ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
TECHNICAL UNIVERSITY OF CRETE

# Reconfigurable Logic-Based System for Image Processing of Fishery Nets


Author:
Theofilos Zacheilas

Thesis Committee:
Prof. Apostolos Dollas
Prof. Michalis Zervakis
Dr. Euripides Sotiriades


A thesis submitted in fulfillment of the requirements for the DIPLOMA of Electrical and Computer Engineering
in the
School of Electrical and Computer Engineering
Microprocessor and Hardware Lab

Chania

June, 2021

# Abstract

The aim of this thesis is to develop an embedded system which is able to detect the deformed holes of fishery nets in an input video. In recent times, there is a need for production of automation systems which save time and manual efforts of human resources. The simulation of the system is done in MATLAB, whereas the final design is implemented and accelerated in Vivado HLS tool. The produced hardware component can fit into an actual FPGA platform (ZCU102), in order to accelerate the corresponding software part with less energy consumption. There are two main topics that affect the appearance of aquatic environment, the lighting variation with haze and the variation in size of holes along each frame. There is a progressive evolution on the production of final results. Firstly, the use of an edge-aware filtering, called 'guided image', takes place to face the haze removal and the lighting smoothing of each frame. Furthermore, morphological opening is used for detail enhancement. The image, also, breaks in windows for local identification of deformed holes and overcoming the problem in variation of holes' size along different parts of the frame. Finally, combination of consecutive frames happens for further improvements. Speaking with numbers, the dimensions of each frame are $270_{height} x 480_{width}$, and hardware achieves an average of 181 frames per second, and a speed-up of about 12.7 times compared to MATLAB. The significance of this study is that it can actually cope with each of the obstacles through a series of observations, help the user understand the circumstances and become the base for the creation of a real time system that operates well in every occasion.

# Περίληψη

Ο σκοπός αυτής της πτυχιακής εργασίας είναι να ανπτύξει ένα ενσωματωμένο σύστημα, το οποίο μπορεί να ανιχνεύσει τις παραμορφωμένες οπές των διχτυών ιχθυοκαλλιέργειας σε ένα βίντεο εισόδου. Τα τελευταία χρόνια, υπάρχει ανάγκη για παραγωγή συστημάτων αυτοματισμού που εξοικονομούν χρόνο και χειροκίνητες προσπάθειες ανθρώπινου δυναμικού. Η προσομοίωση του συστήματος γίνεται σε MATLAB, ενώ ο τελικός σχεδιασμός υλοποιείται και επιταχύνεται στο εργαλείο Vivado HLS. Το παραγόμενο υλικό μπορεί να χωρέσει σε μία πλατφόρμα FPGA (ZCU102), προκειμένου να επιταχύνει το αντίστοιχο τμήμα λογισμικού με λιγότερη κατανάλωση ενέργειας. Υπάρχουν δύο κύρια θέματα που επηρεάζουν την εμφάνιση του υδάτινου περιβάλλοντος, η παραλλαγή φωτισμού με την ομίχλη και η διακύμανση του μεγέθους των οπών κατά μήκος κάθε στιγμιότυπου του βίντεο. Υπάρχει μια προοδευτική εξέλιξη στην παραγωγή των τελικών αποτελεσμάτων. Πρώτον, πραγματοποιείται η χρήση ενός φίλτρου με γνώμονα την ακμή, που ονομάζεται «φίλτρο καθοδηγούμενης εικόνας», για να αντιμετωπίσει την αφαίρεση της ομίχλης και την εξομάλυνση φωτισμού κάθε στιγμιότυπου. Επιπλέον, το μορφολογικό άνοιγμα χρησιμοποιείται για την ενίσχυση της λεπτομέρειας. Η εικόνα, επίσης, σπάει σε παράθυρα για τοπική αναγνώριση παραμορφωμένων οπών και ξεπερνά το πρόβλημα στην παραλλαγή του μεγέθους των οπών κατά μήκος διαφορετικών τμημάτων του στιγμιότυπου. Τέλος, πραγματοποιείται ο συνδυασμός διαδοχικών στιγμιότυπων για περαιτέρω βελτιώσεις. Μιλώντας με αριθμούς, οι διαστάσεις κάθε στιγμιότυπου είναι $270_{\upsilon\psi o\varsigma} x 480_{\pi\lambda\alpha\tau o\varsigma}$ και το υλικό επιτυγχάνει κατά μέσο όρο 181 στιγμιότυπα ανά δευτερόλεπτο, με επιτάχυνση περίπου 12,7 φορές σε σύγκριση με τη MATLAB. Η σημασία αυτής της μελέτης είναι ότι, βασικά, μπορεί να αντιμετωπίσει κάθε ένα από τα εμπόδια μέσω μιας σειράς παρατηρήσεων, να βοηθήσει τον χρήστη να κατανοήσει τις συνθήκες και να γίνει η βάση για τη δημιουργία ενός συστήματος πραγματικού χρόνου που λειτουργεί σωστά σε κάθε περίσταση.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Explanation of main problem

In modern industrial era, it is necessary to use methods for the quality control of products and their production machinery. Very often, automated techniques are used for immediate fault detection, to reduce the production of defective products, but also to avoid the possible risks that these can cause.

Image processing techniques are commonly used by the industry for automation of quality control. Their purpose is to detect faults and defects in the products produced, using systems which have some type of camera. The control system provides information that describe the product with the desired result, and this is compared each time with the produced image. Automation is important, because many times control is difficult or time-consuming for humans and the likelihood of wrong conclusion is high.

One such case is the finding of holes in aquaculture nets. Fish farms use special enclosures, made of piles and nets, to trap fish. The presence of holes in the nets can lead to loss of some fish. Systematic control is important but also extremely difficult and time consuming. This is because the nets are in the sea and should either be removed from the water and be checked or be inspected by a diver. The automation of this process will help reduce the time and cost of maintaining aquaculture nets.

## 1.2 Contribution of University's previous thesis

The exploration of this problem started from Nick Badogiannis' thesis, where initial steps were made about how to approach it. Actually, that thesis focused on normal circumstances, where static images were considered to be the main problem and proposed power efficient algorithms. It also provided very important techniques that are necessary in observation of fishery nets, and are able to solve the initial problems, before moving to more complicated situations.

## 1.3 Contribution of this thesis

The main objective of this thesis is to make an interference to the realistic underwater model. This model can provide many challenges due to the various factors that can lead to the uncertainty of observation. As it is well known, underwater areas suffer from hazy regions and unpredictable lighting effects. These characteristics appear in every image or video that is taken underwater, so a global method is demanded to be found, in order to make a more clear and stable statement about the behavior of the underwater physiology.

This dissertation has to do with an underwater video, which was taken at day from HCMR - Hellenic Centre for Marine Research (in greek: ΕΛ.ΚΕ.Θ.Ε) and provides a circular view, while moving around an area, where fishing nets are placed. Further information about the video will be shown in results of next chapters. The final project has the following capabilities.

- Deal with a Realistic Lighting Model.
- Process a video instead of still Images.
- Make an assessment of Regions of Uncertainty.

### 1.3.1 More Realistic Lighting Model

The video of processing was taken under the sun light and it has quite uneven regions when it comes to the lighting effects. This is too important, because, a global method has to be found in order to normally observe any area of interest, even when differentiation in luminance appears, rather than concentrate on particular approaches that limit the analysis of alternative types of incoming data (e.g. video taken at night rather than day, sea nature and animals that prevent clear view of nets, too much haze etc).

### 1.3.2 Video instead of Still Images

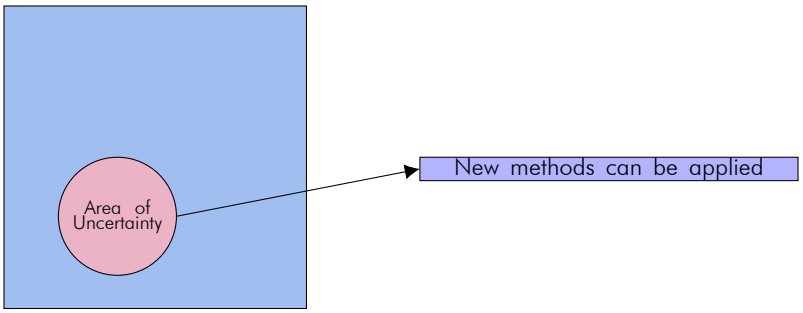
Compared to previous estimations that contained still images with standard characteristics, this one refers to data packs that form consecutive frames. Frames need a fast implementation to achieve, at least, normal frames per second (fps) and satisfy the user's needs. Still, they can operate to user's advantage and affect the final result. For example some frames can be combined to verify if an area is faulty or not, which is finally adopted in the overall project.

### 1.3.3 Assessment of Regions of Uncertainty

Due to the existence of blur and other similar factors, vision becomes too unspecified and the results too unpredictable. So, the current system is assigned to detect these areas and mark them as unidentified, because it cannot provide any safe result for them. Adding this specification to the

system can contribute to insertion of new methods afterwards, which can resolve the uncertainty of these areas and produce more explicit and secure results.

Figure 1.1: Representation of uncertainty areas




## 1.4 Structure of the Thesis

The current thesis consists of six chapters including the current one. The basic content of each of the following chapters is described below.

Chapter two refers to relevant work that has taken place in the past. It contains previously published methods that refer to underwater lighting processing, which was the main problem in this thesis. It also includes the way of calculating the size of each hole in the net.

Chapter three presents the modeling of the full system in MATLAB. Each method, which was finally included, is described in a different section.

Chapter four contains the hardware design which is targeted for FPGA systems. Specifically, it shows the top level design hierarchy, what tools were used to achieve it, the simulation results and comparisons with the MATLAB model.

Chapter five makes the performance evaluation for the FPGA design compared to the MATLAB model. It mentions the part that is parallelized, the speed up that FPGA achieves, and the energy consumption.

Chapter six includes the overall conclusions and possible future work, based on the techniques that are provided in this thesis, in order to contribute to the manufacturing of a final system that is able to work in real time.

# Chapter 2

# Relevant Works

## 2.1 University's Previous Thesis Proposal

The previous work on this project, proposed from Nick Badogiannis, analyzed the template matching and edge detection algorithms. Template matching involves sum of absolute differences, sum of squared differences, cross-correlation, normalized sum of squared differences, and normalized cross-correlation. Edge detection uses Sobel filtering. Edge detection proves to be quite faster than template matching, due to less computations, with a little more energy consumption. It also proves to be more flexible in noise interference. However, both types of algorithms lack the ability to operate appropriately to more difficult circumstances. For example, when haze and lighting noise participate at input data, the false positive and false negative percentages grow up quite significantly.

