
Federal Institute of Technology, Lausanne, EPFL
Electrical and Electronics Engineering Section, SEL
Microelectronic Systems Laboratory, LSM

# Real Time Implementation of the Omnidirectional Vision Construction Algorithm of the Panoptic Camera

- Master Thesis Report -

Student : Vladan Popović

Supervisor : Hossein Afshari

Professor : Yusuf Leblebici

June, 2011
Lausanne

# Table of Contents

Index of Tables....iv
Index of Figures....v
1 Introduction....1
2 The Panoptic camera configuration....3
2.1 Camera Arrangement....3
2.2 Camera Orientation and Parameters....4
2.3 FPGA Development Platform and Camera Prototype....5
2.3.1 Concentrator FPGA Architecture....7
2.3.2 Concentrator FPGA Performance....8
2.3.3 Central FPGA Architecture....9
3 Omnidirectional vision construction....10
3.1 First Algorithmic Step....10
3.2 Second Algorithmic Step....12
4 Hardware implementation....14
4.1 Matlab Model....15
4.2 Concentrator FPGA ....18
4.2.1 Angle Generation....19
4.2.2 Omega Vector Generation....21
4.2.2.1 SCC sub-block....22
4.2.3 Camera Select and Distance Generation....23
4.2.3.1 Square root operator....25
4.2.4 Pixel Position Generation....27
4.2.4.1 Division operator....30
4.2.4.2 Polynomial estimation....32
4.2.5 Interpolation....33
4.2.6 Register Map....35
4.3 Central FPGA ....39
5 Results and Comparison....40
5.1 FPGA Implementation....40
5.2 Omnidirectional Vision Reconstruction....42
6 Real-Time Testing....45

6.1 Panoptic Media Software....45
7 Discussion....47
8 Conclusion....48
Bibliography....49
Appendix A: Full Register Map....50
Appendix B: HD Image Reconstruction....51
Appendix C: Reconstructed Test Images....52

# Index of Tables

Table 2.1: Concentrator FPGA device utilization summary....8
Table 4.1: SNR of 16-bit reconstruction model for image ELA-2....16
Table 4.2: Trigonometric function calculation algorithms comparison....22
Table 4.3: Square root algorithms comparison....26
Table 4.4: Comparison of results of various division algorithms....32
Table 4.5: Comparison of results of two polynomial estimation algorithms....33
Table 4.6: Concentrator FPGA register map....37
Table 4.7: Bit description of Image processing application mode register....38
Table 5.1: Estimated maximum frequencies....40
Table 5.2: Full system utilization summary in the concentrator FPGA....41
Table 5.3: Full system utilization summary in the central FPGA....41
Table 5.4: SNR comparison of the model and actual realization....42

# Index of Figures

Figure 2.1: Hemispherical structure with five floors....3

Figure 2.2: The camera side view....3

Figure 2.3: Vectors identifying the target direction and the coordinates of the cameras' image frame plane....4

Figure 2.4: Architecture of 100 imagers support system....5

Figure 2.5: Setup of the FPGA platform for the 7-floor Panoptic system with 30 embedded cameras....6

Figure 2.6: Concentrator FPGA architecture. ....7

Figure 3.1: Cameras contributing to the pixel position ....11

Figure 3.2: a) Projection of camera centers contributing in direction  onto ω-plane b) Another view of the projection where the intensity values are assigned as the height of the projected contributing camera center points....12

Figure 4.1: Design flow....14

Figure 4.2: Signal-to-Noise Ratio of red channel for various bit precisions....16

Figure 4.3: Panoramic reconstruction using linear interpolation algorithm....17

Figure 4.4: Image processing unit architecture....18

Figure 4.5: Angle generation accumulators....19

Figure 4.6: Architecture of the Omega vector generation block....21

Figure 4.7: Sine and cosine calculator block diagram....23

Figure 4.8: Architecture of the camera select and distance generation block....24

Figure 4.9: Architecture of the pixel position generation block without pipeline registers shown....29

Figure 4.10: Architecture of the address resolving unit....30

Figure 4.11: Architecture of the linear interpolation block....34

Figure 4.12: Image processing application mode register....38

Figure 5.1: Reconstruction simulation results using linear interpolation algorithm....43

Figure 5.2: Final reconstruction simulation result using linear interpolation algorithm. ....44

Figure 6.1: The Panoptic Media application....45

# 1 Introduction

*"... In Greek mythology, Panoptes ("the all-seeing") was an epithet for both Helios and Argus... Argus Panoptes was a giant with a hundred eyes. ... "*

At the modern-day market, almost all of the cameras image the world on a single plane. For a very long time that principle has worked perfectly as there was no interest in making a new camera, but just improving the existing ones.

However, if the images are to be reliable representatives of the real world, they should look like what we see with our eyes. The humans have a very complex stereo-vision system which allows us to have a three-dimensional vision. Because of the stereo-vision we are able to estimate depth and coordinate ourselves in space.

Even though human visual system is very complex and serves us well, there are far more advanced visual systems in the other living beings. One example is a common fly. A fly has almost a complete 360 degrees angle of vision. This type is called omnidirectional vision.

The visual system of the flying insects is the source of inspiration for the presented hardware vision system. The faceted eyes of a common fly provide it with omnidirectional vision, egomotion and depth estimation. The fly's faceted eye is an omnidirectional vision system comprising of several thousands of rudimentary image sensors called ommatidias [1]. Emulating omnidirectional vision is not possible with a single standard camera. One step ahead was made long time ago by introducing the panoramic images, but they require a number of images taken consecutively and it was nowhere near working in real time.

Utilizing the concept of faceted eyes, an omnidirectional camera is realized by layering CMOS image sensors over the surface of a hemispherical structure. This system is referred to as the Panoptic camera and it has two distinguished features. First, it is an omnidirectional camera capable of recording light information from any direction around its center. Second, it is a polydioptric system where each CMOS facet consists of a CMOS camera with a distinct focal plane; hence the full system is a multiple aperture camera.

The PANOPTIC project aims at designing and studying a visual sensor made of 100 planar CMOS cameras. This is a joint project of the Signal Processing Laboratory (LTS2) and Microelectronic Systems Laboratory (LSM) of the Swiss Federal Institute of Technology (EPFL). The project has been partly conducted with the support of the Swiss NSF grant.

This visual sensor can potentially strongly impact the fields as varied as surveillance or autonomous systems navigation. The main ability of the Panoptic camera is real time 3D imaging.

This report is organized in 7 chapters.

The Panoptic camera project will be introduced in Chapter 2. Also, the current state of the project and the hardware development platforms will be explained in the same chapter.

In Chapter 3, omnidirectional vision construction algorithm will be explained. A detailed description of the algorithm is given with all possible options for different steps in the process.

Chapter 4 describes the hardware architecture of the image processing unit which is the main part of this thesis. Each block and sub-block is explained. Block diagrams and pseudo-codes are also provided. Additionally, each choice in the implementation is backed up by comparison with the other options and possibilities. The design includes two design versions which will be mentioned later.

The results are shown in Chapter 5. The resulting images are shown and their comparison to the reference reconstructed image. Also, the resources used are given and the operating frequencies estimations are provided. The real-time implementation results and the developed software is explained in Chapter 6.

Chapter 7 and Chapter 8 summarize the done work during the project and discuss the obtained results.

# 2 The Panoptic camera configuration

## *2.1 Camera Arrangement*

The Panoptic camera is constructed by assembling the MOS imagers on the surface of a structured hemisphere. The hemisphere is structured such that it contains the positions for installation of the CMOS imagers. Each camera upon installation observes a distinct direction.

Each CMOS imager is resembled by a constant circular face with a radius $\alpha_0$. The circular face represents the area occupied by one CMOS imager, its package and embedded connectivity. The objective is to cover the hemisphere surface with the constant circular faces as much as possible. For this purpose a unit radius hemisphere is divided into $N_{\text{flo}} + 1$ latitude floors. Each latitude floor is populated by the constant circular faces. $N_n$ circular faces located on the same floor have the same center latitude angle $\theta_n$. The top most floor located on the North pole of the hemisphere contains one circular face. The latitude angle $\theta_n$ of the $n^{\text{th}}$ floor ( $0 \leqslant n \leqslant N_{flo}$ ) is obtained from:

$$\theta_n = 2\,n\,\alpha_0\ , \quad \alpha_0 = \frac{\pi}{2(2N_{\text{flo}}+1)}\ . \tag{1}$$

The centers of the circular faces are identified by the spherical coordinates $(\theta_n, \phi_{n,j})$ and are evenly positioned according to:

$$\phi_{n,j} = j\,\Delta\phi_n\ , \quad \Delta\phi_n = \frac{2\pi}{N_n}\ . \tag{2}$$

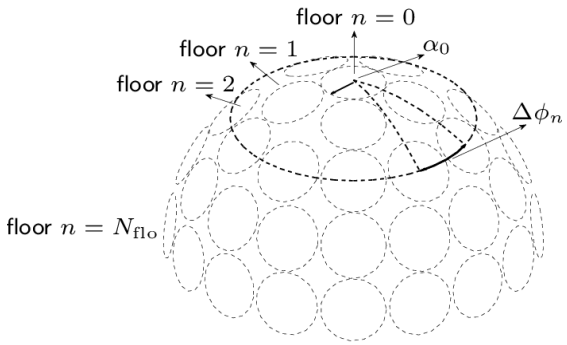



*Figure 2.1: Hemispherical structure with five floors*

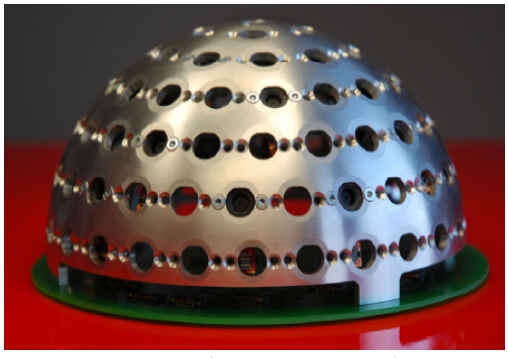


*Figure 2.2: The camera side view.*

As an example, Figure 2.1 depicts a hemispherical structure with five floors ($N_{\text{flo}} = 4$). The 5-floor hemispherical structure contains $N_{\text{cam}} = 49$ cameras. A fabricated Panoptic camera with seven floors ($N_{\text{flo}} = 6$) and $N_{\text{cam}} = 104$ camera positions is shown in Figure 2.2. The diameter of the hemisphere structure shown in Figure 2.2 is $2r_{\odot} = 129$ mm.

## *2.2 Camera Orientation and Parameters*

Each camera is characterized by three unit vectors:

- "target" vector $\vec{t}$ pointing in the camera line of sight (focus direction)
- "up" vector $\vec{u}$ providing the vertical direction in the pixel representation of the camera
- $\vec{v}$ vector orthogonal to the first two, that is the horizontal direction in the pixel domain

The vectors $\vec{u}$ and $\vec{v}$ form an orthogonal reference system for the pixel coordinates of each camera. The $(\vec{t}, \vec{u}, \vec{v})$ system has the center of each camera as its origin. Since there is a need to represent all three vectors in the same domain, orthogonal Descartes $(\vec{x}, \vec{y}, \vec{z})$ system is used. The relations between the two systems for the $i^{\text{th}}$ camera are:

$$\begin{bmatrix} \vec{t}_i \\ \vec{u}_i \\ \vec{v}_i \end{bmatrix} = \begin{bmatrix} \sin(\theta_i)\cos(\phi_i) & \sin(\theta_i)\sin(\phi_i) & \cos(\theta_i) \\ -\cos(\theta_i)\sin(\phi_i) & -\cos(\theta_i)\cos(\phi_i) & \sin(\theta_i) \\ -\sin(\phi_i) & \cos(\phi_i) & 0 \end{bmatrix} \begin{bmatrix} \vec{x} \\ \vec{y} \\ \vec{z} \end{bmatrix}. \tag{3}$$

Figure 2.3 presents the arrangement of these three vectors for each camera of the Panoptic device made of 5 floors. The vectors $\vec{t}, \vec{u}, \vec{v}$ are shown in red, green and blue colors, respectively.

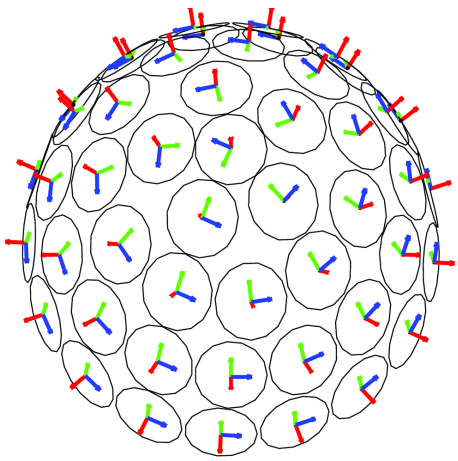


*Figure 2.3: Vectors identifying the target direction and the coordinates of the cameras' image frame plane.*

In addition to the location and orientation of the camera, the main intrinsic camera parameters include the focal length $f_L$, the angle-of-view (AOV) $\alpha$ and resolution of the camera, *i.e.* the size $I_h \times I_\omega$ of the pixel grid.

The focal length controls the mapping between the light ray direction and the pixel coordinates, while the AOV limits the angle around its target direction, beyond which the camera is unable to record light. The camera AOV strictly depends on the direction (horizontal, vertical or diagonal), since the field-of-view is rectangular. However, it can be assumed that the AOV is equal to the minimum one, disregarding the observation direction.

## *2.3 FPGA Development Platform and Camera Prototype*

A multi-layer FPGA system is designed and developed in LSM to support the real time deployment of the Panoptic camera applications. The system consists of 4 layers:

- layer A: 100 imagers with programmable resolution, up to CIF (352 $\times$ 288 pixels)
- layer B: five concentrator FPGA, handling local image processing over 20 imagers in parallel each
- layer C: one central FPGA which processes the final omnidirectional image, based on the data transmitted in parallel from the concentrators
- layer D: a PC is in charge of the applicative layer consisting of displaying the operation results transmitted from the central FPGA.

