(LUT_INDEX)
          0:    LUT_DATA   <=    24'h000000;
          1:    LUT_DATA   <=    Mirror_d;//Mirror Row and
                                          //Columns
```

```
2:   LUT_DATA  <=   {8'h09,senosr_exposure};
                                 //Exposure Time
3:   LUT_DATA  <=   24'h050000;//   H_Blanking
4:   LUT_DATA  <=   24'h060019;//   V_Blanking
5:   LUT_DATA  <=   24'h0A8000;//   change latch

6:   LUT_DATA  <=   24'h2B034d;//   Green 1 Gain
7:   LUT_DATA  <=   24'h2c061f;//   Blue Gain
8:   LUT_DATA  <=   24'h2d031f;//   Red Gain
9:   LUT_DATA  <=   24'h2e000d;//   Green 2 Gain
10:  LUT_DATA  <=   24'h100051;//   PLL power on

11:  LUT_DATA  <=   24'h113008;
               //PLL_m_Factor<<8+PLL_n_Divider
12:  LUT_DATA  <=   24'h120001;
               //   PLL_p1_Divider
13:  LUT_DATA  <=   24'h100053;//   set USE PLL
14:  LUT_DATA  <=   24'h624000;//   enable
                                 //calibration

15:  LUT_DATA  <=   24'h60015f;//   green offset
16:LUT_DATA    <=   24'h630010;//   red offset
17:  LUT_DATA  <=   24'h640012;//   blue offset

25:  LUT_DATA  <=   24'hA00048;//Test pattern
                                 //control
26:  LUT_DATA  <=   24'hA103ff;//   Test green
                                 //pattern value
27:  LUT_DATA  <=   24'hA203ff;//   Test red
                                 //pattern value
28:  LUT_DATA <= 24'hA303ff;     // Test blue
                                 //pattern value

18:  LUT_DATA  <=   sensor_start_row;//  set
                                     //start row
19:  LUT_DATA  <=   sensor_start_column;//set
                                 //start column
20:  LUT_DATA  <=   sensor_row_size;//set row
                                     //size
21:  LUT_DATA  <=   sensor_column_size;// set
                                     //column size
22:  LUT_DATA  <=   sensor_row_mode;//   set row
                                 //mode in bin mode
23:  LUT_DATA  <=   sensor_column_mode;// set
                    //column mode in bin mode
```

```
                24:  LUT_DATA  <=   24'h4901E8;//row black target

            default:LUT_DATA<=   24'h000000;
            endcase
        end
```

Table 3.2. Initial Conditions of the Computer Vision System

Parameter	**Value [unit]**
PLL output freq.	144 [MHz]
Exposure time	18,779 [ms]
Row size	599 [pixels]
Column size	799 [pixels]
G_1 gain	4,469
R gain	6,781
B gain	5,328
G_2 gain	1,625

3.2. **Lightning Change**

If the user wants to change the light on the scene, the effect must be considered. The Channel gains for the CMOS sensor are adjusted for the white LED lightning on a Merkur kit shown in figure 1.4. If this needs to be changed for any reason, the steps where LUT_INDEX value is 6, 7, 8, and 9 on the *case* statement in Listing 3.2 must be reviewed.

As mentioned before, the CMOS sensor acquires the image in raw mode. The values selected in these steps are the gain provided to each channel (R, G_1, G_2, and B). Different lightning means different channel gains, but these values are nonlinear; so equation 3-3 and table 3.3 describe how to calculate the actual channel gain represented by the 16 LSB assigned.

$$Effective\ Channel\ Gain = \left(\frac{DG}{8}+1\right)\cdot(AM+1)\cdot\left(\frac{AG}{8}\right) \tag{3-3}$$

For example, G_1 channel in Listing 3.2 has a value of 0x034d, hence DG=3; AM=1 and AG=13. Evaluating these values in eq. 3-2 we obtain the result ECG=4.469.

Table 3.3. Channel gain according to bit distribution. **[5]**

Bits	**Name**	**Description**
15	X	Reserved
14:8	Digital Gain (DG)	The actual digital gain is (1 + value/8). Legal values: [0, 120]
7	X	Reserved
6	Analog Multiplier (AM)	Analog gain multiplier for the channel minus 1. Legal values: [0, 1]
5:0	Analog Gain (AG)	Analog gain setting for the channel times 8. Legal values: [8, 63]

In summary, the actions shown in this document have been simplified to facilitate understanding. However, if the exposed documentation is not enough, please refer to the bibliography.

Bibliography

[1] Barnes, R. (2016) *Optical system for measuring position of metallic colored pellets on a platform*. (unpublished Bachelor thesis) Costa Rica Institute of Technology, Cartago,Costa Rica.

[2] Simonian, A. (2014) *Feedback control for planar parallel magnetic manipulation*. (Diploma thesis) Czech Technical University in Prague, Czech Republic.

[3] Terasic Inc. (2010) *Cyclone IV device handbook, Volume 1.* Consulted at: http://mail.terasic.com.cn/~wyzhou/tPad_v2.0.0_CDROM.rar

[4] Terasic Inc. (2010) *DE2-115 user manual.* Consulted at: http://www.terasic.com.tw/cgi/bin/page/archive.pl?Language=English& CategoryNo=165&No=502&PartNo=4

[5] Terasic Inc. (2010) *Terasic D5M hardware specification* Consulted at: http://mail.terasic.com.cn/~wyzhou/tPad_v2.0.0_CDROM.rar

[6] Terasic Inc. (2010) *Tpad user manual*. Consulted at: http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=183&No=637