## 2.2 Hazy Image Model

It was mentioned that the processing video includes hazy areas, that complicate the estimation of lighting effect. The main problem, definitely, needs solution in this area, as any further processing is going to fail in the correct notification of underwater environment. The widely used model to describe the formation of a hazy image is:

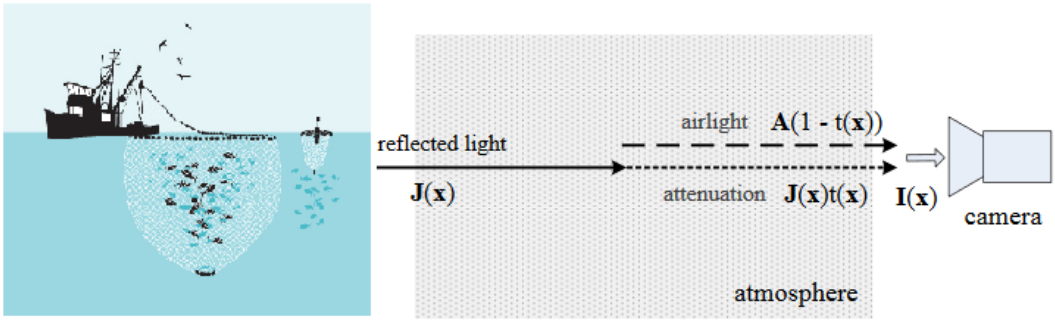
$$\mathbf{I}(x) = \mathbf{J}(x)\,t(x) + \mathbf{A}(1 - t(x)),$$

where variables are explained in the following:

- x=(x, y) is a 2D vector representing the coordinates (x, y) of a pixel's position in the image.
- I represents the hazy image observed. I(x) is a 3D RGB vector of the color at a pixel.
- J represents the scene radiance image. J(x) is a 3D RGB vector of the color of the light reflected by the scene point at x. It would be the light seen by the observer if this light was not through the haze. So, the scene radiance, J, is, often, referenced as a haze-free image.

- t is a map called transmission or transparency of the haze. t(x) is a scalar in [0,1]. Intuitively, t(x) = 0 means completely hazy and opaque, t(x) = 1 means haze-free and completely clear, and 0<t(x)<1 means semi-transparent.
- A is the atmospheric light. It is a 3D RGB vector, usually, assumed to be spatially constant. It is often considered as "the color of the atmosphere, horizon, or sky"

Figure 2.1: A macro physical picture of the haze imaging model (image from Olive Ridley Project).




The goal of haze removal is to recover J, A, and t from I, in order to make an evaluation of haze factor.

## 2.3 Edge-aware/Haze Removal Filtering

Filtering is one of the most important operations in computer vision/graphics. Simple linear translation-invariant (LTI) filters like Gaussian filter, and Sobel filter are widely used in image blurring/sharpening, edge detection, feature extraction and others. The kernel of these filters is explicitly defined. LTI filters can also be designed implicitly. For example, solving a linear system $\mathrm{A}x = b$ is equivalent to filtering the map b by $\mathrm{A}^{-1}$, because, $x = \mathrm{A}^{-1}b$, where the filtering kernel $\mathrm{A}^{-1}$ is not explicitly computed.

One of the main purposes of the filters applied in the images of this thesis interest, is supposed to preserve edges, because, this is the way to make the separation between foreground and background pixels. The kernels of LTI filters are spatially invariant and independent of any image content. The bilateral filter is perhaps the most widely used edge-aware filter. However, the bilateral filter has some limitations, like the gradient reversal artifacts and its speed, which is very low when the kernel radius, r, is large. Other methods require quantization (approximation) to achieve satisfactory speed, but at the expense of quality degradation. In sum, a fast and high quality explicit filter is still demanded in many edge-aware applications.

The presentation of "guided image filtering", takes place in this section. Guided image filtering is treated in this application as an edge-aware and haze removal process, that combines the information of two images, namely, a filtering input image (denoted as p) and a guide image (denoted as I), and generates one filtering output image (denoted as q). The filtering input p

determines the colors, brightness, and tones of the filtering output q, whereas the guide image I determines the edges of q.

In this thesis, Guided filter, replaces the soft matting step, which is required in haze removal for the estimation of transmission map, t. It is used for single image texture smoothing or denoising, so it is treated as a special case of guided image filtering, where the filtering input (p) and the guide (I) are identical, because, the colors and the edge information are from the same image, but combined unequally. Though the guided filter is an edge-aware filter like the bilateral filter, it avoids the "gradient reversal" artifacts. These artifacts of the bilateral filter may appear when the filtering process is for detail enhancement/extraction.

The key assumption of the guided filter is a local linear model between the guide I and the filtering output q. It is assumed that q is a linear transform of I in a window $w_k$ centered at the pixel k:

$$q_i = a_k \mathrm{I}_i + b_k, \forall i \in w_k,$$

where $(a_k, b_k)$ are linear coefficients assumed to be constant in $w_k$. A square window of a radius r is used.

This local linear model ensures that q has an edge only if I has an edge, because $\nabla q = a\nabla \mathrm{I}$. But the output q should have similar colors or tones with the input p. So a solution has to be found that minimizes the difference between q and p. After some steps this solution indicates the following equations:

$$a_k = \frac{cov_k(\mathrm{I}, p)}{\sigma_k^2 + \epsilon}$$

$$b_k = \bar{p} - a_k \mu_k$$

Parameter $\epsilon$ is for regularization but it is important, because, it controls the smooth degree. Specifically, $\epsilon$ is analog with smoothness, $cov_k(\mathrm{I}, p)$ is the covariance of I and p inside the window k, $\bar{p}$ is the mean value of p, and $\mu_k$ is the mean value of I. The linear model is applied to all local windows in the entire image. But a pixel i is covered by many local windows $w_k$, so the value of q may change when it is computed in different windows. A simple strategy is to average all the possible values of $q_i$. So after computations of $(a_k, b_k)$ for all the windows $w_k$ in the image, the filter output q is given by:

O(n) time algorithm for guided filter

$$q_i = \frac{1}{|w|}\left(\left(\sum_{k \in w_i} a_k\right)\mathrm{I}_i + \left(\sum_{k \in w_i} b_k\right)\right)$$

It is proven that, solving the last three equations (instead of computing any kernel that is a

combination or simplification of them), whose time computation is O(n), is the best scenario for time consumption.

There is another filter inside guided filter called, box filter, which is responsible for the summary of values inside the radius of the window of guided filter. In depth, the overall operation of box filter is shown in the next figures, where s is source matrix/image, c is an intermediate/cumulative matrix, and d is destination matrix.

Figure 2.2: Box Filter's cumulative sum over Y axis

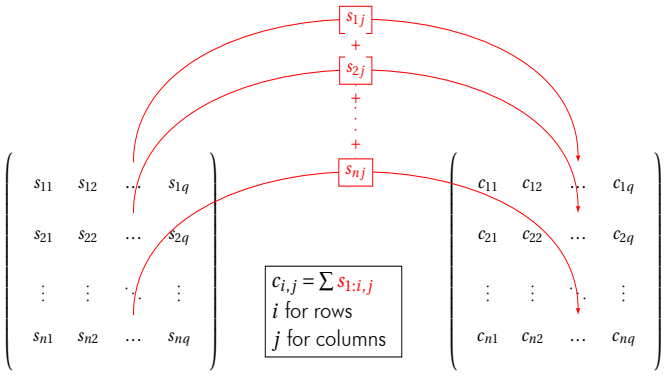
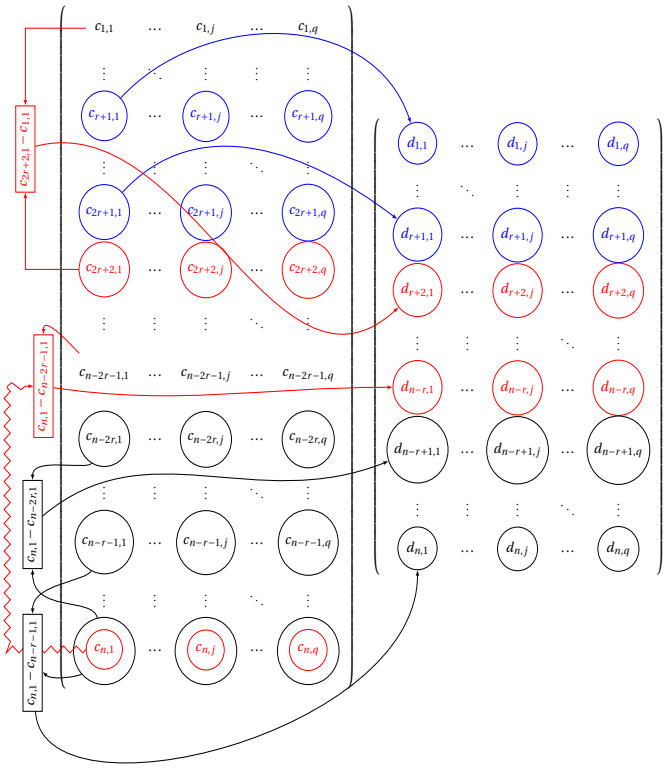



Figure 2.3: Box Filter's difference over Y axis



MATLAB code for figure 2.3:
d(1:r+1, :) = d(1+r:2*r+1, :)

d(r+2:hei-r, :) = d(2*r+2:hei, :) - d(1:hei-2*r-1, :)
d(hei-r+1:hei, :) = d(hei, :) - d(hei-2*r:hei-r-1, :)

The same procedure follows over X axis.

Figure 2.4: Box Filter's cumulative sum over X axis

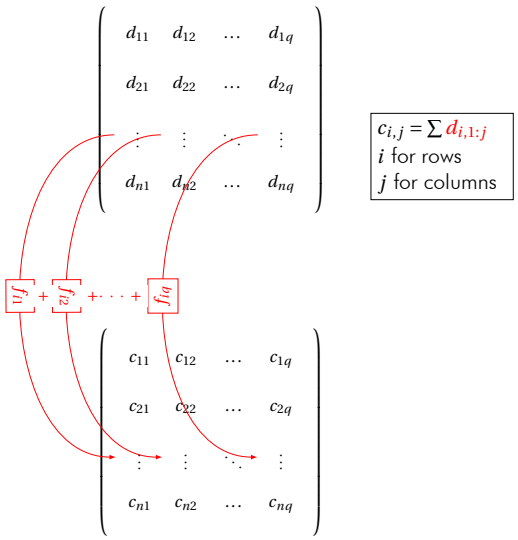
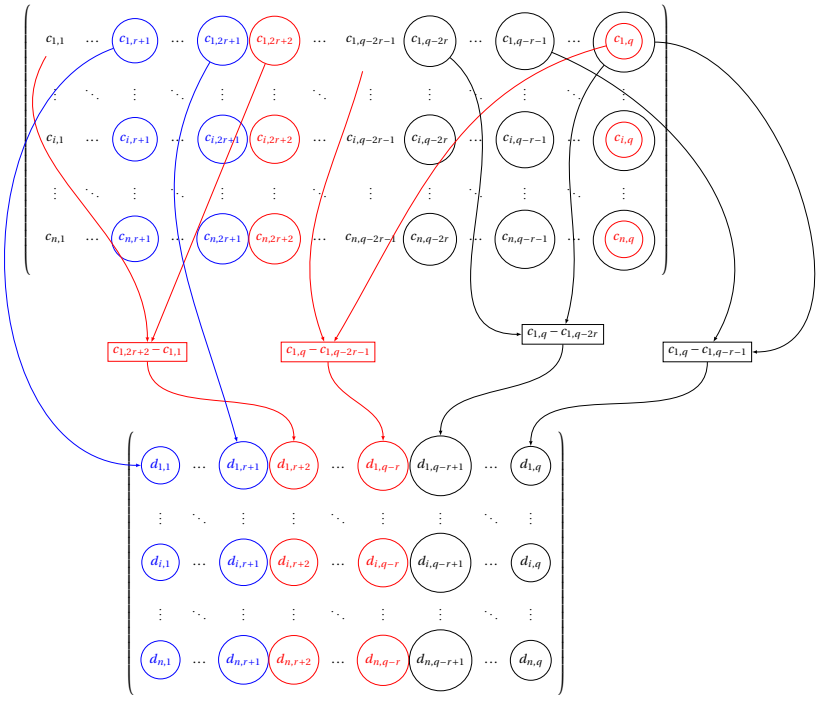