Architecture of the Panoptic camera system is shown in Figure 2.4.

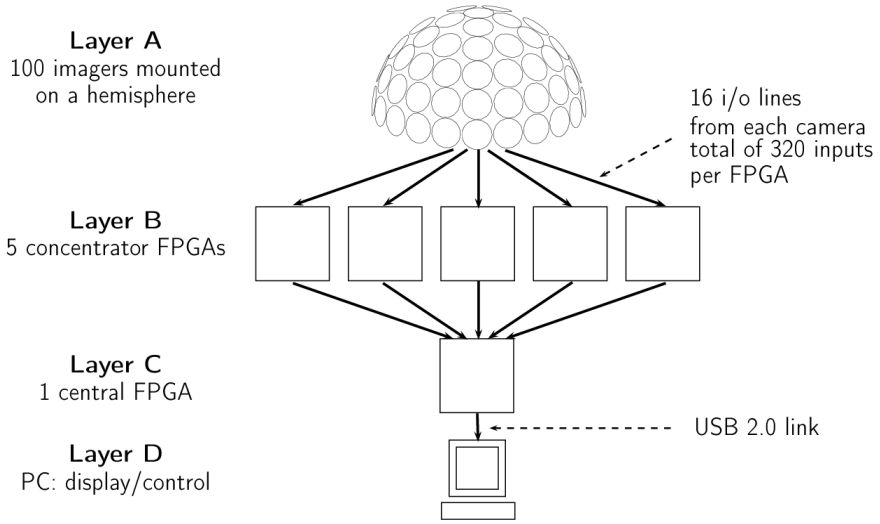



*Figure 2.4: Architecture of 100 imagers support system.*

This system supports up to 100 imagers generating 16-bit CIF images at 25 frames per second rate. In the testing of the system, however, not all 100 imagers were used. The tests were done on a 30 cameras prototype shown in Figure 2.5.

*Figure 2.5: Setup of the FPGA platform for the 7-floor Panoptic system with 30 embedded cameras.*

Both the central and the concentrator FPGAs are Xilinx Virtex5 XC5VLX50-1FF1153C which have 7200 Virtex-5 slices, where each slice consists of 4 registers, 4 LUTs and the additional logic elements [2]. The concentrator FPGA supports up to 20 cameras with 16 input/output lines, each. To support higher number of camera interfaces, multiple boards are stacked. In Figure 2.5, two concentrator boards are connected to the central board. For scalability and extension purposes the designed board also contains high speed LVDS serial links and extension connectors. For external access and high speed data transfer the board is also equipped with a USB 2.0 device chipset. The FPGA board also contains two zero bus turnaround (ZBT) SRAMs with 36 Mb capacity and an operating bandwidth of 166 MHz for each.

The camera modules used in the built Panoptic prototype are PixelPlus PO4010N single chip CIF 368 $\times$ 304 pixels camera (with an effective resolution of 352 $\times$ 288).

A software development platform used in this project is Xilinx ISE 11.5.L.70. Apart from it, the models and the feasibility tests were done in Matlab 7.9.0 (referred as Matlab in the rest of the text) and the behavioral simulations were done in ModelSim SE 6.5c (Modelsim).

### 2.3.1 Concentrator FPGA Architecture

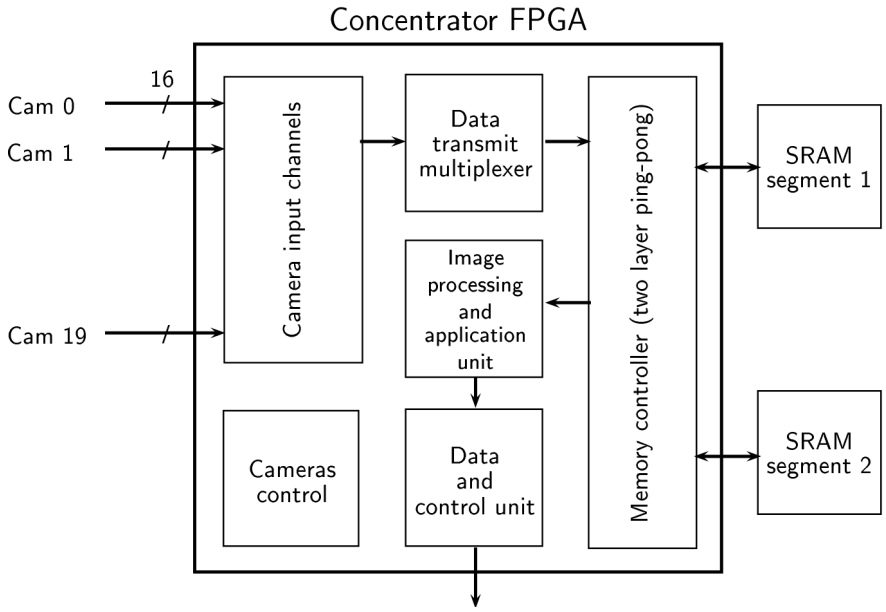



*Figure 2.6: Concentrator FPGA architecture.*

The architecture of the FPGA is shown in Figure 2.6. The concentrator FPGA consists of six major blocks. The arrow lines depicted in Figure 2.6 show the flow of image data inside the concentrator FPGA. Image data streaming from the cameras enters the concentrator FPGA via the camera input channel block. To store the incoming frame data from all the camera modules into one of the single data port SRAMs, a time multiplexing mechanism must be conducted. Hence the data transmit multiplexer block time multiplexes the data received by the camera input channel block and transfers it to the memory controller block for storage in one of the SRAMs. The memory controller block interfaces with two external SRAMs available on the board. The memory controller block provides access for storing/retrieving the incoming/previous twenty image frame data in the SRAMs. The SRAMs swap their role (*i.e.*, one being written to and one being read from) with the arrival of each new image frame from the cameras. The image processing and application unit block is in charge of signal processing and basic functionalities such as single/dual video channel streaming, all channel capture, etc. The image processing and application unit block access the SRAMs via the memory controller block when the camera data is needed and transmits the processed image data to the data link and control unit block. The data link and control unit block provides transmission capability on external interfaces available on the board such as high speed LVDS serial links or USB 2.0 depending on the system version. In the single board version USB link is used to communicate to PC, while in the multiple board version LVDS interface is used to transmit data to the central FPGA. The block also applies the control commands that it receives from computer/central FPGA using the KCPSM3 PicoBlaze microcontroller. The cameras control block is in charge of programming and synchronizing the cameras connected to the FPGA board.

The FPGA resources used by the concentrator FPGA system without image processing block are given in Table 2.1. This data will later be used for comparison with the image processing block resources.

*Table 2.1: Concentrator FPGA device utilization summary.*

| **Resources** | **Used** | **Available** | **Utilization** |
|---|---|---|---|
| Occupied Slices | 2436 | 7200 | 33% |
| Slice LUTs | 4513 | 28800 | 15% |
| Slice Registers | 6323 | 28800 | 21% |
| BlockRAM/FIFO | 13 | 48 | 27% |

### 2.3.2 Concentrator FPGA Performance

Functionality of the concentrator FPGA is divided into two major tasks. The first one is related to the multiplexing of the camera input channels and the second one is related to the speed of the omnidirectional reconstruction inside the image processing block. Each of these constraints will give the minimum acceptable operating frequency for the observed block. In the end, the higher frequency will be taken as the minimum performance limit of the concentrator FPGA.

The cameras output their image data line by line, assuming the synchronization of all 20 cameras connected to the concentrator FPGA. FPGA first captures the incoming line from all 20 cameras. While receiving the next line, the concentrator FPGA multiplexing block stores the previous data into the SRAM. Thus, the concentrator FPGA must store $N_{cam} = 20$ lines of the image data coming from each camera into the SRAMs, in the same amount of time that it takes for a single camera to generate one single image data line. In the mathematical terms, this is expressed as:

$$N_{cam} \times \frac{I_{\omega}}{F_{fpga}} \leqslant \frac{C_{\omega}}{F_{cam}} . \tag{4}$$

In the equation (4), $C_{\omega}$ represents the camera frame width, $I_{\omega}$ the image frame width, $F_{cam}$ the transmit rate of the cameras and $F_{fpga}$ the operating frequency of the system. Using the datasheet information for the cameras $I_{\omega}=352$, $C_{\omega}=464$ and $F_{cam} = 7.5$ MHz, the minimum operating frequency for the multiplexing block is $F_{fpga} = 114$ MHz.

The second performance constraint reflects the amount of time a concentrator FPGA spends to conduct an image processing application. Irrespective of the application, the real-time feature of the system demands the processing time to be less than or equal to the amount of time a single camera spends to generate one full image frame. In particular to the target application of omnidirectional vision reconstruction this performance requirement is obtained from:

$$\frac{N_{pix} \times C_{pix} + C_{latency}}{F_{fpga}} \leqslant \frac{1}{f_{ps}} \tag{5}$$

where $f_{ps}$ is the camera frame per second rate, $N_{pix}$ is the total number of output pixels (*i.e.*, $\vec{\omega}$) being processed in a frame generation time, $C_{pix}$ is the number of clock cycles to process a single pixel and $C_{latency}$ is latency, in terms of number of clock cycles, of the processing stage.

With the targeted frame rate of 25 fps, resolution of 0.25 Mpixel and assumption that 1 clock cycle per camera is dedicated ( $C_{pix} = N_{cam} = 20$ ), latency does not influence the final system operating frequency very much. As the initial assumption it will be chosen that $C_{latency}$=100 , which gives the minimum frequency to be $F_{fpga}$ = 131 MHz.

Since the image processing unit has higher needed frequency, the chosen system clock operating frequency is $\boldsymbol{F_{fpga}}$ = **133 MHz**. This frequency will be the meeting goal of the later synthesis and place and route timing reports.

### 2.3.3 Central FPGA Architecture

The central FPGA has the same architecture as the concentrator FPGA with the exclusion of the camera input channels. A change of the data transmit multiplexer block is also needed. The data transmit multiplexer sends the received data to the SRAM in case of simple operations (single capture, all camera capture, single/dual video) or directly to the image processing block in case of the omnidirectional reconstruction. The main task devoted to the central FPGA consists of receiving the data processed by the concentrator FPGAs, applying the final image processing stage and transferring the final results to a PC, through a USB 2.0 link, for display. The minimum performance requirement for the omnidirectional vision reconstruction application for the central FPGA is also derived from (5).

The image processing unit is much simpler than on the concentrator FPGA and the differences will be explained in Chapter 4.3.

# 3 Omnidirectional vision construction

The Panoptic camera can be used to emulate the omnidirectional vision of a "virtual" observer located anywhere inside the hemisphere by combining the information collected by each camera. The method used to achieve reconstruction is interpolation of light information in the light ray space domain [3]. Constructing an omnidirectional view on a point inside the hemisphere corresponds to estimating the intensity value of all light rays crossing the observer location point.

In this reconstruction process, the omnidirectional view on a discretized sphere surface $S_d$ of directions is estimated. The surface of this sphere is discretized into an equiangular grid with $N_\theta$ latitudes and $N_\phi$ longitudes. As the object of work is a sphere, the logical assumption is that the spherical coordinate system should be used, such that the direction of each pixel is identified by the unit vector $\vec{\omega} \in S_d$ and relates to the spherical coordinates by $\omega = (\theta_\omega, \phi_\omega)$.

The construction of the virtual omnidirectional view $\angle(\vec{q}, \vec{\omega}) \in \mathbb{R}$, where $\vec{q}$ is the observation point, is performed in two algorithmic steps. First, all the cameras having $\vec{\omega}$ in their field-of-view are determined. Since $\vec{\omega}$ does not match with the exact pixel grid locations on the camera image frames, a first level of interpolation is conducted to extract the light intensity in that direction for each contributing camera. The second interpolation step is done on the data estimated in the first step for each $\vec{\omega}$ direction. The data from all the cameras for a chosen direction is taken and interpolated to obtain the final value. The same process is applied to all directions (pixels in the reconstructed image).

Both algorithmic steps will be explained in more detail in the following chapters.

## *3.1 First Algorithmic Step*

The first that should be done is to determine the cameras having $\vec{\omega} \in S_d$ in their field-of-view. The cameras which should be considered must fulfill the condition

$$\omega_{t_i} = \vec{\omega} \cdot \vec{t}_i > \cos(\alpha/2), \tag{6}$$

where $\alpha$ is the camera's AOV and $i$ is the camera index $0 \leqslant i \leqslant N_{cam}$. The angle between $\vec{\omega}$ and $\vec{t}_i$ is supposed to be smaller than $\alpha/2$.

Figure 3.1 illustrates an example of a selection of the contributing cameras for a random $\vec{\omega}$ direction. The edges of 17 contributing camera faces are drawn in full line.

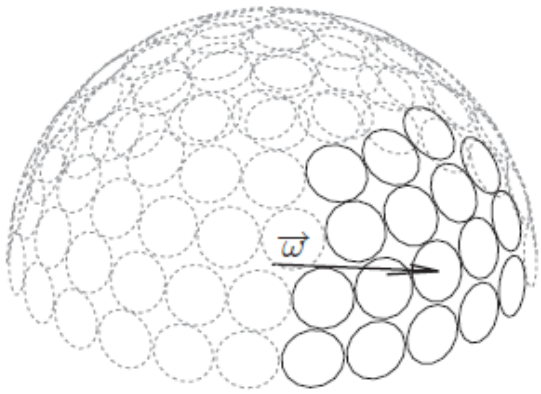



*Figure 3.1: Cameras contributing to the pixel position* $\vec{\omega}$ .

After finding the contributing cameras, the next step is to translate the direction $\vec{\omega}$ into the pixel coordinates of the selected cameras. Using the pinhole camera model [4], the contributing two-dimensional positions $(x_{u_i}, x_{v_i})$ on the $i^{\text{th}}$ camera image plane is expressed as

$$(x_{u_i}, x_{v_i}) = -(\vec{\omega} \cdot \vec{u}_i,\ \vec{\omega} \cdot \vec{v}_i) \frac{f_L}{\omega_{t_i}}, \tag{7}$$

where $f_L$ represents the camera focal length. The equation (7) is an ideal translation equation. However, in reality due to the camera's extrinsic parameters an additional compensation is needed. The non-idealities and the compensation algorithm will be explained in Chapter 4.2.4.

The coordinates $(x_{u_i}, x_{v_i})$ do not need to be integer numbers. This means that the directions $\vec{\omega}$ do not match with any of the pixels in the image frame. Two principles are tested for obtaining the light intensity in the direction $\vec{\omega}$:

- the nearest neighbor technique
- linear interpolation between the neighboring pixels

A detailed explanation of these algorithms can be found in [5]. The tests have shown that by using linear interpolation the results improve very little and that is not worth making the system more complex than it is. Therefore, the nearest neighbor technique is chosen, where for the light intensity in the direction $\vec{\omega}$ the closest pixel is taken instead.

The first algorithmic step is finished with this, providing the pixel value for each camera that contributes in each direction $\vec{\omega}$.

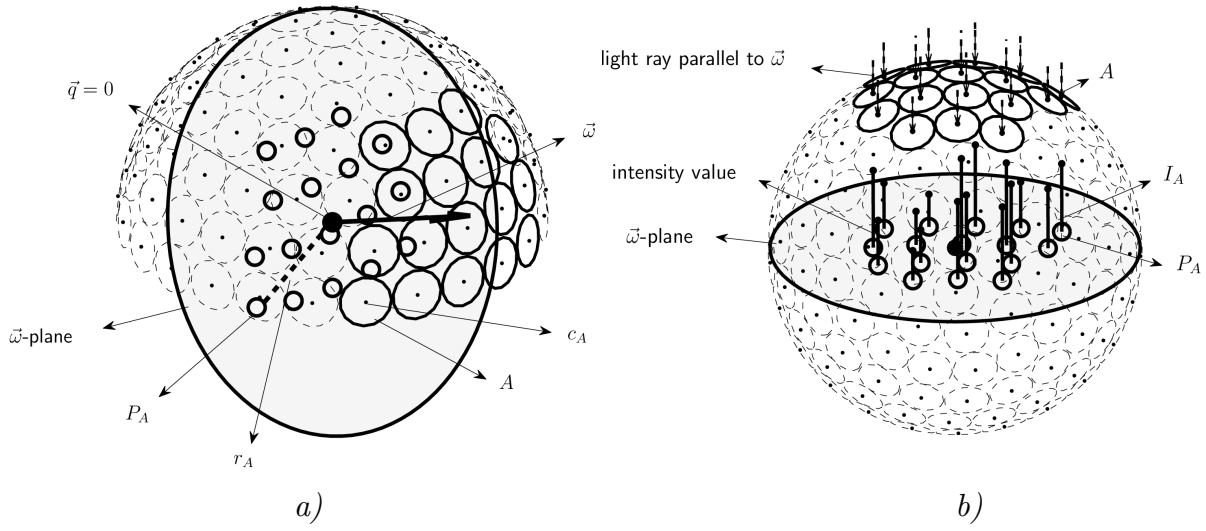
## *3.2 Second Algorithmic Step*

With the obtained light intensities from the first algorithmic step, a second algorithmic step is needed to calculate the final light intensity $\angle(\vec{\omega})$ for the virtual omnidirectional vision.

The second algorithmic step is performed in the space of light rays given by direction $\vec{\omega}$ and passing through the camera origins. Under the assumption of constant light flux (CLF), the light intensity remains constant on the trajectory of any light ray. Following the CLF assumption, an *orthographic* plane is defined such that for a given direction $\vec{\omega}$, light ray intensity only varies in the plane normal to $\vec{\omega}$. The orthographic plane is indicated as the "ω-plane" in Figure 3.2a for the direction $\vec{\omega}$. For clarity Figure 3.2a is redrawn with a change in point of view in Figure 3.2b. The light rays of the direction $\vec{\omega}$ recorded by each contributing camera intersect the ω-plane in points that are the projections of the camera focal points on this plane.

The projected points of the contributing camera positions in the direction $\vec{\omega}$ onto the ω-plane are highlighted by hollow points in Figure 3.2. Each projected camera position on the planar surface is assigned an intensity value calculated in the first algorithmic step.

As an example, the camera indicated as A in Figure 3.2b contributes in the direction $\vec{\omega}$. The projected camera center point of camera A onto ω-plane (*i.e.*, $P_A$) in Figure 3.2b is assigned the intensity value $I_A$. The virtual observer point inside the hemisphere (*i.e.* $\vec{q}$) is also projected onto the ω-plane. The light intensity value of the projected observer point (*i.e.*, $\angle(\vec{q},\vec{\omega})$), is estimated through the algorithmic aggregate of other intensity values, or a subset of them to extract one unique intensity value.




*a)* *b)*

*Figure 3.2: a) Projection of camera centers contributing in direction $\vec{\omega}$ onto ω-plane b) Another view of the projection where the intensity values are assigned as the height of the projected contributing camera center points.*

For the seventeen intensity values shown in Figure 3.2 the intensity value which is observed in the direction $\vec{\omega}$ for observer position $\vec{q}=0$, which is the center of the sphere and indicated by a bold dot in Figure 3.2, can be estimated by the interpolation methods.

The complex interpolation techniques exist for both algorithmic steps. This report only covers implementation of the nearest neighbor and the linear interpolation techniques. The discussion of other techniques is beyond the scope of this work and the reader is referred to [5] for more details.

The details of implementation and linear interpolation steps will be further explained in Chapter 4.2.5.

# 4 Hardware implementation

This chapter elaborates the implementation of the omnidirectional vision reconstruction using the Panoptic camera at the architectural level. Both the nearest neighbor and linear interpolation techniques are used and explained.

Before going into the details of the design, it is important to explain design flow and steps in creating such a large system. Figure 4.1 represents a typical sequence of the design methodology used in this project.

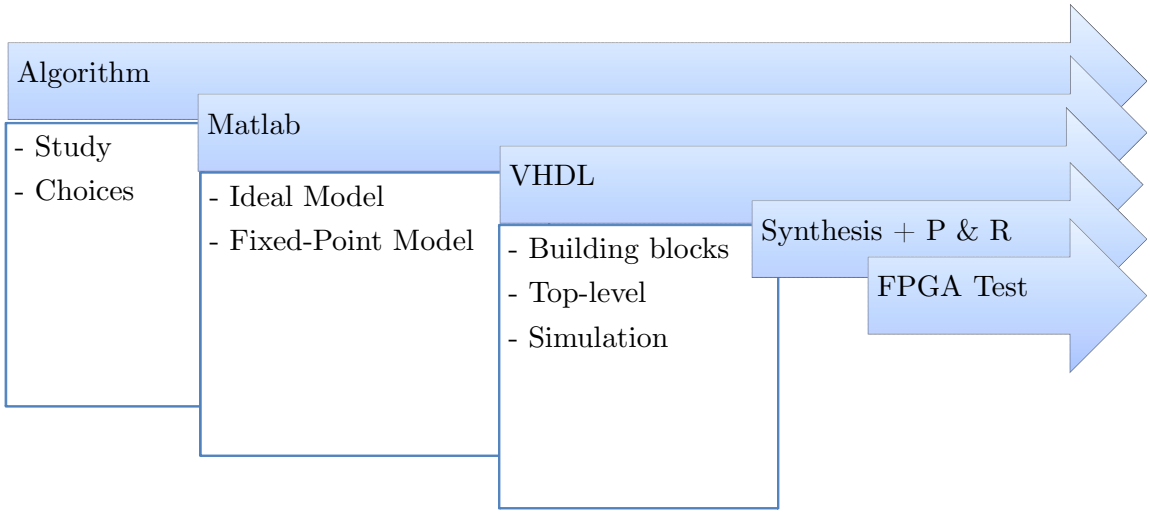



*Figure 4.1: Design flow.*

After the initial study of the algorithm and the ideal Matlab model provided by Hossein Afshari from LSM and Luigi Bagnato and Pierre Vandergheynst from LTS2, the next step was to make a fixed-point Matlab model of the system. A fixed-point Matlab model is necessary for more than one reason. First, it can confirm that the 16-bit precision for all values is enough for credible reconstruction. Some small, reasoning errors might be noticed during creation or testing of the model and they can be easily corrected as it is still just a beginning phase. Another important reason is verification of VHDL model which will be made afterward. The obtained results in Matlab should correspond to the simulation results of the system in ModelSim.

Apart from just fix-pointing the values, it is important to replace certain mathematical operations with the hardware implementable algorithms. Operations such as division, square root or trigonometric functions are not a part of standard VHDL language and algorithms for its implementation have to be chosen. Fixed-point model includes these algorithms, since it is supposed to correspond to the final system in great detail.

Writing the VHDL modules is the next step in the process. The bottom-up

approach was used in the design process, as the smaller blocks were created at first. In the end, the full image processing unit is formed and embedded in the full system which was already created. When each module is finished, it was tested using ModelSim and the results are compared to the one obtained in Matlab.

Following the creation of the system, synthesis and place and route (P&R) follow. Here it is important that the operating frequency is always greater than 133 MHz which was calculated in Chapter 2.3.2. When the full system was successfully simulated and routed, FPGA implementation and testing were done as the last step.

It is important to note that all the design process last until the very end. It is an iterative process, where an error noticed in any part of the design has to be checked and corrected in the previous steps. That way, both Matlab models and VHDL design keep updating until the successful real-time test.

## *4.1 Matlab Model*

Making the fixed-point model is crucial for this project. By proving that the system can work with reduced and fixed precision it can be shown that the final images will be of sufficient quality.

For this project, a fixed-point model with generic number of $N$ bits is made. This means that all the values in the system are represented as fix-point numbers with $N$ bits. However, depending on the range of the values (most of them are normalized to [0..1], but some are not) the position of the point changes. Throughout the model it has been taken care of the current position of the point in order to have valid results in the end.

The fixed-point model is tested on the previously acquired images of EPFL auditory ELA-2 and compared to the ideal model results. The color channels in the images are already represented in a RGB565 representation, where red and blue channels are represented with 5 bits and green with 6 bits. A comparison is made for various bit precisions and the results are shown in Figure 4.2 for range [8..22] bits. The tested algorithm was the nearest neighbor.

It can be observed that for precisions higher than 16 bits, the SNR does not grow too much and it saturates quickly to the maximum. Also, it should be noted that enlarging the number of bits, more logical elements are used when implemented in hardware. Also, number of multipliers, which are critical in this design grow very fast with the increase of bits. This is due to the iterative division algorithm (Chapter 4.2.4) where with each new iteration two new multipliers are needed and the increase of bit precision must be followed by increase of the iterations. Also, hardware multipliers inside the FPGAs are 18-bit and going above that limit requires two hardware multipliers per operation.

When all the facts are taken into consideration, the choice was to use 16-bit representation due to savings in the resources and no significant gain in the image quality.

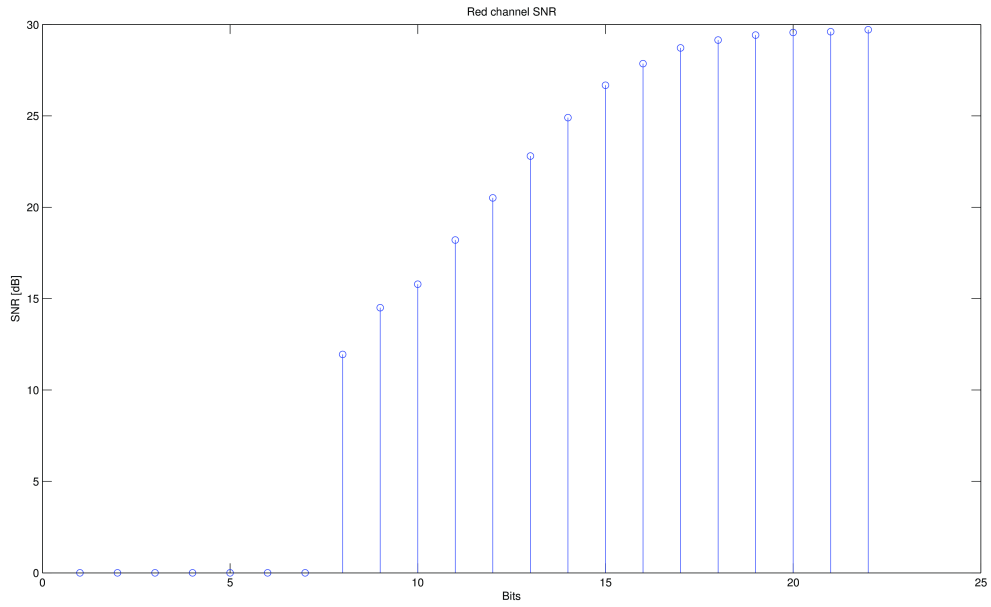



*Figure 4.2: Signal-to-Noise Ratio of red channel for various bit precisions.*

When the choice is made, the number of iterations for division and square root algorithm is calculated, the test has been made on the 16-bit optimized design for the linear interpolation method. The resulting data is shown in Table 4.1 and the obtained images and their differences[1] are shown in Figure 4.3.

*Table 4.1: SNR of 16-bit reconstruction model for image ELA-2*

| | **Red channel** | **Green channel** | **Blue channel** |
|---|---|---|---|
| **SNR [dB]** | 25.84 | 30.17 | 26.06 |

It can be observed that most of the differences originate from the subtle change of color. This change is due to mathematical operations in linear interpolation algorithm where calculation precision is lost and the changes in color can be observed. These differences are shown as grayish pixels in Figure 4.3c.

Other differences are due to the wrongly picked pixels due to loss in precision in operations leading to the calculation of pixel position in the image frame. However, these errors occur less frequently and can be recognized as white dots in Figure 4.3c.