Figure 2.5: Box Filter's difference over X axis



MATLAB code for figure 2.5:
d(:, 1:r+1) = c(:, 1+r:2*r+1);
d(:, r+2:wid-r) = c(:, 2*r+2:wid) - c(:, 1:wid-2*r-1)
d(:, wid-r+1:wid) = c(:, wid) - c(:, wid-2*r:wid-r-1)

## 2.4 Connected Component Labeling

This section is crucial and unavoidable in this kind of research. Specifically, it was initially known that every hole should, somehow, be recognized as a distinct object, which has its own attributes that separate it from any other hole. This segment is, also, proposed in Badogiannis' thesis, and is assimilated and changed according to the current needs. Connected component labeling, in short CCL, performs this operation. Supposing that nets form the black pixels and holes the white pixels in figure 2.7 (a), then holes can be separated between each other, with the intervention of nets, and receive specific labels, based on the connectivity parameter which can help in calculating valuable features, like their size.

The important part of this algorithm is the observation of the *connectivity* parameter. This has to be either *eight* or *four*. Connectivity four means that the pixel which is going to receive a label, takes into account its north and west adjacent pixels, whose labels have already been set previously and form with it a vertical or horizontal line. Similarly, connectivity eight also allows diagonal lines to be formed. The fact that there is a need to reduce, as much as possible, the possibility of detecting wrongly faulty holes, especially with such an undetermined luminance, it is rational that the connectivity had to encounter less adjacent pixels, forming an object/hole.

Figure 2.6: CCL's example for a processed pixel (orange) and its neighborhood (o) with connectivity configuration, (a) 4 and (b) 8

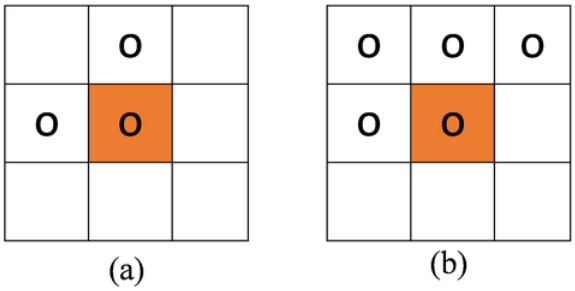
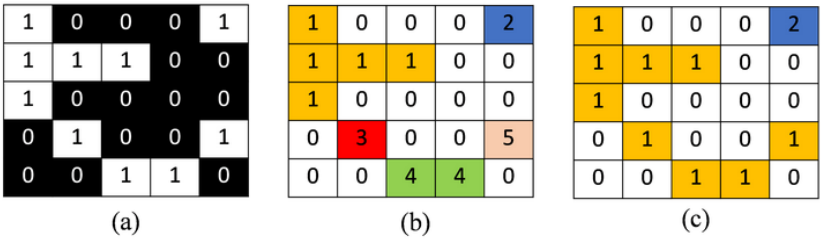



Figure 2.7: Binary input image (a), (b) labelling result using 4-connectivity, (c) labelling result using 8-connectivity



# Chapter 3

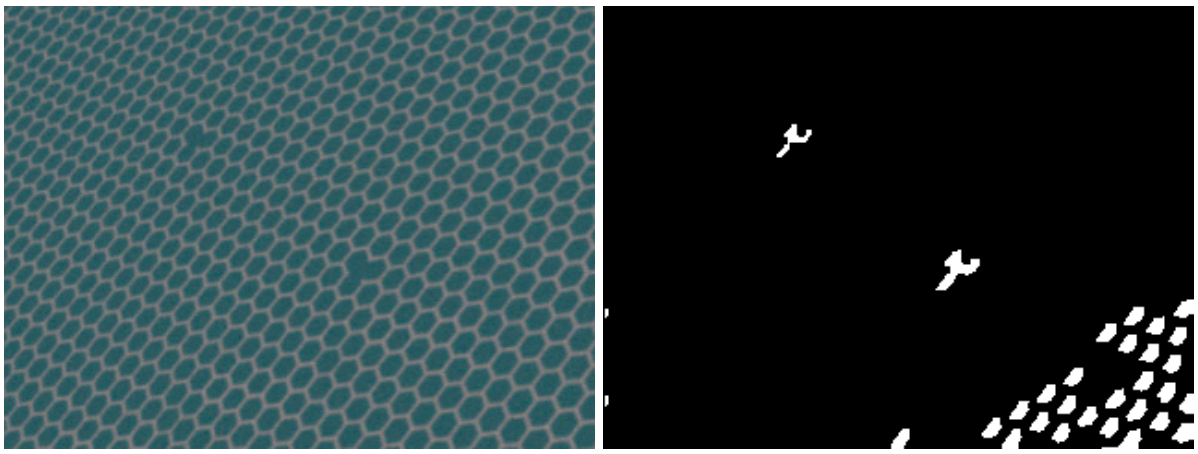
# Modeling of the System

## 3.1 Previous thesis' open problems

In Badogiannis' thesis, the method that was implemented is edge detection with sobel filtering. This method behaves greatly in normal circumstances and it is more simple and straightforward in terms of complexity. However, it leads to problems when it comes to real time processing. These problems relate with the following findings.

- There is identification of more faulty holes than the real ones, because processing considers the entire image as a unique unit and distortions in different parts can lead in wrong results.

Figure 3.1: Open problem with different holes' sizes



- The project focuses on smooth images without change of lighting, and when lighting varies, results, again, have issues.

Figure 3.2: Open problem with change of lighting

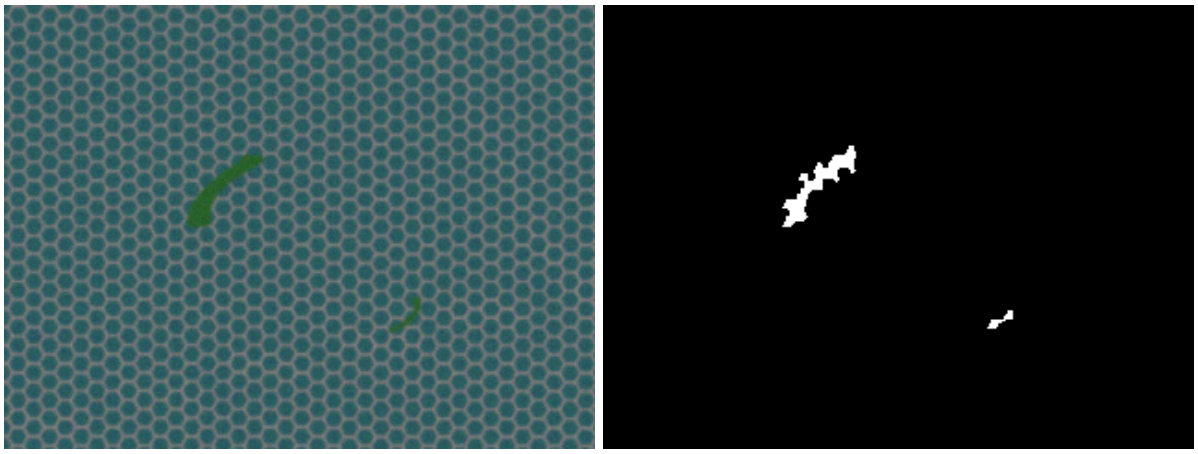
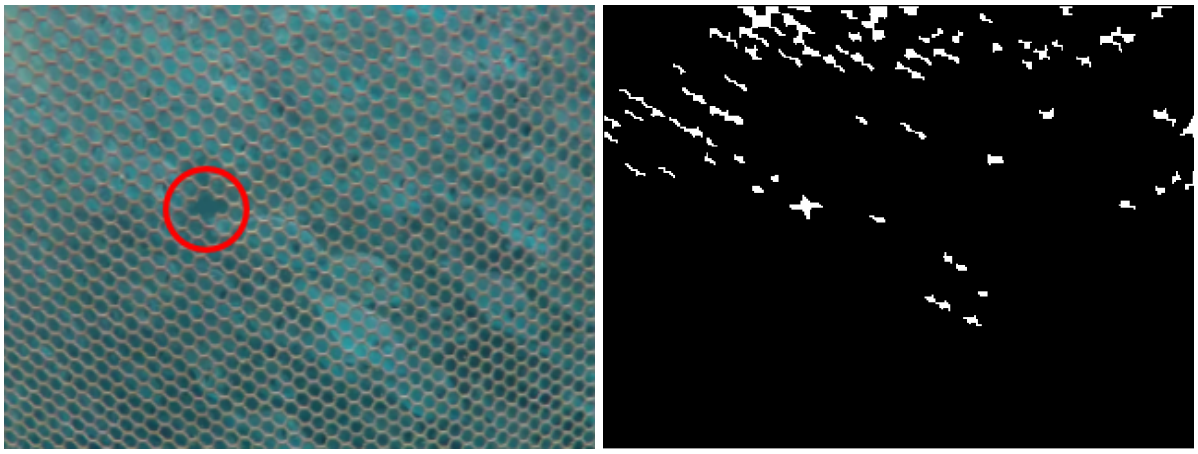


Figure 3.3: Open problem with presence of fishes



- It has to do with static images without the need for any further foundations and experiments.

## 3.2 Modeling of the System in Matlab

MATLAB tool was used for modeling of the system. The main function receives an input video and, inside a loop, decomposes it into a sequence of frames. The initial dimensions of the frames are $1080_{height}x1920_{width}$ and are resized to $270_{height}x480_{width}$ in order to aim at higher frame rates and save a lot of resources. Each frame gets passed in a sequence of functions. Each function serves a special role and forms a different stage of computation for the extraction of final results in each frame, as shown below:

i. *Image smoothing with edge detection and separation of colors,brightness and tones between water and nets.*

ii. *Noise relief with composure of more compact nets for better detection of holes.*

iii. *Estimation of holes' sizes.*

iv. *Separation of holes in regions.*

v. *Detection of defective holes in each region.*

vi. *Display of all faulty holes and nets in the final image.*

vii. *Combination of consecutive frames for safer conclusions.*

Each of these functions is analyzed more deeply and independently in the aftermath of this chapter.