The final conclusion that can be drawn from the Matlab models is that it is possible to reconstruct an omnidirectional image using a 16-bit system with a quality enough not to be noticeable by the human eye.

1 The differences are shown with original intensities, with black representing a good match. Displaying them in inverted manner was not showing anything in the printed version of the report.

*a) Ideal (floating-point) reconstruction*

*b) 16-bit fixed-point reconstruction*

*c) Difference between two reconstruction models*

*Figure 4.3: Panoramic reconstruction using linear interpolation algorithm.*

## *4.2 Concentrator FPGA*

The implementation details of the omnidirectional vision reconstruction unit of the image processing and application block of the concentrator FPGA is presented in this section. Figure 4.4 illustrates the image processing unit architecture and its five sub-blocks. The flow of data and control strobes is shown with arrows in Figure 4.4.

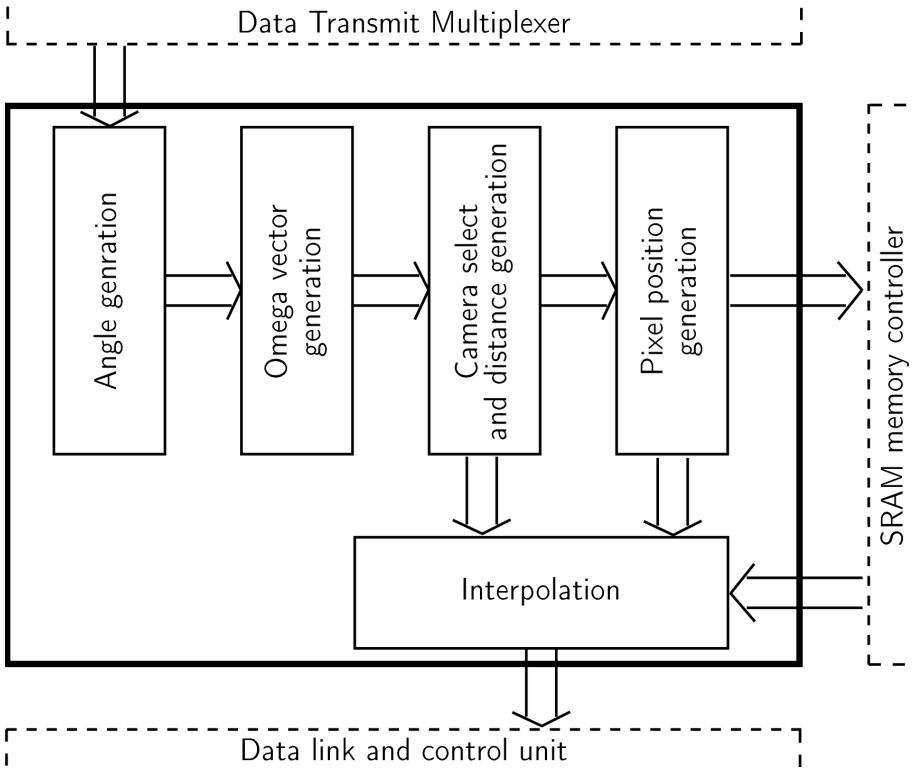



*Figure 4.4: Image processing unit architecture.*

The Angle generation module creates the spherical coordinates of pixel direction $\vec{\omega}=(\theta_\omega, \phi_\omega)$ . The omega vector module generates the $\vec{\omega}$ unit vector. The camera select and distance generation module identifies which cameras are contributing in pixel direction $\vec{\omega}$. This sub-block also provides the distance factors for interpolation sub-block used in the second step of the reconstruction algorithm. The pixel position module generates pixel coordinates inside the image frame and provides an address in the SRAM where that pixel is located. The retrieved value is directed to the interpolation sub-block. The interpolation sub-block conducts the second algorithmic step and provides the estimated intensity value to the data link and control unit block. For the multiple board system, apart from the estimated pixel intensity it provides the distance factor for the final interpolation on the central FPGA.

### 4.2.1 Angle Generation

Purpose of the angle generation module is to create pairs of angles $(\theta_\omega, \phi_\omega)$ which represent latitude and longitude angles in the spherical coordinate system. Actual values of $\phi_\omega$ and $\theta_\omega$ are:

$$\begin{aligned} \phi_\omega(i) &= \frac{2\pi}{N_\phi} \times i \\ \theta_\omega(j) &= \frac{\pi}{N_\theta} \times (j + \frac{1}{2}) \end{aligned} \tag{8}$$

where $N_\phi$ and $N_\theta$ are number of latitudinal and longitudinal pixels, respectively.

The presented pixelization scheme (angle generation) does not provide homogeneous density of pixel positions over the surface of the sphere. Density of the pixels is higher over areas closer to the top. There are other pixelization schemes which provide homogeneity, but the presented one is chosen because of its regularity and ease of implementation.

To go through all the pixels on the spherical surface, an order must be established. First all pairs for one constant $\theta_\omega$ are generated. After all of them are generated, $\theta_\omega$ is increased and the same process is repeated until all pairs are provided.

The angles are represented in a $\pi$ based representation using $N = 13$ bits. The representation is given by the expression:

$$a_0\, a_1 \cdots a_{N-1} = 2\pi \sum_{i=0}^{N-1} a_i \cdot 2^{-(i+1)}$$

The block is implemented to work with generic number of bits $N$, but in the synthesis it is set to 13. This will set the maximum $N_\phi = N_\theta = 2048$ and the maximum possible image resolution of 4 Mpixels, which allows a possible reconstruction of a minimum HD resolution ($1024 \times 768$) display.

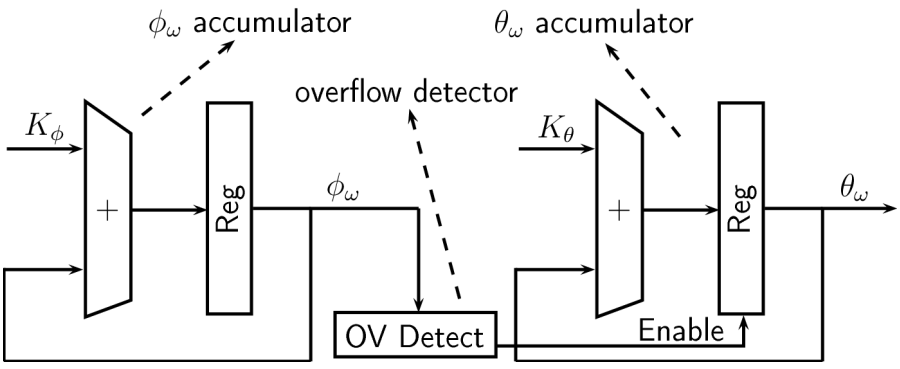



*Figure 4.5: Angle generation accumulators.*

Generation of the angles $\phi_\omega$ and $\theta_\omega$ is implementable using an accumulator for each angle. To generate all possible combinations of $\phi_\omega$ and $\theta_\omega$, one accumulator is chosen to increment when the other accumulator completes its full range cycle. This concept is shown in Figure 4.5. The $\theta_\omega$ angle is kept constant until the $\phi_\omega$ angle accumulator does not generate an overflow signal, which triggers $\theta_\omega$ to increment.

The two accumulators are reset when the new frame signal is generated by the multiplexing stage. Afterward, the same process starts again, for the new frame reconstruction.

The $\phi_\omega$ and $\theta_\omega$ accumulators have separate incrementing indexes $K_\phi$ and $K_\theta$ respectively. The incrementing indexes $K_\theta$ and $K_\phi$ govern the resolution of the constructed omnidirectional vision. In addition, delimiting intervals $(\theta_{min}, \theta_{max})$ and $(\phi_{min}, \phi_{max})$ is defined for both angles. By changing these boundaries, display part of the image can be changed. This is useful for operations such as zooming in, when shortening these intervals and lowering steps (*i.e.*, increasing resolution), zooms one part of the panoramic image without any loss in resolution.

The span of the reconstructing vision is settable through configuration of the latter interval parameters. All the parameters are set by the user through the registers used by PicoBlaze microcontroller.

The pseudo code of the Angle generation module is presented in Algorithm 1. To support a minimum HD resolution (1024 × 768) display, two eleven bit accumulators have been selected for implementation.

**Algorithm 1:** Generate Angle

```
loop
  while θω<θmax do
    while φω<φmax do
        φω := φω+Kφ
    end while
     φω := φmin
     θω := θω+Kθ
  end while
   θω := θmin
end loop
```

### 4.2.2 Omega Vector Generation

The omega vector generation module receives angles $\phi_\omega$ and $\theta_\omega$ of pixel directions and generates the unit vector $\vec{\omega}$ according to following equation:

$$\vec{\omega}=\sin(\theta_\omega)\cos(\phi_\omega)\vec{x}+\sin(\theta_\omega)\sin(\phi_\omega)\vec{y}+\cos(\theta_\omega)\vec{z} \quad (9)$$

In order to reduce the implementation cost and save on hardware multipliers inside the FPGA, the simple addition-based identities for product of trigonometric functions, the new equation is obtained:

$$\vec{\omega}=0.5(\sin(\theta_\omega+\phi_\omega)+\sin(\theta_\omega-\phi_\omega))\vec{x}+0.5(\cos(\theta_\omega-\phi_\omega)-\cos(\theta_\omega+\phi_\omega))\vec{y}+\cos(\theta_\omega)\vec{z} \quad (10)$$

By using equation (10) no multipliers are needed in the design. On the other hand, now sine and cosine values are needed for three different angles: $\theta$, $\theta-\phi$, $\theta+\phi$. The implementations of the trigonometric functions of sine and cosine have been the focus of direct digital frequency synthesizers (*i.e.*, DDFS) for the past decades. Hence many algorithms have been developed for the purpose of the implementation of the basic trigonometric functions. Look-up table (LUT) based algorithms [6-7], the CORDIC algorithm rotation mode [8-9] and polynomial approximation based algorithms [10] are the major categories in this field. Selection of the sine and cosine calculator (*i.e.*, SCC) algorithm is arbitrary and dependent on system requirements such as performance and resource usage.

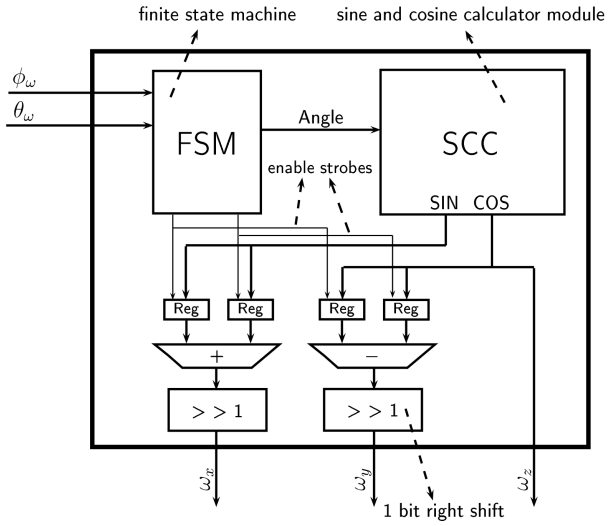



*Figure 4.6: Architecture of the Omega vector generation block.*

**Algorithm 2:** Generate Omega Vector

**for all** $\vec{\omega}$ **do**

$$\omega_x := \frac{\sin(\theta_\omega-\phi_\omega)+\sin(\theta_\omega+\phi_\omega)}{2}$$

$$\omega_x := \frac{\sin(\theta_\omega-\phi_\omega)+\sin(\theta_\omega+\phi_\omega)}{2}$$

$$\omega_z := \cos(\theta_\omega)$$

**end for**

Internal architecture of the Omega vector generation block is shown in Figure 4.6 and next to it is the pseudo code of the algorithm. The $x$, $y$ and $z$ components of the $\vec{\omega}$ are calculated utilizing a finite state machine (*i.e.*, FSM) and a single SCC module. The SCC module calculates the sine and cosine at the same time. Hence by inputting the following angle combinations to the SCC module: $\theta$, $\theta-\phi$, $\theta+\phi$, and combining their respective sine and cosine outputs in the correct order, the $\vec{\omega}$ is obtained.

#### *4.2.2.1 SCC sub-block*

The core of the block is the SCC, which has angle value as its input and it outputs sine and cosine values of the input angle at the same time. Two of many possible algorithms for realization of SCC are considered:

- LUT based algorithm
- CORDIC algorithm

Both of these algorithms were tested in Matlab. The final results of comparison of the two algorithms are given in Table 4.2. Conclusion is that the LUT algorithm is more precise, apart from being faster. Bit precision of the LUT algorithm is close to 16, which was expected as 16 bit values were stored in the LUT. The only drawbacks are hardware requirements which are higher than in the case of the CORDIC algorithm implementation. However, in the FPGA we have the ability to use internal BlockRAMs for storing large amounts of data and in the case of LUT algorithm implementation, 2 BlockRAMs will be used.

The bit precision in Table 4.2 was calculated using the formula:

$$N = (SNR - 1.76) / 6.02 \tag{11}$$

*Table 4.2: Trigonometric function calculation algorithms comparison.*

| | **LUT** | **CORDIC** |
|---|---|---|
| **MSE** | 6.4991e-7 | 3.281e-6 |
| **SNR [dB]** | 97.9951 | 90.9635 |
| **N [bit]** | 15.9859 | 14.8179 |

The block diagram of the SCC block is given in Figure 4.7. The SCC block has two LUTs (one for sine and one for cosine) stored in BlockRAMs. Inside the LUTs only first octant of the functions is stored. Storing the first octant is enough for the reconstruction of the full sine or cosine signal. For the detailed explanation and results the reader is referred to the semester project report written by the same author.

The rest of the components are logical gates used for reconstruction of the full signal. Also, a few pipeline stages are added to the design, but they are not shown in Figure 4.7.

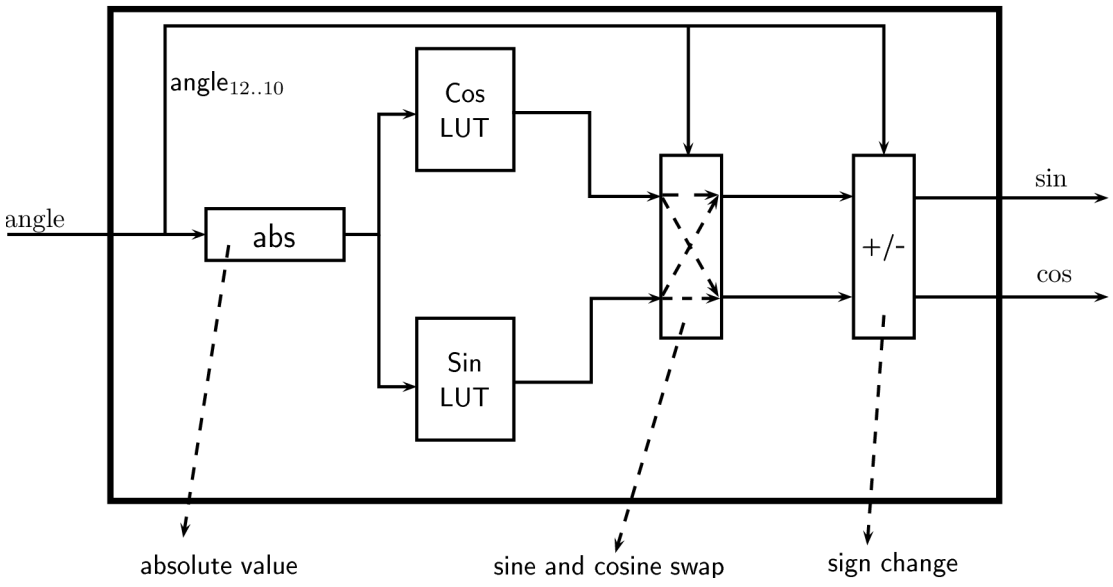



*Figure 4.7: Sine and cosine calculator block diagram.*

### 4.2.3 Camera Select and Distance Generation

The camera select and distance generation sub-block identifies which cameras observe the pixel direction $\vec{\omega}$ and calculates the distance between the projected focal point and the projected virtual observer point on the ω-plane for each contributing camera. For example this distance is identified with $r_A$ and depicted by a dashed line for the contributing camera A in Figure 3.2a.

The inverse of this distance is used as the coefficient (*i.e.*, weight) of the intensity value of a contributing camera in the linear interpolation scheme of the second algorithmic step. The contributing camera which yields the minimum distance for pixel direction $\vec{\omega}$ is used as the sole contributor in a nearest neighbor scheme of the second algorithmic step.

Utilizing the isotropic model (*i.e.*, a conic field of view centered at the camera focal point with an opening angle of $\alpha$) for all the cameras, a camera observes a pixel direction $\vec{\omega}$ if it satisfies the following inequality:

$$\vec{\omega} \cdot \vec{t} \geqslant \cos\left(\frac{\alpha}{2}\right) . \tag{12}$$

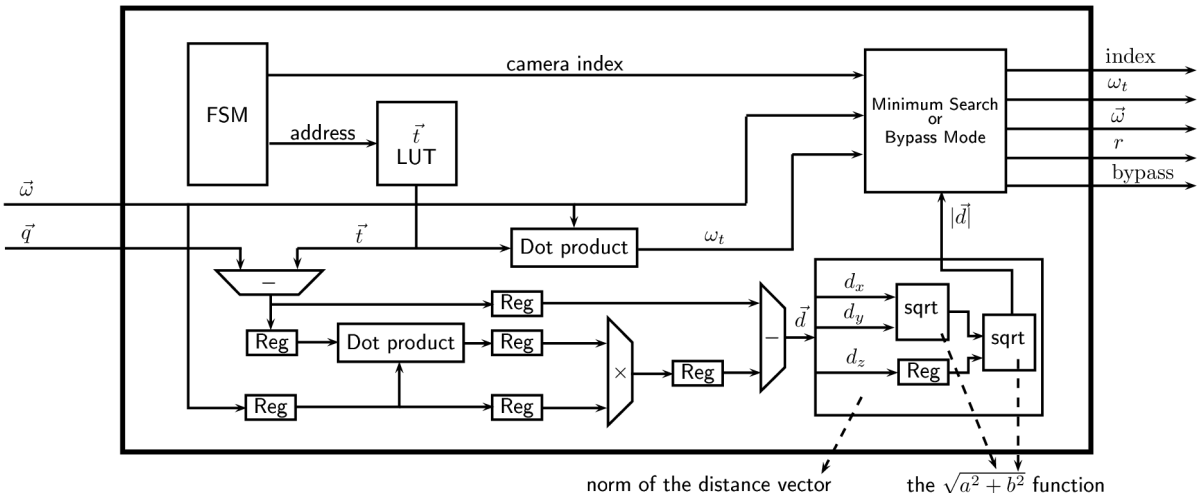



*Figure 4.8: Architecture of the camera select and distance generation block.*

The architectural overview of this block is shown in Figure 4.8. For each new pixel direction $\vec{\omega}$ that the camera select and distance generation block receives, it sequentially provides the stored $\vec{t}$ values of the interfaced cameras of the board, to a dot product and inequality checker unit described in (12). The dot product unit of this block is implemented using a standard implementation with three multipliers and two adders. For gain in speed, all intermediate results are pipelined before the next operation.

The indexes of the contributing cameras are transferred to the minimum search/bypass mode unit of this block. At the same time the distance vectors of the projected focal points of the cameras are calculated through the following formulation:

$$\vec{d} = \left| (\vec{q} - \vec{t}) - ((\vec{q} - \vec{t}) \cdot \vec{\omega}) \times \vec{\omega} \right| . \tag{13}$$

The magnitude of the $\vec{d}$ vector is passed as the distance value $r$ to the minimum search/bypass mode unit. If a linear interpolation scheme is chosen for the second algorithmic step, the contributing cameras $r$ values and their corresponding indexes are passed to the interpolation sub-block and pixel position generation block. If a nearest neighbor scheme is chosen, a minimum search is applied on the $r$ values of the contributing cameras, and the index of the contributing camera with the lowest $r$ value is returned to the pixel position generation block. The $\vec{\omega}$ and $\omega_t$ are also

transferred to the pixel position generation block.

The pseudo code of the operation of this block is shown in Algorithm 3.

**Algorithm 3**: Camera Select and Distance Generation

$d_{min} := 1$
**for all** cameras **do**
  $\omega_t := \vec{\omega} \cdot \vec{t}$
  $\vec{d} := (\vec{q} - \vec{t}) - ((\vec{q} - \vec{t}) \cdot \vec{\omega}) \times \vec{\omega}$
  **if** $\omega_t > \cos(\alpha/2)$ **then**
    **if** interpolation == nearest_neighbor **then**
      **if** $|\vec{d}| < d_{min}$ **then**
        $d_{min} := |\vec{d}|$
        **STORE** camera_index
        $\max_{\omega_t} := \omega_t$
      **end if**
    **else if** interpolation == linear **then**
      $r := |\vec{d}|$
      index := camera_index
    **end if**
  **end if**
**end for**
**if** interpolation == nearest_neighbor **then**
  $r := d_{min}$
  $\omega_t := \max_{\omega_t}$
  index := camera_index
**end if**

#### *4.2.3.1 Square root operator*

For $|\vec{d}|$ calculation a square root operator is needed. There are a lot of algorithms for square root calculation with their advantages and disadvantages over the rest. Three most used iterative algorithms were picked and compared:

- Bit-by-bit
- Babylonian method (derived from Newton-Raphson numerical method)
- CORDIC vectoring mode

Bit-by-bit algorithm is the simplest one in terms of complexity. If the goal is to calculate $\sqrt{x}$, the algorithms starts checking from $y = x/4$ and compares $y^2$ and $\sqrt{x}$.

If $y^2$ is lower, then the next check is for $(y+y/2)^2$, but if not then for $(y/2)^2$. Thus, after a number of iterations, the algorithm will converge to a solution with the needed bit precision. As it was mentioned before, the advantage of this algorithm is its simplicity. However, implementation of this algorithm requires $N_{bit}-1$ iterations and a multiplier in each of them if the system is to be pipelined.

The Babylonian method [11] uses the following iterative equation:

$$y_{n+1}=\frac{1}{2}(y_n+\frac{x}{y_n}) . \tag{14}$$

The advantage of this method is very fast convergence, which results in small number of stages. On the other hand, the great disadvantage is that in each stage a divider is needed. According to the chosen dividing algorithm (explained in Chapter 4.2.4.1), one divider needs five multipliers for the correct operation.

The third algorithm, CORDIC is used in its vectoring mode [12]. The algorithm uses the following iterative equations:

$$\begin{bmatrix} X_{k+1} \\ Y_{k+1} \end{bmatrix} = \begin{bmatrix} 1 & -d_k 2^{-k} \\ d_k 2^{-k} & 1 \end{bmatrix} \begin{bmatrix} X_k \\ Y_k \end{bmatrix}$$
$$d_k=\begin{cases} 1, & Y_k<0 \\ -1, & \text{otherwise} \end{cases} \tag{15}$$

The advantage of the algorithm is the absence of multipliers in iterations since multiplication by the powers of 2 can be implemented using shifters. The only needed multiplication is for the coefficient scaling. Furthermore, in each iteration the accuracy of the results improves by 2 bits. Also, the algorithm does not compute the square root of one number but $\sqrt{a^2+b^2}$. At the same time it is also a disadvantage due to the fact that for modulus calculation of a 3D vector, this algorithm should be applied twice.

A comparison of three above-mentioned algorithms is done for the case where $\sqrt{a^2+b^2+c^2}$ should be calculated. The results are shown in Table 4.3.

*Table 4.3: Square root algorithms comparison.*

| **Algorithm** | **Number of stages** | **Number of multipliers** |
|---|---|---|
| Bit-by-bit | 17 | 17 |
| Babylonian | 5 | 18 |
| CORDIC | 16 | 2 |

The number of stages is not critical because it influences only latency (in case of pipelined system), the obvious choice is to use CORDIC algorithm.

Implementation of the final square root operator are two consecutive, two input norm calculator units (*i.e.*, $\sqrt{a^2+b^2}$) used to realize a three input norm functionality (*i.e.*, $\sqrt{a^2+b^2+c^2}$). The two input norm calculator has been implemented using an eight stage pipelined version of CORDIC algorithm in vectoring mode.

### 4.2.4 Pixel Position Generation

The Pixel position generation block extracts the pixel coordinates in the image frame of the camera corresponding to the pixel direction $\vec{\omega}$. The pixel position is, in a simplified way, described using the equation (7). However, in reality, the mapping of a 3-D scene on to an observed 2-D plane of a camera image frame is not as simple as described in (7). The camera's intrinsic parameters are necessary to explain the vision of a camera. These parameters characterize the mapping between a 3-D scene and the observed 2-D plane and they are split in two classes.

The first class is the linear *homography*, defined by a 3 × 4 camera matrix mapping of 3-D points (homogeneous) coordinates into 2-D (homogeneous) pixel coordinates [4]. The second class models the non-linear mapping effects such as lens distortion. The reader is referred to [4] for more details of the theoretical description and estimation of these parameters. These parameters are estimated through a calibration process and stored in a LUT for each camera that is connected to the observed FPGA concentrator board. The parameters are applied on input $(X_u, X_v)$ coordinates to attain a new set of $(X'_u, X'_v)$ positions. The algorithmic description of this procedure is shown in Algorithm 4 and the compensation equations are given as:

$$\begin{aligned} X'_u &= \left[(1+k_1R^2+k_2R^4+k_5R^6)\cdot X_u + 2k_4X_uX_v + k_3(R^2+2X_u^2)\right]\cdot f_u \\ X'_v &= \left[(1+k_1R^2+k_2R^4+k_5R^6)\cdot X_v + 2k_3X_uX_v + k_4(R^2+2X_v^2)\right]\cdot f_v \end{aligned} \tag{16}$$

where $X_u = \frac{\vec{\omega}\cdot\vec{u}}{\omega_t}$ and $X_v = \frac{\vec{\omega}\cdot\vec{v}}{\omega_t}$.

**Algorithm 4:** Pixel Position Generation

$$\omega_u := \vec{\omega} \cdot \vec{u}$$
$$\omega_v := \vec{\omega} \cdot \vec{v}$$
$$X_u := \frac{\omega_u}{\omega_t}$$
$$X_v := \frac{\omega_v}{\omega_t}$$
$$R^2 := X_u^2 + X_v^2$$
$$poly := 1 + k_1 R^2 + k_2 R^4 + k_5 R^6$$
$$X'_u := \left[ poly \cdot X_u + 2k_4 X_u X_v + k_3 (R^2 + 2X_u^2) \right] \cdot f_u$$
$$X'_v := \left[ poly \cdot X_v + 2k_3 X_u X_v + k_4 (R^2 + 2X_v^2) \right] \cdot f_v$$
$$X := X'_v + c_v$$
$$Y := X'_u + c_u$$

The cameras are positioned such that their center positions along with their $(\vec{t}, \vec{u}, \vec{v})$ vectors are known in the 3-D space as described in Chapter 2.2. Camera placement is not an ideal process and hence the accurate values of the extrinsic camera parameters need to be estimated through another calibration process referred to as extrinsic calibration [4]. The estimated vector values of all cameras are stored in a LUT. For the ease of portability, all the LUT which contain camera related parameters are provided external access reconfiguration capability from the PC.

Obtained $(X'_u, X'_v)$ values do not necessarily coincide with the exact camera's pixel grid positions. The nearest pixel position technique is chosen for the first algorithmic step, since the linear interpolation does not provide better enough results to compensate for the additional design complexity. Hence, the intensity value of the closest pixel grid position is chosen as the intensity value of the $(X'_u, X'_v)$ position on each camera frame. In reality, this is realized by changing the values $(X'_u, X'_v)$, to the ones of the closest pixel.