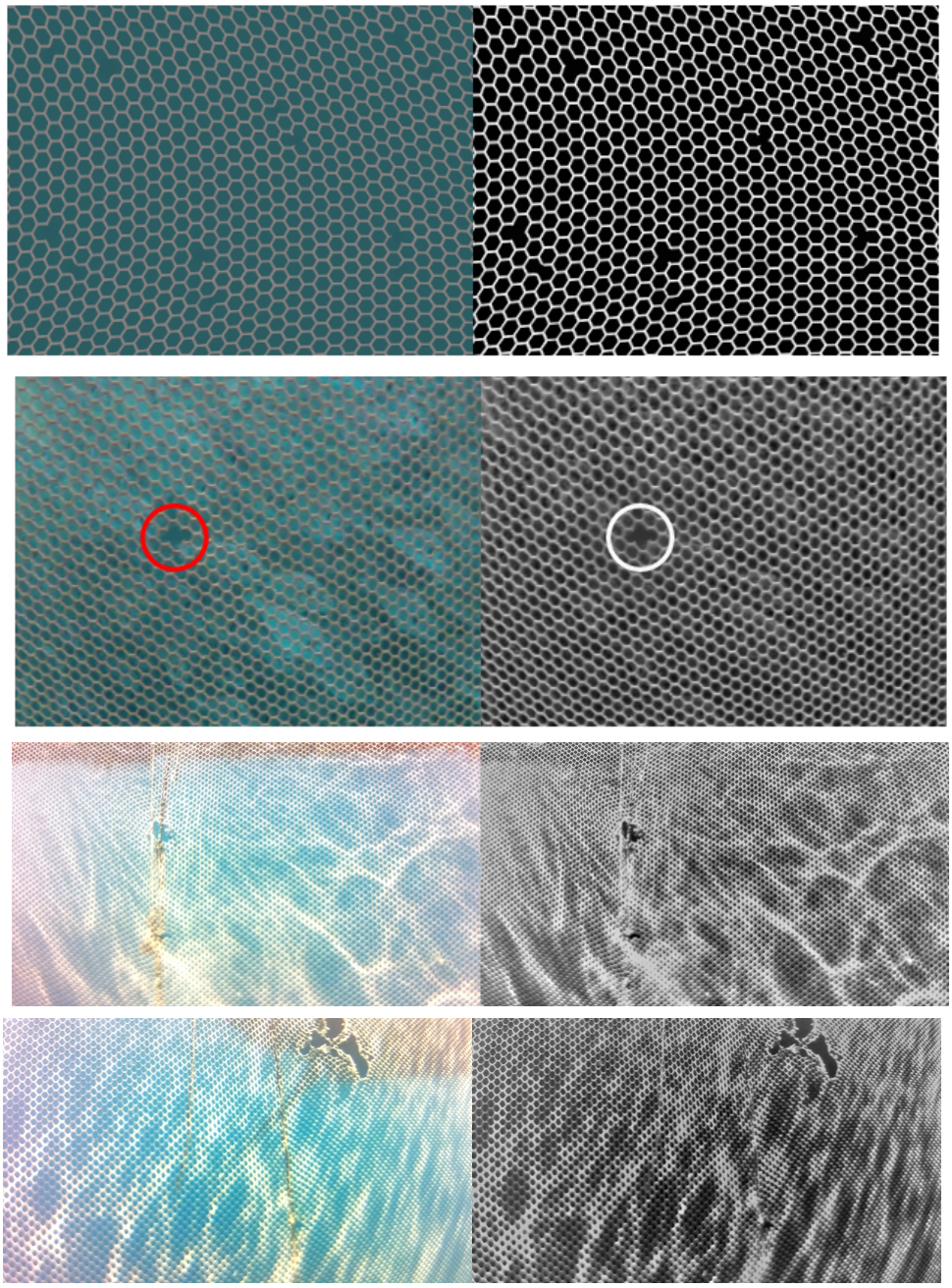
### 3.2.1 Image Enhancement

The initial and most crucial part has to deal with fog, blur, and haze which appear in the processing video. It is necessary to make a stable and clear assessment between the different elements that appear in the video. These elements are the water, underwater population, and fishery nets. There are not many options about the avoidance of underwater population, as its ignorance would be useful. However, the current video does not contain any fishes, so they will not prevent the observation of the nets. Although, there is an example of an image with fishes that interfere to the main scene and their existence is finally resolved. As a result, fishery nets and water remain the only factors that affect the image with their own characteristics. In underwater conditions, the main characteristic is the absorption of sun light that enters the sea. Nets seem to absorb more sun light than water does. This is straightforward in the video, because, nets are more illuminated, as they encircle sun light, and holes provide a passage for it. illumination greatly determines the shape and the joints along the length and width of the net. On the other side, holes between nets do not hold much illumination and appear darker. Previous observations lead to the following statements. *Nets will be called foreground, due to the greater absorption of sun light, whilst water will be the background.* Their separation is the initial step in the procedure.

This is done with function ex_enhancement, which implements the behavior of guided filter, mentioned in chapter two. *The key point here is that the differences of illumination are more visible in the red channel of ex_enhancement's output image.* Because of this, computations inside ex_enhancement are executed only in the red channel of input image.

Below are represented some examples of the red channel of the image, which are produced through ex _enhancement.

Figure 3.4: Initial images and their red channels from ex_enhancement



### 3.2.2 Morphological Operations

The presence of environmental noise within the underwater area makes it difficult to observe the nets, despite the estimation of image enhancement. Actually, nets reflect the rays of the absorbed sun light and block the boundaries of holes. This situation reduces the size of holes and is going to prevent the appliance of strict limits for their separation from rest objects. It is clear that the boundaries of holes have to be extended. A morphological operation can have such an impact, in order to make the image qualify for the final stage of computation. The operation that is finally

used is morphological opening. This operation is built on dilation and erosion. In mathematical morphology, opening is the dilation of the erosion of a set, $\mathrm{I}$, using the same structuring element $elem$ for both operations.

$$\mathrm{I} \circ elem = (\mathrm{I} \ominus elem) \oplus elem$$

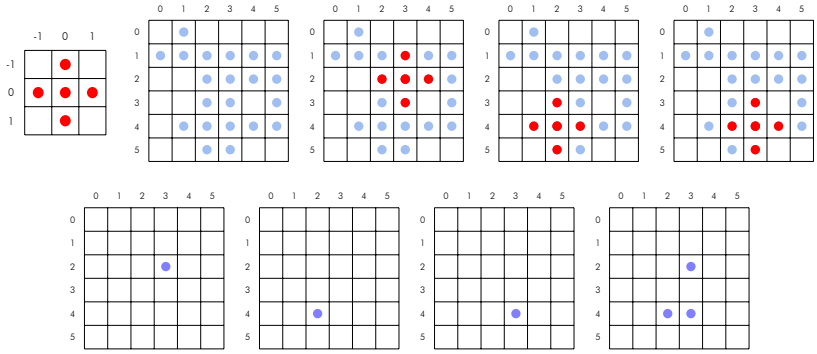
Alternatively

$$\mathrm{I} \circ elem = \cup\{elem_w : elem_w \subseteq \mathrm{I}\}$$

On the one hand, the basic idea of *erosion* is that it erodes away the boundaries of foreground (hole) object. So, as the kernel slides through the image (as in 2D convolution), a pixel in the original image (either 1 or 0) will be considered 1, only if all the pixels under the kernel are 1, otherwise it is eroded (made to zero). It is useful for removing small white noises, detaching two connected objects, and others.

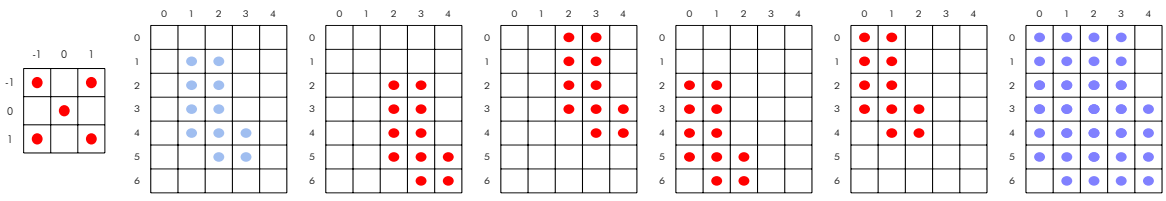
Erosion: $\mathrm{A} \ominus \mathrm{B} = \{w : \mathrm{B}_w \subseteq \mathrm{A}\}$,
where A represents an image, B represents a structuring element, w represents a pixel in A, and $\mathrm{B}_w$ represents a structuring element, B, that can be formed in A and have central pixel, w.

Table 3.1: Erosion example



On the other hand, *dilation* is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel/structure is '1'. So it increases the size of foreground objects.

Table 3.2: Dilation example



Normally, in cases like noise removal, erosion is followed by dilation. Erosion removes white noises, but it also shrinks the foreground objects. So the objects need to be dilated. Since noise is gone, it won't come back, but the object area will be increased. Opening is also useful in joining broken parts of an object.

This function succeeds in morphological noise removal. Opening removes small objects from the foreground (usually taken as the bright pixels) of an image, placing them in the background. This technique can also be used to find specific shapes in an image.

Opening can be used to find things into which a specific structuring element can fit, like edges and corners. In this case, after some tests, the element that was used is the disk. This was the best element that could best represent the holes. Disk is equivalent to the following matrix:

Table 3.3: Disk structuring element as a matrix

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 |

Table 3.4: Image Opening parameters

| | |
|---|---|
| Radius (px) | 5 |
| Structure | disk |
| Structure Type | 0⇔ rectangle |