The values $(X'_u, X'_v)$ are the coordinates of the pixel in the image frame with the system origin placed in the center of the frame. To move the coordinate system origin to the top-left the center correction is applied. The frame center coordinates are added to the $(X'_u, X'_v)$ values and the final coordinates $(X, Y)$ are obtained. The center coordinates are estimated in the camera calibration process and stored inside the LUTs.

The norm of the $(X'_u, X'_v)$ positions (*i.e.*, $R'$) is also calculated and sent to the interpolation block. This value is used for the vignetting [4] and exposure correction on the read pixel intensity values.

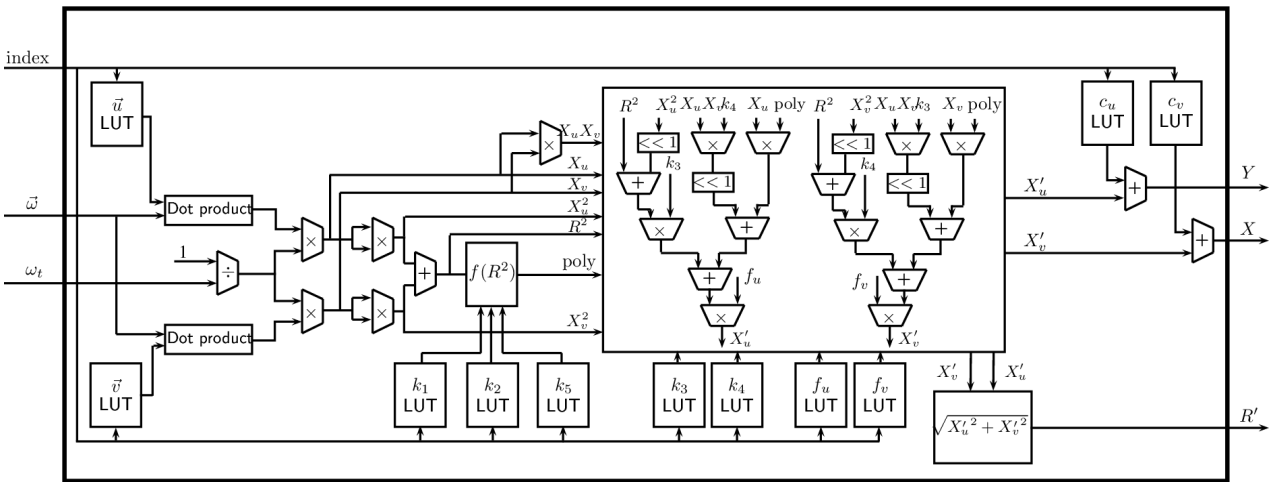



*Figure 4.9: Architecture of the pixel position generation block without pipeline registers shown.*

The internal architecture of the pixel position generation sub-block is shown in Figure 4.9.

Both divisions in the equation (7) have a common denominator $\omega_t$. Since the division operation uses a lot of resources, it is of utmost importance to try to reduce the number of division and thus use less multipliers in the design. In this case it is possible to use only one divider and two additional multipliers by first forming the $\frac{1}{\omega_t}$ value and then multiply with the needed values.

Resource sharing has also been applied through factorization of similar terms used in camera compensation in Algorithm 4. It can be noted that this block needs division operator, as well as efficient algorithm for polynomial calculation. The arithmetic division operators are implemented using a three stage (*i.e.*, iteration) convergence method described in the following chapter. Also, the chapter after will explain calculation of the polynomial function $f(R^2)$, expressed as poly term in Algorithm 4.

Each external SRAM on the FPGA board is partitioned into twenty equal size segments. Each segment corresponds to one image frame of pixel data acquired by the assigned camera. The SRAM has a one dimensional addressing scheme (*i.e.*, one address port). The two dimensional $(X, Y)$ positions are mapped to their corresponding addresses in the SRAMs through an address resolving unit shown in Figure 4.10 which implements the following mapping:

$$address = frame\_width \cdot (Y-1) + X - 1 \tag{17}$$

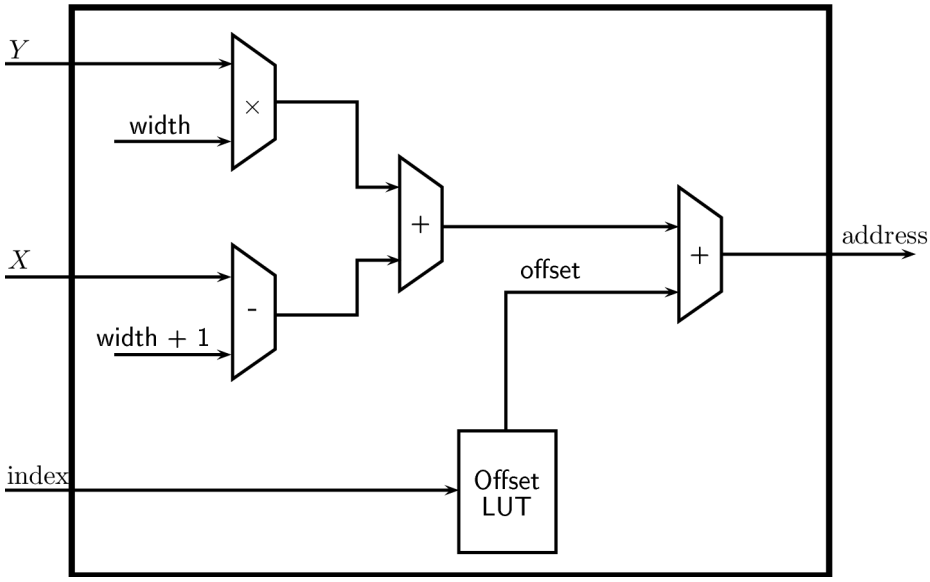



*Figure 4.10: Architecture of the address resolving unit.*

The pixel position generation block is the most process intensive entity in the overall design. Hence, the adequate pipeline register stages have been added along the combinational path to improve the performance of this block.

#### *4.2.4.1 Division operator*

The division operation is the most complex operation of all four basic arithmetic operations. Hence, there are a lot of algorithms for dividing which have different approaches and, in terms of hardware implementation, different performance and complexity. There are two main groups of algorithms in terms of the convergence speed:

- Linear convergence
- Quadratic convergence

The simplest algorithm with linear convergence is the "restoring algorithm", well-known as the "pen-and-paper" method. In this algorithm, the denominator is subtracted from the nominator until the difference becomes negative. Then, the last value is restored (hence the name) and the denominator is divided by 2 (logic bit shift right for 1 position). The process is repeated until the denominator becomes 1.

This algorithm is very simple, does not require any multiplication, but has a disadvantage of large number of steps. Two steps are needed to determine one quotient bit. One step is used for subtraction and the second for comparison and restoring. There are slightly different implementations, where the two steps are

combined into one, but then the speed performance is not high enough for such a constrained system as Panoptic camera.

Therefore, the quadratic convergence algorithms should be considered. One of the famous algorithms is based on calculation of the reciprocal of the denominator and then multiplying it by the nominator value. The reciprocal is calculated using the Newton algorithm for finding a zero of a polynomial. The polynomial is

$$f(x)=\frac{1}{x}-D \tag{18}$$

where $D$ is the denominator and $x$ is the wanted reciprocal. The Newton's iterative equation can be expressed as

$$x_{k+1}=x_k\left(2-D\cdot x_k\right) \tag{19}$$

From (19) it is obvious that in each iteration step requires 2 multiplications. If the goal is to calculate as fast as possible both multiplications and the subtraction should be calculated during the same clock cycle, which reduces the speed performance of the system. The performance can be enhanced by adding three pipeline stages between each iteration, but then the lower latency advantage over linear convergence algorithms is lost.

The improved version of the Newton's algorithm is proposed by Anderson et al. [13]. This algorithm transforms the iterative equation into three different ones. Thus, it is possible to parallelize the computation. In this algorithm both numerator and denominator are multiplied by approximation factor $f_k$. The iterative equations are given as:

$$\begin{aligned} f_k &= 2-t_k \\ x_{k+1} &= x_k\cdot f_k \\ t_{k+1} &= t_k\cdot f_k \end{aligned} \tag{20}$$

The iterations are repeated until the required precision is reached. In the iteration process the initial values are set to $x_0=N$ and $t_0=D$.

However, this algorithm also has certain additional constraints. In order to converge fast, the initial values should be normalized such that $D$ is close to 1, preferably $0.5\leqslant D<1$. Due to this fact the pre-scaling before each division should be implemented.

This algorithm has the disadvantage of using two multipliers per stage, but it needs only 3 stages to have an 18-bit precise result.

A comparison of the three mentioned algorithms in terms of stages and multipliers needed is given in Table 4.4. The comparison is done for 16-bit inputs.

*Table 4.4: Comparison of results of various division algorithms.*

| **Algorithm** | **Number of stages** | **Number of multipliers** |
|---|---|---|
| Restoring | 32 | 0 |
| Newton's | 3 | 7 |
| Anderson | 3 | 5 |

Anderson algorithm even needs one multiplier less than expected in theory due to the fact that the last $t_k$ value is not needed.

The chosen algorithm for implementation is the Anderson algorithm since the number of needed stages for the linear convergence algorithms is too high. Even with adding one pipeline stage per iteration, the Anderson algorithm is the best option.

#### *4.2.4.2 Polynomial estimation*

In (16) it can be observed that the equations have the same common factor. This factor is a polynomial of 3rd degree of variable $R^2$. Two algorithms for polynomial evaluation are discussed:

- Direct approximation
- Horner Scheme

Direct approximation algorithm evaluates the polynomial using its canonic form:

$$p(x) = \sum_{k=0}^{N-1} p(k) \cdot x_k . \tag{21}$$

This algorithm is not very suitable for hardware implementation due to the large number of multipliers it uses, both for the iteration steps and for calculating the powers of $x$.

The second and the most used algorithm is the Horner scheme. The Horner scheme evaluates the polynomial using the equation:

$$\begin{aligned} s(k) &= s(k+1) \cdot x + p(k) , \\ s(N-1) &= p(N-1) . \end{aligned} \tag{22}$$

The estimated result is $p(x) = s(0)$. This algorithm is by far better that the direct approximation, because of the fact that the needed number of multipliers is one less than the order of the polynomial.

A comparison of the two algorithms for a third order polynomial used in Panoptic system is given in Table 4.5.

*Table 4.5: Comparison of results of two polynomial estimation algorithms.*

| **Algorithm** | **Number of multipliers** |
|---|---|
| Direct approximation | 5 |
| Horner scheme | 2 |

### 4.2.5 Interpolation

The interpolation block conducts the second algorithmic step of the omnidirectional vision construction of the Panoptic camera. There are two interpolation modes available in this sub-block. The nearest neighbor technique is already accomplished in the pixel position generation block. This block only applies a vignetting/exposure correction on the read intensity values from SRAM. This correction is done by weighting (*i.e.*, multiplying) the intensity values. The weight factor is a function of distance R′, where R′ denotes the distance of a contributing point from the focal center on a camera image frame. This distance is evaluated from the following equation:

$$R' = \sqrt{X'^{2}_{u} + X'^{2}_{v}}, \tag{23}$$

and the correction function is denoted as $G(R')$. The output of this block in the nearest neighbor scheme for a pixel direction $\vec{\omega}$ is derived from

$$\angle(\vec{q}, \vec{\omega}) = \angle(c_j, \vec{\omega}) \cdot G(R'_j) \tag{24}$$

where $\angle(c_j, \vec{\omega})$ is the pixel intensity read from SRAM for $j^{\text{th}}$ camera.

The linear interpolation scheme incorporates all the contributing cameras intensity values through a linear combination. The weight of a contributing camera is an inverse function of its distance (*i.e.*, $r$) between its projected focal point and projected virtual observer point on the ω-plane. Hence the smaller the distance the higher its contribution is. The weights are also normalized to the sum of the inverse of all the contributing cameras distances.

The linear interpolation in mathematical terms is expressed in (25).

$$\mathcal{L}(\vec{q},\vec{\omega})=\frac{\sum_{i\in I}\frac{1}{r_i}\mathcal{L}(c_i,\vec{\omega})\cdot G(R'_i)}{\sum_{i\in I}\frac{1}{r_i}}, \tag{25}$$

where $I$ is a set of contributing camera indexes.

For the purpose of implementation (25) is rewritten in (26). The $a_i$ resembles the weight of the $i^{\text{th}}$ contributing camera.

$$\mathcal{L}(\vec{q},\vec{\omega})=\sum_{i\in I} a_i\cdot\mathcal{L}(c_i,\vec{\omega}) \tag{26}$$

Hence $a_i$ is represented as in (27) for each contributing camera in pixel direction $\vec{\omega}$.

$$a_i=\frac{\frac{1}{r_i}\cdot G(R'_i)}{\sum_{i\in I}\frac{1}{r_i}} \tag{27}$$

The architecture of the linear interpolation block is shown Figure 4.11. The reason for first creating the sum of weights is in lower resource usage. If the linear interpolation algorithm is implemented as a straightforward following of the mathematical equations the logical order would be to multiply the weights with the pixel intensities and divide in the end with sum of the weights. However, this implementation requires separate dividers for each channel for the final division.

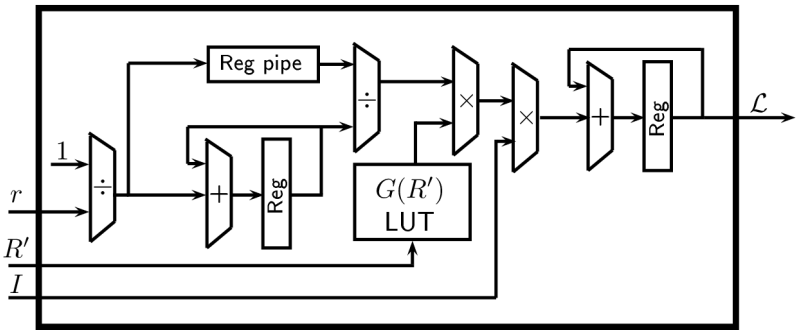



*Figure 4.11: Architecture of the linear interpolation block.*

Thus, creating a universal array of weights $a_i$ which uses only one divider is a much better option even though a few more pipeline stages are needed.

The weight generation is followed by a multiply and accumulator unit to conduct the linear interpolation operation as shown in Figure 4.11. The correction

function $G(R')$ is a Gaussian function for the presented implementation and has been realized using a LUT. The same filter is applied to each frame of each camera, thus this LUT is common for the whole system and will not take a lot of resources on FPGA.

One of the important additions to the pure implementation of the equations is the addition of the bypass branch in case that only one camera has the direction $\vec{\omega}$ in its field-of-view. Thus, an unnecessary division of type $x/x$ is omitted. This is important due to the fact that the division algorithm for smaller values of $x$ give less than 1 as the quotient which results in a darker reconstructed image.

Furthermore, the output format is again RGB565. It should be noted that in order to reduce the number of bits from 16 to 5 or 6, the *round* operation is used instead of *floor* that is used throughout the system. Here the round is necessary, because of smaller number of bits and error in only one bit can result in a large error in the end and low SNR of the reconstructed image.

The pseudo code of the efficient linear interpolation algorithm is shown in Algorithm 1.

**Algorithm 8:** Linear interpolation calculator

$w_{acc} := \sum_{i \in I} \frac{1}{r_i}$
**for all** $i \in I$ **do**
  $a_i := \frac{1}{r_i} \cdot \frac{G(R'_i)}{w_{acc}}$
**end for**
**for all** color_channels **do**
  $\angle_{RGB} := \sum_{i \in I} I_{RGB} \cdot a_i$
**end for**

### 4.2.6 Register Map

The Panoptic camera is a fully programmable system. It can be accessed from PC and be given commands through developed GUI software. However, inside the FPGA a PicoBlaze microcontroller is instantiated for the control operations and it accesses the internal registers of the camera. Through those registers, the camera can be fully parametrized.

A certain register bank has already been developed for the basic functioning of the Panoptic camera. However, the omnidirectional vision reconstruction needs an additional set of registers which is shown in Table 4.6. The full register map of the single board version can be seen in Appendix A: Full Register Map. A multiple board

system has the same register map, however, not all the registers are used on all boards. They can also be seen in Appendix A: Full Register Map.

The registers are 8-bit in size. Hence, for the 16-bit values two registers should be allocated. The registers starting from address 0x20 until 0x2B are registers used to set the parameters of the Angle generation block. These values set the wanted resolution and the wanted part of the full image.

The rest of the registers are used for reading from or writing to the LUTs in the system. The data that is to be written to the LUT is first written to the registers 0x2C and 0x2D from the PC. The address in the LUT where the data is to be written is in register 0x2E. The registers from 0x2F to 0x31 contain W/R strobes for the LUTs. Only one bit in these three registers can be set to '1' at the same time. This prevents accidental writing of the same data to the multiple LUTs. The LUTs also have a multiplexer in the input which allows either microcontroller or the Image processing unit to access it. The select signals for these multiplexers are set in register 0x32. By default, all bits in these four registers are set to 0 (LUTs are set in *read* mode by the Image processing unit).

The LUTs can be read the same way and the data is stored to registers 0x33 and 0x34 and later sent to the PC via USB link.

*Table 4.6: Concentrator FPGA register map.*

| **Address** | **Name** | **Default value** |
|---|---|---|
| 0x20 | Theta angle step LB | 0x04 |
| 0x21 | Theta angle step HB | 0x00 |
| 0x22 | Theta angle max value LB | 0x00 |
| 0x23 | Theta angle max value HB | 0x04 |
| 0x24 | Theta angle min value LB | 0x02 |
| 0x25 | Theta angle min value HB | 0x00 |
| 0x26 | Phi angle step LB | 0x02 |
| 0x27 | Phi angle step HB | 0x00 |
| 0x28 | Phi angle max value LB | 0xFF |
| 0x29 | Phi angle max value HB | 0x07 |
| 0x2A | Phi angle min value LB | 0x00 |
| 0x2B | Phi angle min value HB | 0x00 |
| 0x2C | LUT data LB | 0x00 |
| 0x2D | LUT data HB | 0x00 |
| 0x2E | LUT address | 0x00 |
| 0x2F | LUT W/R enable register 1 | 0x00 |
| 0x30 | LUT W/R enable register 2 | 0x00 |
| 0x31 | LUT W/R enable register 3 | 0x00 |
| 0x32 | LUT read select (system/uC) | 0x00 |
| 0x33 | LUT read data LB | -- |
| 0x34 | LUT read data HB | -- |

Apart from adding the new registers, the register on address 0x0A is extended. This is the *Image processing application mode* register. This register is the most important one in the Panoptic camera configuration and it will be explained in more detail.

The bit description of the registers is shown in Figure 4.12 and the meaning of each bit in Table 4.7.


*Figure 4.12: Image processing application mode register.*

*Table 4.7: Bit description of Image processing application mode register.*

| **Name** | **Bit position** | **Description** |
|---|---|---|
| Mode | 7..6 | Main operating mode of the camera:<br>'00' – Regular mode<br>'11' – Omnivision mode |
| F | 5 | Gaussian filtering:<br>'0' – off<br>'1' – on |
| I | 4 | Interpolation mode:<br>'0' – nearest neighbor<br>'1' – linear interpolation |
| C | 3 | Color mode:<br>'0' – Monocolor<br>'1' – RGB color |
| E | 2 | Endiannes of the data:<br>'0' – little endian<br>'1' – big endian |
| Option | 1..0 | Options in regular operating mode:<br>'00' – Single camera capture<br>'01' – All camera capture<br>'10' – Single channel video<br>'11' – Dual channel video<br>Options in omnivision mode:<br>'00' – Video mode<br>'11' – Capture mode |

## *4.3 Central FPGA*

Each concentrator FPGA board can interface with maximum twenty cameras. In order to support more cameras and increase the output bit rate of the Panoptic camera, multiple concentrator FPGA boards must be incorporated. Hence omnidirectional construction work load must be distributed and run in parallel on all concentrator FPGA boards and combined after. A central FPGA is required to receive the output data from the concentrator FPGA boards, apply the last stage processing and transfer the image data to a PC for display.

The second algorithmic step mentioned in Chapter 3.2 assumes the knowledge of all contributing cameras for any pixel direction $\vec{\omega}$. Each concentrator FPGA board only contains the parameter information of the cameras that are connected to it. Hence each concentrator FPGA board only constructs the omnidirectional vision for the set of cameras that are interfaced with it. The output of the $j^{\text{th}}$ concentrator FPGA board for pixel direction $\vec{\omega}$ and virtual observer position $\vec{q}$ is denoted as $\angle_j(\vec{q},\vec{\omega})$. Concentrator FPGAs also provide the distance information to the central FPGA. The distance information is either the minimum distance $r_i$ for the direction $\vec{\omega}$ in case of nearest neighbor technique or sum of reciprocal distances in case of linear interpolation (denominator in equation (25)). The central FPGA uses this data to complete the second algorithmic step of the omnidirectional vision reconstruction algorithm.

The central board has the same architecture as the concentrator boards. The major difference is the much simpler architecture of the image processing unit. By observing Figure 2.6 in case of omnidirectional vision reconstruction on the central board, the data is not stored in the SRAMs. Both pixel values and distances/weights are forwarded to image processing unit directly from the multiplexing stage.

Inside the image processing unit there is only the interpolation block. In case of the nearest neighbor algorithm, the distances are compared and the pixel corresponding to the minimum distance is outputted. For linear interpolation algorithm, there is an additional step in the beginning to restore sums in numerator and denominator in equation (25). This step includes multiplying pixel value $\angle_j(\vec{q},\vec{\omega})$ by its weight (sum of distances). Thus, a needed value is obtained. The rest of the process follows the equations (26) and (27) and it is the same as on the concentrator boards. Additionally, the interpolation block on the central FPGA does not include the pixel intensity correction LUT $G(R')$ in Figure 4.11.

The central board receives the data from the concentrator boards via LVDS link. The board has one receiver per board, thus the transfer speed is very high and there is no need for multiplexing until the very end. On the transmission side, the board uses USB controller and sends the final data to the PC.

# 5 Results and Comparison

## *5.1 FPGA Implementation*

The architecture proposed in the Chapter 4 has been implemented and functionally confirmed on the concentrator and central FPGA. Each concentrator is interfaced with twenty cameras.

The proposed architecture for the concentrator FPGA searches for the contributing cameras for each pixel direction $\vec{\omega}$ sequentially in each clock cycle. The rest of the computation path in the proposed architecture is pipelined. The total pipeline latency of the implemented design is $C_{latency}$=120 clock cycles in case of linear interpolation algorithm, which is in the order of 100 that was assumed in Chapter 2.3.2. In the nearest neighbor, latency is even less.

The maximum estimated operating frequencies after synthesis are given in Table 5.1. For the sake of comparison, the frequencies are given separately for the Panoptic concentrator FPGA and Panoptic central FPGA. Also, post place and route maximum frequencies are given. For place and route procedure, a full user constraint file is made with the exact pin allocation and the worst-case conditions.

It should be noted that post place and route timing report meets the constraints only when Xilinx ISE implementation strategy is set to be optimized for timing performance using physical synthesis.

*Table 5.1: Estimated maximum frequencies.*

| **Module** | **Post synthesis** | **Post P&R** |
|---|---|---|
| Concentrator FPGA | 145.18 MHz | 133.45 MHz |
| Central FPGA | 144.41 MHz | 138.06 MHz |

The post place and route estimated frequencies confirm that the projected clock frequency of $f_{FPGA}$=133$MHz$ is really achievable. A drop in the maximum frequency after place and route is expected due to the exact placing of the components in the FPGA. The drop in the central FPGA is lower, because of the smaller design and device utilization, thus the path delays are much smaller.

The resource utilization summary of the concentrator FPGA is given in Table 5.2. The similar utilization values are for the single board version. Also, the numbers are slightly different for 2 concentrator FPGAs, because one of them has only one LVDS transmitter, while the other has 1 LVDS receiver and 2 transmitters. However, in general perspective that difference is negligible.

*Table 5.2: Full system utilization summary in the concentrator FPGA.*

| **Resources** | **Used** | **Available** | **Utilization** |
|---|---|---|---|
| Occupied Slices | 3823 | 7200 | 53% |
| Slice LUTs | 8964 | 28800 | 31% |
| Slice Registers | 10192 | 28800 | 35% |
| BlockRAM/FIFO | 19 | 48 | 39% |
| DSP48E | 44 | 48 | 91% |

Firstly, it should be noted that the DSPs in this design are used purely for multiplication purposes. The number of multiplication operations is much higher than the available DSPs inside the used FPGA. Thus, the rest of the multipliers are made using the logical elements. However, the Xilinx ISE synthesis tool often has a problem with synthesizing circuits with a need for more DSPs than available. It happens that the DSP number exceeds the available number and even though the synthesis successfully finishes the place and route procedure is impossible. The solution that was applied here is to restrict the number of used DSPs to 90%, thus the synthesis tool cannot accidentally exceed 48 DSPs which is the maximum for the used FPGA.

An interesting comparison can be made by observing Table 5.2 and Table 2.1. The system without the omnidirectional vision construction option uses almost half of the resources of the full system. Thus, the designed image processing unit is approximately the same size as all of the other peripherals together. This is just another confirmation of the complexity of the reconstruction algorithm implementation.

The central FPGA, as explained, requires fewer resources than the concentrator FPGA. They are given in Table 5.3.

*Table 5.3: Full system utilization summary in the central FPGA.*

| **Resources** | **Used** | **Available** | **Utilization** |
|---|---|---|---|
| Occupied Slices | 1194 | 7200 | 16% |
| Slice LUTs | 2038 | 28800 | 7% |
| Slice Registers | 2560 | 28800 | 8% |
| BlockRAM/FIFO | 4 | 48 | 8% |
| DSP48E | 14 | 48 | 29% |

## *5.2 Omnidirectional Vision Reconstruction*

The design is tested using a constructed Panoptic camera shown in Figure 2.5 and the smaller single board version. For initial tests 30 images were captured using the multiple board camera system. A detailed testbench file was written emulating the whole acquisition system (cameras), memory storage (ZBT SRAMs) and the full camera system design is tested including the USB link to PC and the operation of the PicoBlaze controller.

The simulation was done on the same images that were used in Matlab model explanations in Chapter 4.1. In order to be as real as possible only first 20 cameras were used. The resolution of the reconstructed image is 256 $\times$ 1024 and the linear interpolation is used in the second algorithmic step. The reference and the reconstructed images are shown in Figure 5.1 as well as their differences.

Two simulations were done with different sets of cameras, emulating two concentrator boards. In the end the output of those boards is provided as the input for the testbench file of the central board. The final reconstructed image on the central board is shown in Figure 5.2.

It is important to determine the quality of the reconstructed images. A comparison of Signal-to-Noise Ratio of the Matlab fixed-point model, single board and multiple board Panoptic cameras is given in Table 5.4.

*Table 5.4: SNR comparison of the model and actual realization.*

| **Name** | **Red channel** | **Green channel** | **Blue channel** |
|---|---|---|---|
| Matlab model | 25.55 | 30.05 | 25.88 |
| Single board | 26.88 | 30.88 | 27.11 |
| Multiple board | 26.74 | 30.66 | 26.87 |

The Panoptic camera can work with much higher resolutions in the capture mode, *i.e.*, to reconstruct a single image instead of a video signal. In this mode even HD images can be reconstructed. An example of the reconstructed HD image of resolution 768 $\times$ 1024 is shown in Appendix B: HD Image Reconstruction.