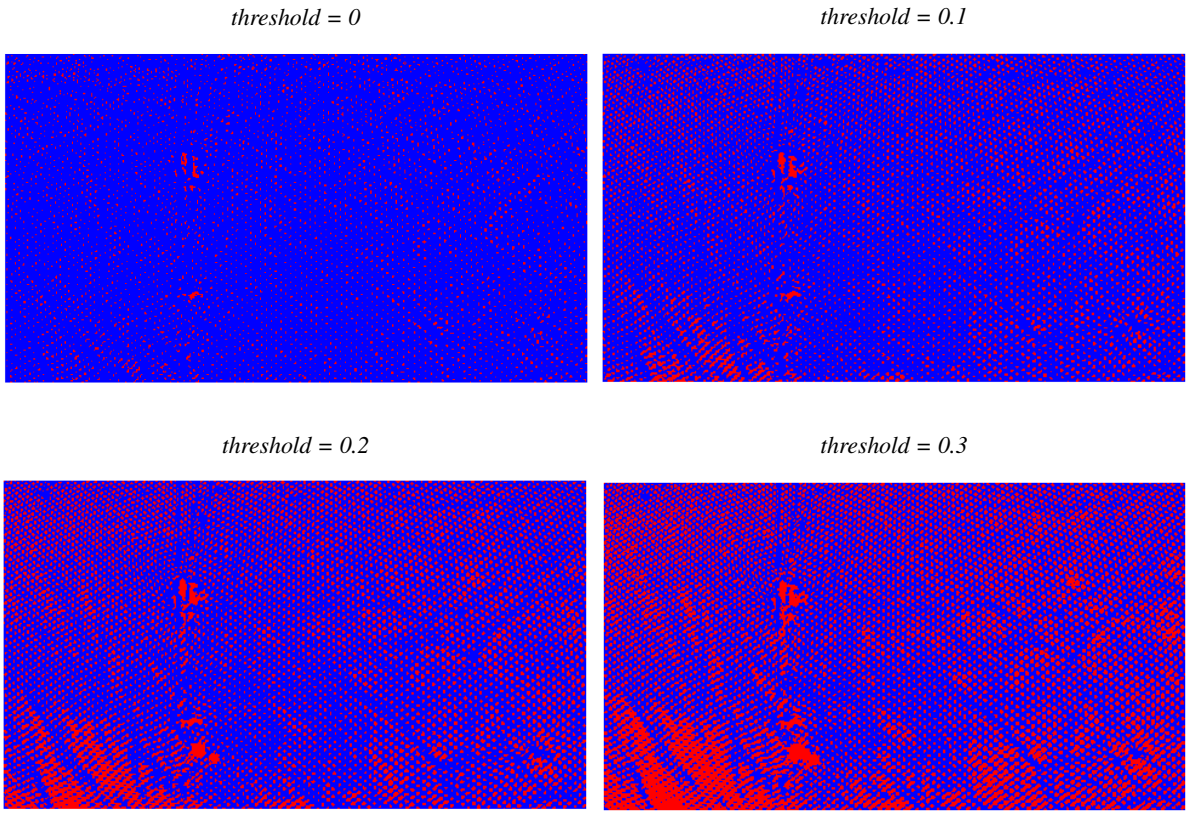
Function imopen refers to the *Morphological Opening* of an area. This means that imopen performs the erosion operation followed by dilation operation. It tends to remove the bright foreground pixels from the edges of regions of foreground pixels. Its impact is to safeguard foreground region that has similarity with the structuring component, or that can totally contain the structuring component while taking out every single other area of foreground pixels. Opening operation is used for removing internal noise in an image.

According to the above, the subtraction of imopen result will increase the brightness of the foreground and will provide a more clear separation from the background. The structuring element which was selected for 'imopen' is the *disk*. The greater the choice of radius, the more blurred the result. So a small radius value had to be chosen. In this case, a generally reliable value was found equal to *five*.

### 3.2.3 Simple Threshold/Holes identification

This section is based on an illumination value and, according to it, a binary illumination image is produced. In the new image, pixels with values lower than illumination threshold represent the places where the light . Black is the opposite - foreground. The larger the illumination the better the holes will be recognized, but the less the nets are going to be with possible misjudgment and the appearance of non existent holes. In contrast, the small ,yet reliable, illumination appliance shrinks the holes enough that they are not recognizable at all or are not presented entirely and appear in an unsuitable and useless way to the user.

Figure 3.5: Images with illumination threshold




### 3.2.4 Adaptive_Threshold / Nets identification & Assessment of Regions of Uncertainty

This technique presents the morphology of the environment. This can help the user understand better how nets are distributed and overcome the problem with regions of uncertainty which constitutes an innovation in this thesis.

Adaptive_Threshold function can perform two different types of filtering. It receives an input image, a window/kernel size, variable 'C', and a number (0 or 1). The last argument determines the type of filtering which is either mean (average) or median filtering. In this case, average filtering is used, based on the given window size. The window gets constructed as a square, according to the size argument. The function can also shift each time the average or median value with the contribution of offset 'C' and produce a local threshold, either mean-C or median-C. Then it performs the separation of input image pixels to either '0' or '1'. In each window, pixels with value greater or equal than local threshold will become '1', whereas the rest '0'. The offset may vary due to the need of the user to correspond to the conditions of the environment. Hazy areas are the ones where nets appear denser with this type of filter, so that even if faulty holes are recognized inside these areas from illumination threshold method, they can be considered as uncertain.

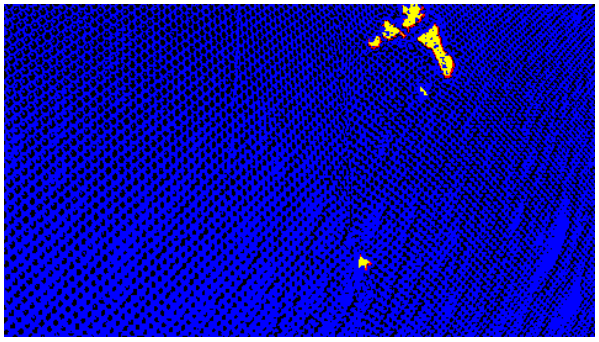
Another key point is that in the top borders of an image, the light might be more reflective on

the nets than in its center. Small window sizes fail to separate reflected light from nets. However, to deal with this manner, selection of larger window sizes in adaptive threshold can be applied or the observer can simply move the camera higher, so that the top parts can be transferred in the middle of the image where they are expected to be more recognizable.

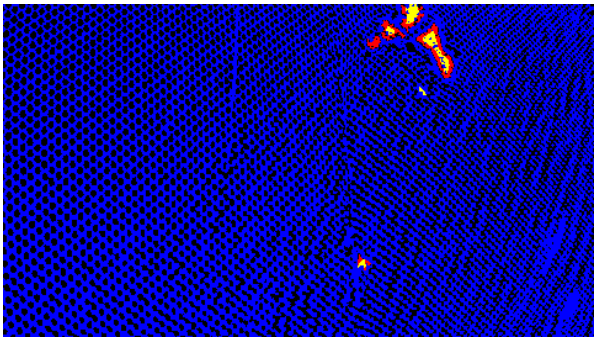
Here are the results for different offset (C) and mean filter size values. Blue shows the nets and red the holes.

Figure 3.6: final results for different window sizes of mean filtering
offset = 0.05

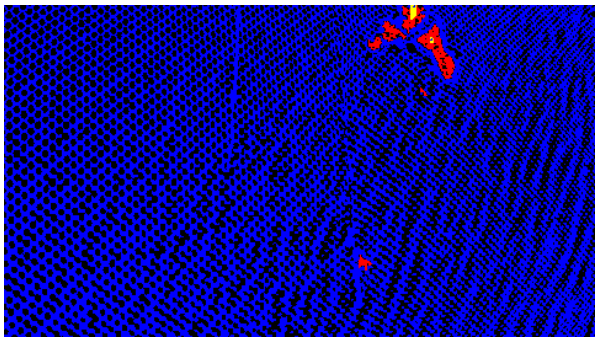
*window size = 4*



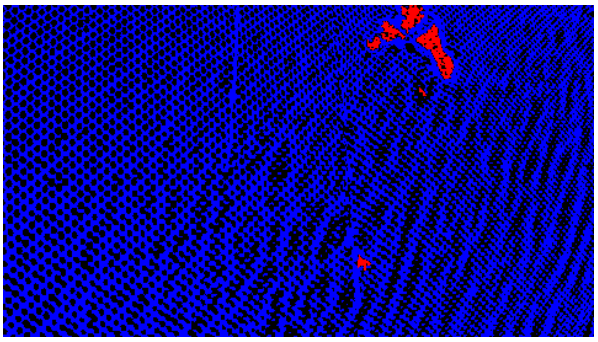
*window size = 9*



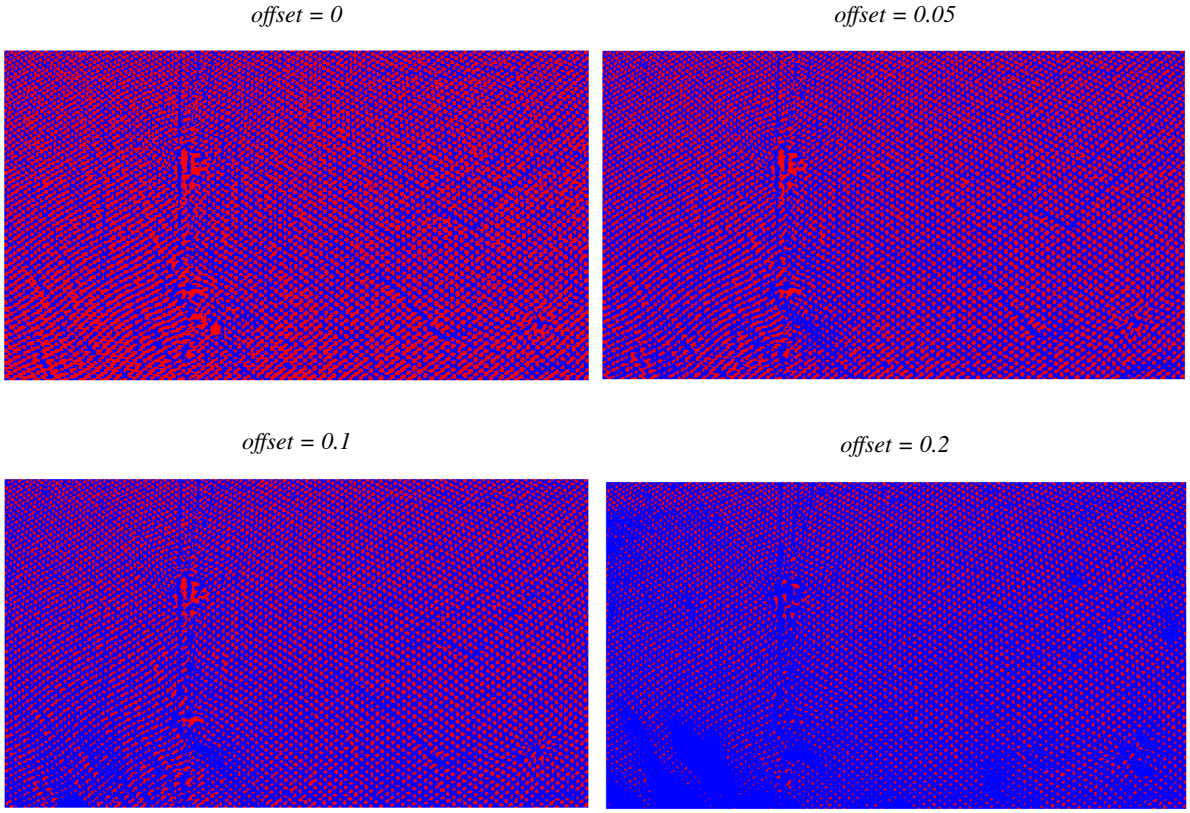
*window size = 15*



*window size = 20*



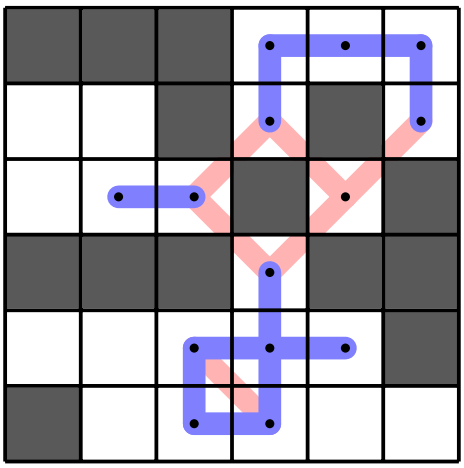
Images of Adaptive_Threshold for different offset values of mean filtering window size = 9




### 3.2.5 Connected Component Labeling

Once nets and holes components have been formed, connected component labeling algorithm takes place to specify a label for each hole. A hole is defined as a part of background that is completely enclosed by nets. This is the rule that separates holes from each other, and guides the algorithm to assign a unique label to each one of them. In MATLAB, this is *bwlabel function and it is applied in illumination image.* After some tests, results clearly favor, *connectivity 4*, to be the optimal solution in labeling holes, as it does not allow them to surpass lines of nets that exist between them and be considered as one object, increasing its size. However, this function considers white pixels as the candidate objects to receive labels. But, holes in illumination image are black pixels, because, until now, they represented the background area. So, a new image, Holes, receives the complement of illumination image and, afterwards, is used in 'bwlabel' algorithm.

Figure 3.7: CCL's possible (blue), and impossible (red) transitions with connectivity 4



**Algorithm 1**: Connected Component Labelling, Connectivity 4

**Data**: Labeling a particular pixel $(x, y)$ in illumination_image

**if** *the pixel (x,y) has '0'* **then**
  Do nothing and proceed to the next pixel (x+1,y)

**else if** *the pixel (x-1,y-1) has a label* **then**
  Assign the label to pixel (x,y)

**else if** *neither pixels (x-1,y) OR (x,y-1) is not labelled* **then**
  Increment label numbering and assign the latest label to pixel (x,y)

**else if** *pixels* $(x-1, y)$XOR$(x, y-1)$ *is labelled* **then**
  Assign the label to pixel $(x, y)$

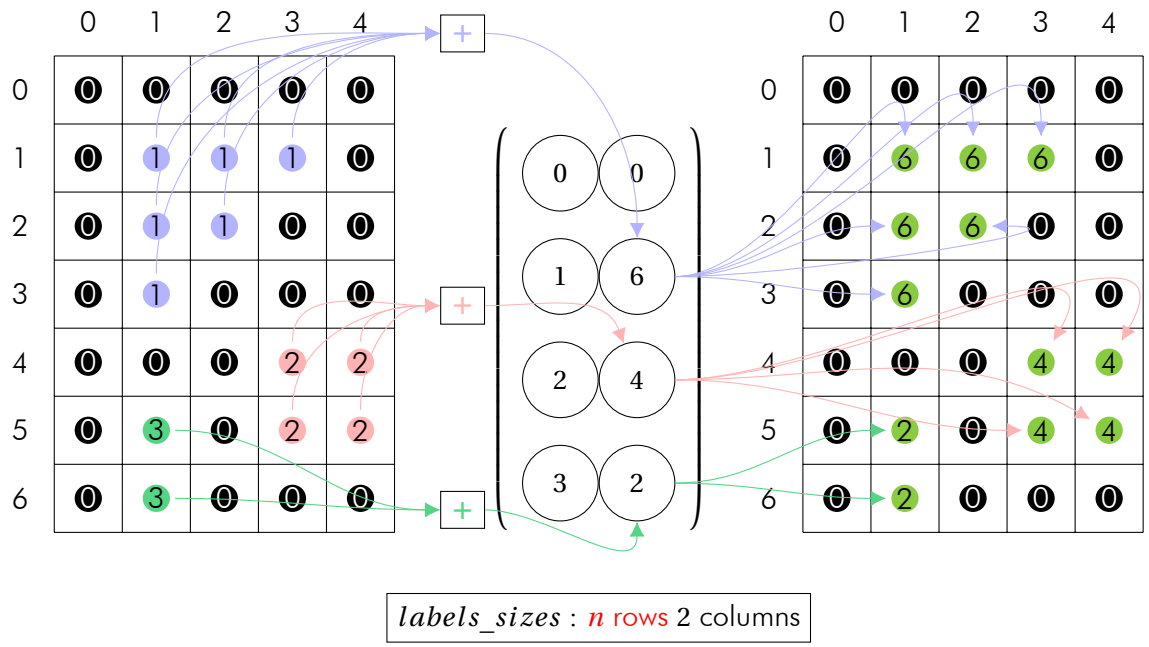
**else if** *both pixels* $(x-1, y)$AND$(x, y-1)$ *are labelled* **then**
  **if** *the label of pixel* $(x-1, y)$ *is greater than the label of pixel* $(x, y-1)$ **then**
    Assign the label of pixel $(x-1, y)$ to pixel $(x, y)$
    Record an equivalence for the label of pixel $(x, y-1)$ with the label of pixel $(x-1, y)$.
  **else if** *the label of pixel* $(x, y-1)$ *is greater than the label of pixel* $(x-1, y)$ **then**
    Assign the label of pixel $(x, y-1)$ to pixel $(x, y)$
    Record an equivalence for the label of pixel $(x-1, y)$ with the label of pixel $(x, y-1)$.
  **else**
    Assign the label of pixel $(x, y-1)$ or $(x-1, y)$ to pixel $(x, y)$

### 3.2.6 Algorithm for detection of defective holes

In matlab code, 'L' is the image holding the labels from 'CCL' method. Another table, labels_sizes, with two columns is used. This table will help on determining whether in an area, one or more objects significantly differentiate in size from other objects in that area, according to some specifications. The first column contains *uniquely* (because at 99%, objects consist of more than one pixels which is equivalent to more than one pixels containing the same label), with MATLAB function, unique, the labels of image L. The second column contains their size, after their number is counted in image L.

A new image 'sizes' is, also, implemented and receives in each pixel, the size of the object that it belongs to, based on the matching in 'labels_sizes' table. Basically, positions that contain the same label in image 'L' are replaced with their label size.

Figure 3.8: Holes' size calculation




Finally, the algorithm of finding faulty holes is applied. In image, containing the sizes (*sizes*), *windows of fifty by fifty pixels (50 x 50)* are created.

A pointer, hypothetically i, starts from the beginning of labels_sizes. Now, it is all about the coefficients that are going to participate in the comparisons between values of labels_sizes, for the least possible appearance of false positive and false negative results. False positive term refers to the holes which are not deformed, but they are identified as faulty. False negative term refers to the holes that are deformed, but they are not identified as faulty.

Keeping in mind the previous statements, it is time to decide about the comparisons of sizes inside labels_sizes. The first thing that comes in mind is, which sizes are going to participate in a comparison. A few methods are studied, and their positive and negative characteristics are provided. An initial method is proposed first and is going to, progressively, become better. In order to compare the methods, a frame with clear holes is considered as the main base for examination, where differences can be explained.

The following methods require the calculation of median values. This is explained from the fact that median values, as the central sizes of holes inside the windows, are not affected from extreme values which provide big and sudden deviations. The contribution of median values indeed leads to safe conclusions, as it is proved in a later stage. Median values are calculated as follows.

Median values require the following procedure. Every size inside a window is added to the

initialized array, labels_sizes. However, these sizes are randomly placed and, also, may appear more than once. Matlab's built in function, *unique*, performs their sorting and duplicates removal. Also, zero - '0' values are manually excluded, because, they represent nets (they do not receive any label<=>size). Then, median values can be safely selected.

There are three different types of median values that are used. Global_median involves all the distinct values of input image. Window_median involves the distinct values of a window. Finally, local_median involves distinct values of a window between a start and an end position inside labels_sizes.

global_median = labels_sizes$\left[\frac{n}{2}\right]$, where n is the number of distinct values in input image.
window_median = labels_sizes$\left[\frac{n}{2}\right]$, where n is the number of distinct values in a window.
local_median = labels_sizes$\left[\frac{pos1+pos2}{2}\right]$, where pos1, pos2 represent positions in a window.

**Algorithm 2:** Windows Algorithm

**Data:** Array $\Longrightarrow$ $window\_sizes$

Assign median value of windows_sizes to $window\_median$

**while** *( (i < N) AND ( ( window_sizes[e]< window_median/2 ) OR (window_sizes[e]<=2) ) )* **do**
  i = i + 1
error = 0, start = i

**while** *(i < end) AND (error == 0)* **do**
  Assign median value between start and i pointers to $local\_median$
  **if** *( (window_sizes[e] > local_median * 2) OR (window_sizes[e] > global_median* 4) )* **then**
    pos=i, error=1
  **else if** *i < end-1* **then**
    **if** *applied method ** **then**
      pos=e+1, error=1
  i=i+1
**if** *error=1* **then**
  *j*=pos
  **while** *j<end* **do**
    Mark with '1', the positions inside the window, that have same value with window_sizes[*j*]

#### 3.2.6.1 First method

The first method acts like a chain of three elements. The sum of the first two is checked with the third element. In case sum is greater, means that the third element violates the normalized sequence of values, and it signals the start of faulty sizes, because, the sizes after it are, also, considered faulty.

Model for detection of defective size at position i.

$$* \left[ \text{window\_sizes(i)} > \text{window\_sizes(i-1)} + \text{window\_sizes(i-2)} \quad \text{for} \quad i \geq 3 \right]$$

However, in case that there is a significantly large hole, which may be the only one that exists inside a window then it is going to affect the final result. This can happen, because this will be the only value added to labels_sizes and will fail correct checking. Such values can be avoided through the calculation of global median. So, before the activation of windows, all values of matrix, sizes, have to be processed through function, unique, and give global median. Global median is going to be an extra validity measure, especially for the first two values of the array, which can not be included in the previous model, due to exceeding array boundaries.

Of course, attention is needed because wrong choice of global median's multiplier can misjudge normal values and increase false positive percentage, as the table below shows.

Figure 3.9: Holes' results with global median threshold

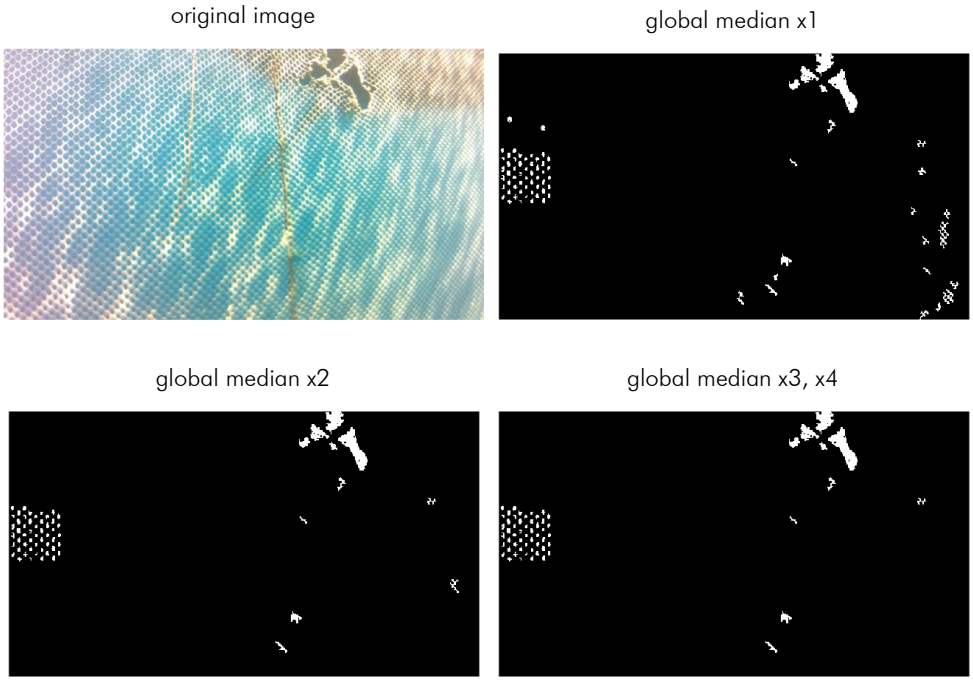



Table 3.5: Results with various global median values

| global median | false negative | false positive |
|---|---|---|
| x1 | ~ 1 | ~ 70 |
| x2 | ~ 1 | ~ 54 |
| x3, x4 | ~ 1 | ~ 53 |

Results remain the same after multiplication with three times the global median, so the final multiplier factor is three - '3'.