*a) Ideal (reference) reconstruction*

*b) Panoptic camera reconstruction*

*c) Difference between the two reconstructed images*

*Figure 5.1: Reconstruction simulation results using linear interpolation algorithm.*

*a) Ideal (reference) reconstruction*

*b) Panoptic camera reconstruction*

*c) Difference between the two reconstructed images*

*Figure 5.2: Final reconstruction simulation result using linear interpolation algorithm.*

# 6 Real-Time Testing

The simulations have verified that the camera design is correct and it gives an adequate reconstructed image. The next step is to test the design in real-time FPGA implementation. The platform used for tests is the single board 3-floor model with 15 cameras installed on the surface. As explained on the previous chapters, this model has a direct connection to the PC via USB link.

## *6.1 Panoptic Media Software*

A specific software is written to support the operation of the camera[2]. The main purpose is to display the reconstructed video signal. However, apart from display the software is used to set up the internal registers of the camera sensors and Panoptic camera internal registers. The software is able to recognize the camera model attached by reading the version-ID register of the camera and set up the registers accordingly. Setup data is kept in XML format generated directly from Matlab models created previously.

When a camera board is connected and turned on, the software initializes it and is ready for operation. The screenshot of the application is shown in Figure 6.1.

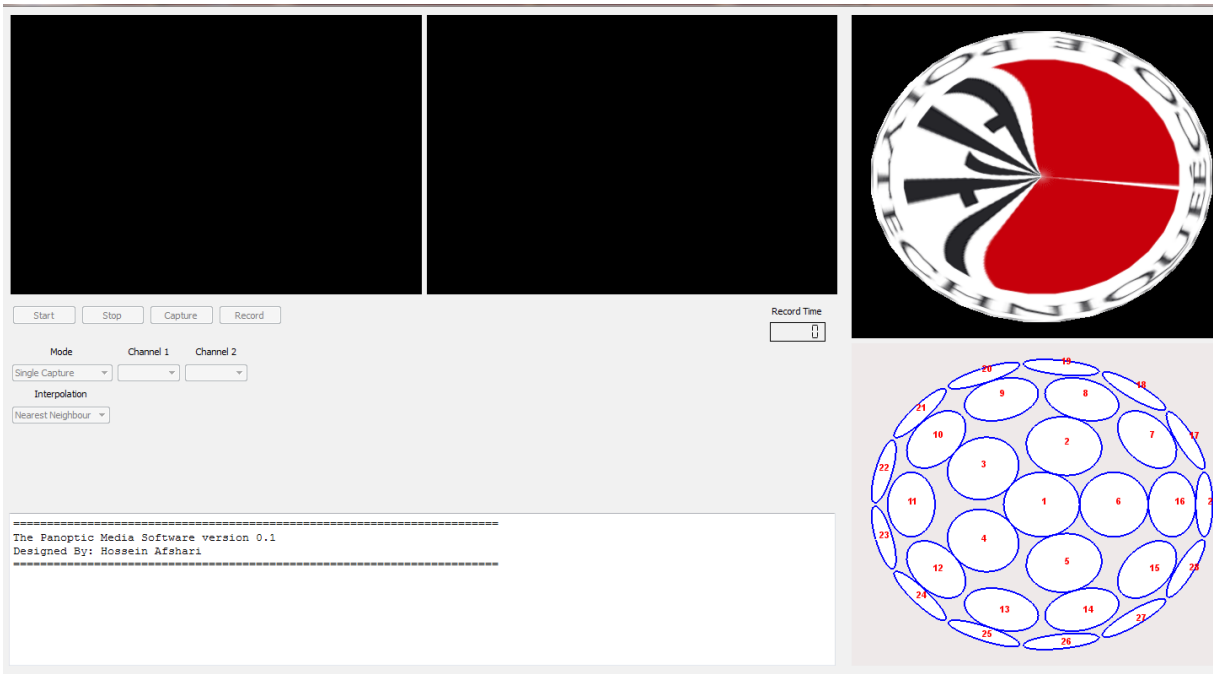



*Figure 6.1: The Panoptic Media application.*

2 The author of the major part of the software program is Hossein Afshari from LSM

Two windows in top-left corner are used for displaying single and dual channel videos form the camera sensors. The camera channel can be either chosen from a drop-down menu or by clicking on the camera in the bottom-right corner of the software.

The software currently supports six modes: single camera capture, all camera capture, single camera video, dual camera video, omnidirectional vision capture and omnidirectional vision video. The last two modes are the most important ones. When the software and the camera itself are running in one of these two modes, two windows in top-left corner are replaced by one window displaying the omnidirectional reconstructed image/video. Additionally, the image/video is displayed on a sphere showed in top-right corner. The sphere can be rotated by dragging mouse over it. The full 3D reconstruction can be actually seen on this sphere with no distortions that can be seen in planar display. The software also allows the user to change between the reconstruction algorithms by another menu.

Apart from just displaying, the software has buttons used to initiate the capture and to record a reconstructed video signal. The recorded video length is currently set to 10 seconds and uses MPEG-2 standard for compression.

The omnidirectional reconstructed image is shown in Appendix C: Reconstructed Test Images. A few black pixels can be noticed in the left part of the images. These pixels are due to incorrectly produced hardware structure thus the cameras do not have a full coverage of the surroundings. However, the camera works despite these problems and with a better produced camera it will have a full coverage.

Additionally, a few recorded omnidirectional reconstructed videos are available on the enclosed CD.

# 7 Discussion

During this project the image processing unit of the Panoptic camera is designed and tested. This unit reconstructs the omnidirectional vision from 30 cameras installed on a hemisphere in the shown model.

Firstly, the Matlab model was made and it was determined that a 16-bit precision system is precise enough for an accurate reconstruction. A fixed-point model was compared to the floating-point and satisfactory results were obtained.

The actual hardware design of the processing unit is done after. Customized implementations over several algorithms for mathematical operations were realized and the full block is completed afterward. The block is then embedded into an already existing architecture.

The main constraint of the design was to be operable on 133 MHz frequency and to provide output data each 20 cycles. A very long pipelined system was an only option in this case and the results of the place and route assured that the system design was good and that the mentioned constraints are all met.

Apart from physical constraints it is important to obtain information about the reconstructed image quality and visual appearance. Visual appearance is very good, without noticeable artifacts or very disturbing errors. In the image analysis it was shown that there are losses compared to the ideal reconstruction, which was expected. However, in the implemented design SNR was even a bit higher than predicted by the Matlab model. This is due to a tweak in implementation where some of the unnecessary calculations in the linear interpolation mode were bypassed and actual values were provided without loss in precision during the division operations. In the multiple board system SNR was a bit lower due to an additional division, but it is still higher than expected.

Finally, the maximum resolutions for this camera should be mentioned. In video mode with 25 fps, a maximum resolution with 20 cameras per FPGA is 0.25 MP. In the capture mode, on the other hand, there are no timing constraints and the number of bits allocated for pixel grid determination sets the boundary. The maximum resolution of the static image is $2048 \times 2048$.

# 8 Conclusion

The Panoptic camera project can be regarded as a camera of the future. 3D real time imaging is something we currently do not have, even though technology is improving every day.

One small part of this project was realized as semester project in spring 2010 and it is now completed and presented in this master thesis report. This report introduces a real-time omnidirectional camera referred to as the Panoptic camera. The algorithmic details for the omnidirectional vision reconstruction of the Panoptic camera are presented, as well as the architecture of a FPGA based system for the real-time deployment of the reconstruction algorithm. The hardware architecture details of the algorithm are provided and its implementation is described in detail. A built system along with its captured results is illustrated.

Future work should be concentrated on possible extensions of the system. This would mean adding options to the Panoptic camera such as depth map estimation using images obtained from multiple cameras, object tracking in the video signal or increasing the number of cameras to predicted 100 for better image quality.

A new board with a distributed parallel architecture is designed and it will be an upgraded version of the current model with much more capabilities.

In the end, I would like to thank the LSM team who gave me the opportunity to work on such an interesting project, and especially Hossein Afshari for help, guidance and patience during my work on this project. I would also like to thank professors Alexander Schmidt and Yusuf Leblebici from LSM, Luigi Bagnato and professor Pierre Vandergheynst from LTS2 and Laurent Jacques from University of Louvain for their great contribution to this project and thesis.

Apart from people working on this project, I would like to thank my parents and my family for supporting me throughout my studies and teaching me everything I know. Certainly, my studies and the work on my thesis would not be the same if it had not been for my dear friends that I spent with the previous two years. In no specific order they are Nikola Katić, Radisav Ćojbašić, Miroslav Popović, Teodora Kostić, Mihailo Velimirović, Alen Stojanov and Eray Molla, as well as many others that I may have forgotten. The end is reserved for the most important person in my life during the last 2 years, without whom nothing would be the same, Sezin Afşar.

# Bibliography

[1] Zbikowski, R., "Fly like a fly [micro-air vehicle]," *IEEE Spectrum* , vol. 42, no. 11, pp. 46-51, 2005.

[2] "Virtex-5 Family Overview DS100," Xilinx, February, 2009. Visited on June 17, 2011.

[3] Levoy, M. and Hanrahan, P., "Light field rendering," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, *SIGGRAPH '96*, pp. 31-42, ACM, New York, NY, USA, 1996.

[4] Hartley, R. I. and Zisserman, A., *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, Cambridge, UK, 2004.

[5] Raboud, D., Vandergheynst, P. and Jacques, L., "The Panoptic Camera - Plenoptic interpolation in an omnidirectional polydioptric camera," Master's thesis, EPFL, 2009 [Online]. Available: http://infoscience.epfl.ch/.

[6] Tierney, J., Rader, C. and Gold, B., "A digital frequency synthesizer," *IEEE Transactions on Audio and Electroacoustics* , vol. 19, no. 1, pp. 48-57, 1971.

[7] Bateman, A. and Paterson-Stephens, I., *The DSP handbook: algorithms, applications and design techniques*, Prentice-Hall, Upper Saddle River, NJ, 2002.

[8] Volder, J. E., "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers* , vol. EC-8, no. 3, pp. 330-334, 1959.

[9] Walther, J. S., "A unified algorithm for elementary functions," in *Proceedings of the May 18-20, 1971, spring joint computer conference*, *AFIPS '71 (Spring)*, pp. 379-385, ACM, New York, NY, USA, 1971.

[10] Meyer-Baese, U., *Digital signal processing with field programmable gate arrays*, 3rd ed. Springer, Berlin, Germany, 2007.

[11] Heath, T., *A history of Greek mathematics*, The Clarendon Press, Oxford, UK, 1921.

[12] Andraka, R., "A survey of CORDIC algorithms for FPGA based computers," in *Proceedings of the 1998 ACM/SIGDA sixth international symposium on Field programmable gate arrays*, *FPGA '98*, pp. 191-200, ACM, New York, NY, USA, 1998.

[13] Anderson, S. F., Earle, J. G., Goldschmidt, R. E. and Powers, D. M., "The IBM System/360 Model 91: Floating-Point Execution Unit," *IBM Journal of Research and Development* , vol. 11, no. 1, pp. 34-53, 1967.

# Appendix A: Full Register Map

| ADDR | Definition | Type |
|---|---|---|
| 0x00 | USB transfer size LB | W/R |
| 0x01 | USB transfer size MB | W/R |
| 0x02 | USB transfer size HB | W/R |
| 0x03 | I2C slave address | W/R |
| 0x04 | I2C target register address | W/R |
| 0x05 | I2C register data | W/R |
| 0x06 | Byte swap in word mode at BIT.1 | W/R |
| 0x07 | I2C enable channel 7 downto 0 | W/R |
| 0x08 | I2C enable channel 15 downto 8 | W/R |
| 0x09 | I2C enable channel 19 downto 16 | W/R |
| 0x0A | Image processing application mode | W/R |
| 0x0B | camera channel one | W/R |
| 0x0C | camera channel two | W/R |
| 0x0D | camera interface reset at BIT.1 | W/R |
| 0x0E | | W/R |
| 0x0F | | W/R |
| 0x10 | I2C prescale LB register | W/R |
| 0x11 | I2C prescale HB register | W/R |
| 0x12 | LED general purpose output | W/R |
| 0x13 | I2C read data | R |
| 0x14 | Switch general purpose input | R |
| 0x15 | hsync pixel count LB | W/R |
| 0x16 | hsync pixel count HB | W/R |
| 0x17 | USB fifo full threshold LB | W/R |
| 0x18 | USB fifo full threshold HB | W/R |
| 0x19 | USB fifo address | W/R |
| 0x1F | | Reserved |
| 0x20 | Theta angle step LB | W/R |
| 0x21 | Theta angle step HB | W/R |
| 0x22 | Theta angle max value LB | W/R |
| 0x23 | Theta angle max value HB | W/R |
| 0x24 | Theta angle min value LB | W/R |
| 0x25 | Theta angle min value HB | W/R |
| 0x26 | Phi angle step LB | W/R |
| 0x27 | Phi angle step HB | W/R |
| 0x28 | Phi angle max value LB | W/R |
| 0x29 | Phi angle max value HB | W/R |
| 0x2A | Phi angle min value LB | W/R |
| 0x2B | Phi angle min value HB | W/R |
| 0x2C | LUT data LB | W/R |
| 0x2D | LUT data HB | W/R |
| 0x2E | LUT address | W/R |
| 0x2F | LUT W/R enable register 1 | W/R |
| 0x30 | LUT W/R enable register 2 | W/R |
| 0x31 | LUT W/R enable register 3 | W/R |
| 0x32 | LUT read select (system/uC) | W/R |
| 0x33 | LUT read data LB | R |
| 0x34 | LUT read data HB | R |
| 0xC0 | I2C write command | W |
| 0xC1 | I2C read command | W |
| 0xC2 | Image retrieve command | W |
| 0xC3 | Image hold | W |
| 0xC4 | Image release | W |

# Appendix B: HD Image Reconstruction

# Appendix C: Reconstructed Test Images