This method seems to identify quite accurately the existing holes. However, it is still vulnerable in false positive results. These errors happen because array, labels_sizes, still contains very small noisy values that come from small gaps between nets which are considered as holes. Actually, most of these sizes are composed from '1' pixel and increase the false positive percentage.

To cope with previous problem, pointer, i, has to ignore values less than a particular threshold. This is why median value, *window_median*, gets involved from the window that 'gathers' them for processing. Values have to be checked with a threshold which has direct relation with window_median. If they are less than this threshold, they have to be excluded/ignored. So, the procedure of finding the window median is done before any comparison and the best threshold is sought based on it.

Figure 3.10: Holes' results with global median (x3) and various window median thresholds

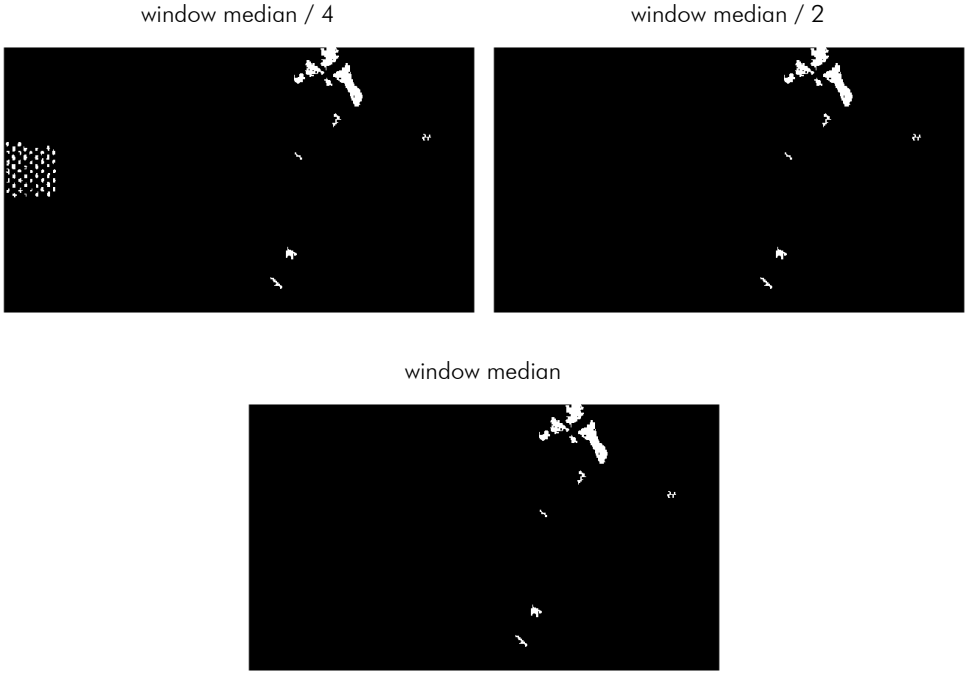



Table 3.6: Results with various window median values and 3x global median

| threshold | false negative | false positive |
|---|---|---|
| window_median / 4 | ~1 | ~ 53 |
| window_median / 2 | ~1 | 4 |
| window_median | ~1 | 4 |

### 3.2.6.2 Second method

The second method follows a different approach. Instead of manipulating three consecutive values, a local median is chosen compared to the value that is currently checked. After pointer, i, has avoided the noisy small sizes it stops at a value greater than window median. Another pointer,

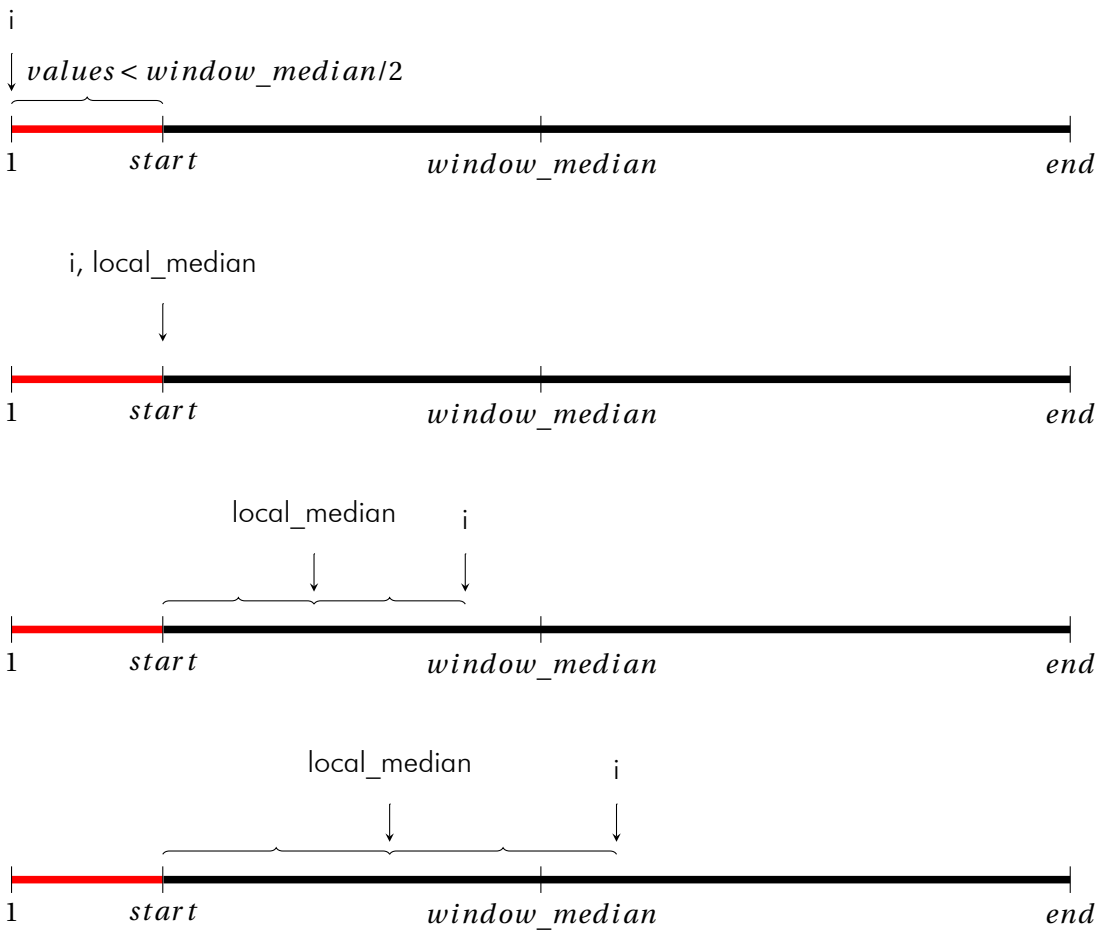
*start*, is used to address the first 'valid' element/size to be tested. Afterwards, the *local median* value is chosen between positions, start and i, while i is moving towards the end of array. Now, the sum of value in position i and local median, is checked with value at position, i+1. This time value in position, i+1, is the start of faulty sizes. The comparison with global median should be kept to indicate if the size at position, i, is faulty.

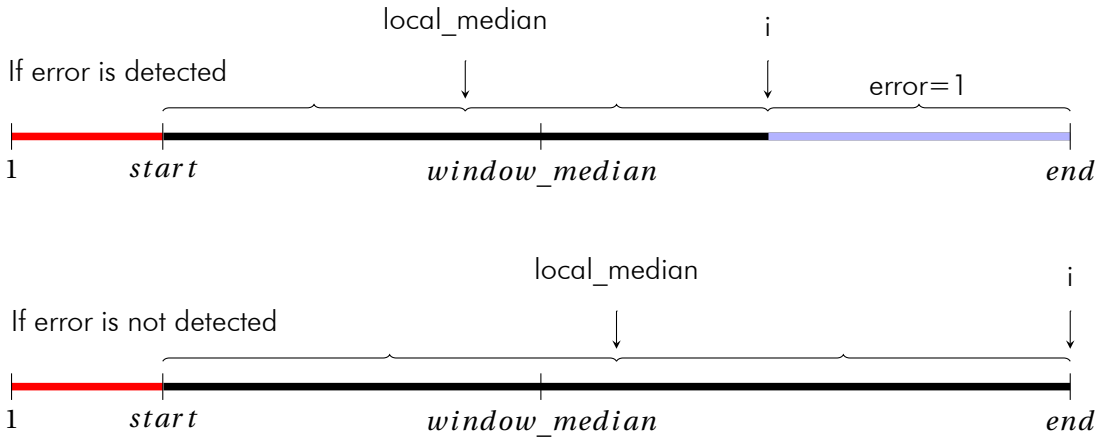
Model for detection of defective size at position i+1.

$$* \left[ \text{labels\_sizes[i]} + \text{local\_median} < \text{labels\_sizes[i+1]} \quad \text{for} \quad i \geq 1 \right]$$

The first faulty position is stored in a local variable, *pos*. In the end, values from position, pos, until the end of labels_sizes are marked as faulty. Pixels in image/array, sizes, that have the same value/size compared to faulty sizes, are highlighted and added to 'holes' image, which contains all the defective holes at the closure of the procedure.

Figure 3.11: Process for identification of labels inside labels_sizes array






### 3.2.7 Multi Frame Processing

Until now, results prove to be clearly effective for the user. Although, there is still room for improvement to maximize the chances of correctly identifying defective holes. Thus, because further analysis could not be applied in an individual frame, due to the lack of more researches/observations, the only chance of improving results is by combining sequential frames. As frames are passing through, the large and existent holes seem to remain marked in the same places with a bit of deviation, due to the movement of camera or nets. However, the wrongly identified holes of sequential frames appear in clearly different places (they are far apart from each other).

The final observation provides two occasions for a particular number of sequential frames. If a hole is detected in a frame, the corresponding one, in the next frame, is either detected or not. Frames appear with high ratio, so, even if camera moves too fast, the same area is going to appear in at least two consecutive frames, and same holes should continue appearing for an amount of consecutive frames.

If a hole is detected in consecutive frames, then it is going to be deformed, due to its deviation in the frames. Deviation depends from the axis to whom the camera is moving. Even if the camera is not moving, then the movement of nets will affect their appearance. The matter is, how many subsequent frames, compared to a starting frame, are able to have at least one same set of an existing hole's pixel coordinates, marked with value '1'. This will mark that neither haze nor nets can block the particular holes, providing a stronger statement for their existence.

To clarify the continuously detected holes in consecutive frames, pixel to pixel multiplications take place between matrices, which contain their holes, and results are stored in a new matrix. Pixels that remain at value '1', after multiplication, are used to represent the final faulty holes for all these frames. Examples for different number of sequential frames with their pixel to pixel multiplications are presented below :

Figure 3.12: Holes' results from two consecutive frames (a) and three consecutive frames (b)

(a) (b)

These figures prove that the multiplications overlay with '0s' some pixels of the detected holes in the combined frames and they will not be represented entirely, due to the deviation. So, after multiplications take place, they produce a final matrix with some pixels that remain at value '1' (faulty holes). These are used for the expansion of the hole that each one belongs to, based on the formation that holes have in the last combined frame. Specifically, labels' matrix, L, of the last combined frame is used to retrieve the labels of the pixels that have the same coordinates with the pixels that remain at '1' in the matrix produced from multiplications of consecutive frames. These labels will be used to present all the pixels that belong to them in the last combined frame.

Figure 3.13: Holes' extension from two consecutive frames (a) and three consecutive frames (b)

(a) (b)

| (a) | | (b) | |
|---|---|---|---|
| false negative | false positive | false negative | false positive |
| 1 | 2 | 7 | 0 |

Tests show that only two consecutive frames satisfy better the main goal. Basically, three or more consecutive frames do not identify any holes at all, due to high deviation. ***Having a low false positive percentage, which means that only a few non existent holes may appear in a frame and possibly are going to disappear in the next frame, is better than a high false negative percentage which means that even the obvious holes will not appear.***

## 3.3 Comparison with previous thesis

The main goal of this section is the reference of improvements that are made compared to Badogiannis' thesis, by presentation of same results side by side. In every figure section 'a' refers to original image, section 'b' to Badogiannis' results and section 'c' to current results.

First of all, two simple examples have to be shown in order to confirm the basic functionality of this project. The images below show a balanced situation where there are no deformities, and illumination is also the same in whole image.

Figure 3.14: Image without holes

(a) (b) (c)

| (b) | | (c) | |
|---|---|---|---|
| false negative | false positive | false negative | false positive |
| 0 | 0 | 0 | 0 |

Next is an image with differentiation in size of holes. It is clear that, in such circumstances, the current model behaves completely accurately, just like the previous one, The user must keep in mind and select correctly the two variables that determine the final result. These are illumination threshold (holes) and offset value for mean filtering (nets) which are responsible for monitoring correctly nets and holes.

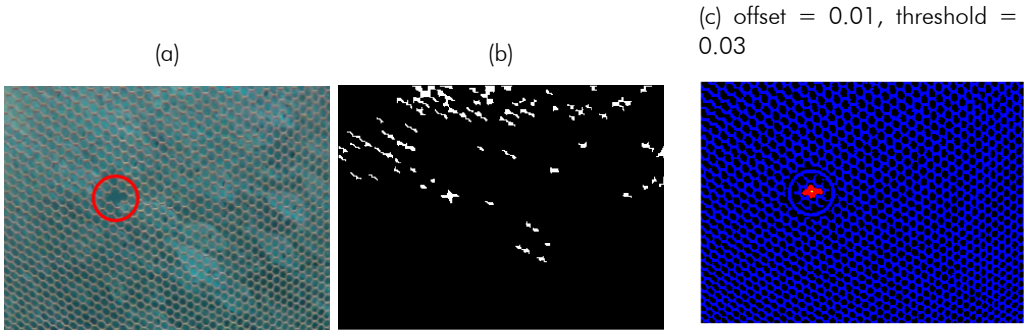
Figure 3.15: Normal image

(a) (b) (c) offset = 0.01, threshold = 0.3

| (b) | | (c) | |
|---|---|---|---|
| false negative | false positive | false negative | false positive |
| 0 | 0 | 0 | 0 |

A real time image with haze, brightness variation and different hole sizes is examined. Previous results show that nets can not be separated from holes, where haze exists (here haze attends fish). As a result, false positive percentage grows significantly. Instead, the current method is able to eliminate these differences, especially through image smoothing.

Figure 3.16: Real time image with fish




| (b) | | (c) | |
|---|---|---|---|
| false negative | false positive | false negative | false positive |
| 0 | ~ 64 | 0 | 0 |

The selection of offset and threshold in figure 3.16 (c) is based on the following observations:

Figure 3.17 shows results, exclusively produced from illumination threshold.

Figure 3.17: Illumination variance for figure 3.16 (a)

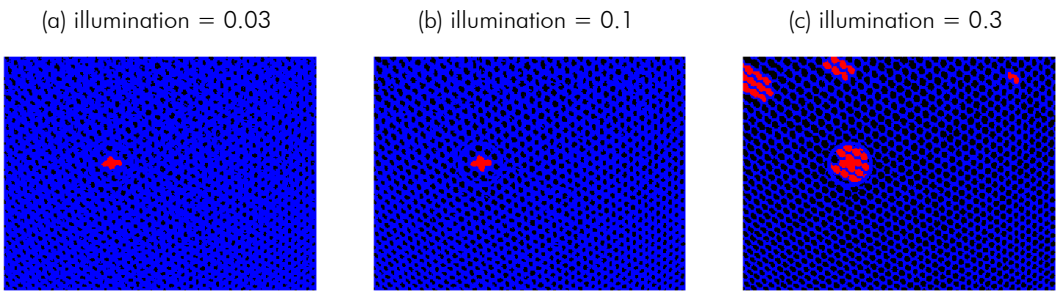
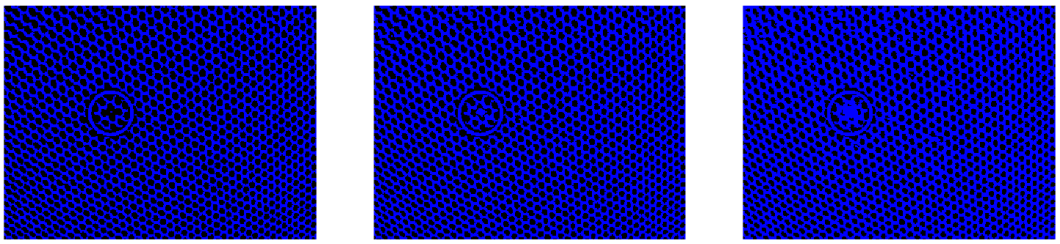



Figure 3.18 shows results, exclusively produced from Adaptive_Threshold.

Figure 3.18: offset variance for figure 3.16 (a)

(a) offset = 0.01 (b) offset = 0.05 (c) offset = 0.1



Considering the above, results in figure 3.16 (c) are the combination of figures 3.17 (a) and 3.18 (a).

Figure 3.19: Image with nets deformation

(a) (b) (c) offset = 0.01, threshold = 0.3

| (b) | | (c) | |
|---|---|---|---|
| false negative | false positive | false negative | false positive |
| 0 | 29 | 0 | 0 |

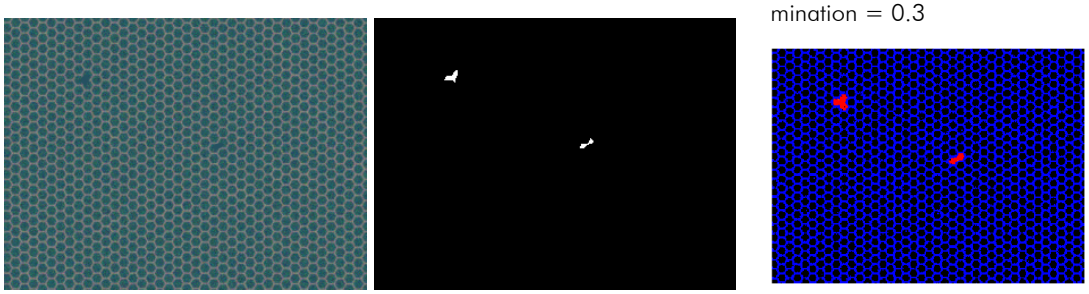
Figure 3.20: Normal image

(a) (b) (c) net's separator = 0.01, illumination = 0.3

| (b) | | (c) | |
|---|---|---|---|
| false negative | false positive | false negative | false positive |
| 0 | 0 | 0 | 0 |

Figure 3.21: Image with noise

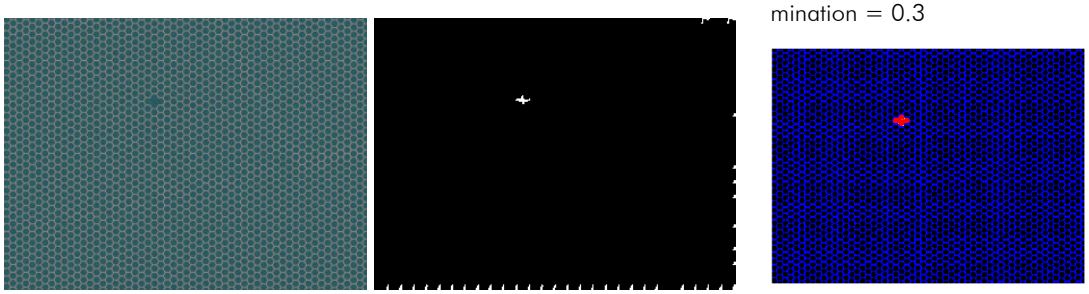
(a) (b) (c) net's threshold = 0.01, illumination = 0.3



| (b) | | (c) | |
|---|---|---|---|
| false negative | false positive | false negative | false positive |
| 0 | 0 | 0 | 0 |

Figure 3.22: Image from high distance

(a) (b) (c) nets separator = 0.01, illumination = 0.3



| (b) | | (c) | |
|---|---|---|---|
| false negative | false positive | false negative | false positive |
| 0 | 36 | 0 | 0 |

In summary, results show that the new method avoids the problems at the edges, recognizes more entirely the holes, and it is resistant in deformation of nets, change of illumination, and different distance between camera and nets.

# Chapter 4

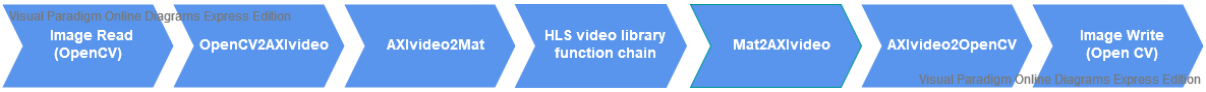
# Hardware Design

## 4.1 Tools

The overall design is implemented with *Xilinx Vivado High Level Synthesis tool (HLS: version 2019.1)*. The programming language that is used is C++, since it allows the implementation of more complex and faster processing algorithms. Each function is translated to a *Register Transfer Level* block and is designed for a specific purpose. There are cases where a function needs to be divided into simpler functions, in order to make the tool understand better, the *Hardware Description Language* conversions, and apply the corresponding transformations.

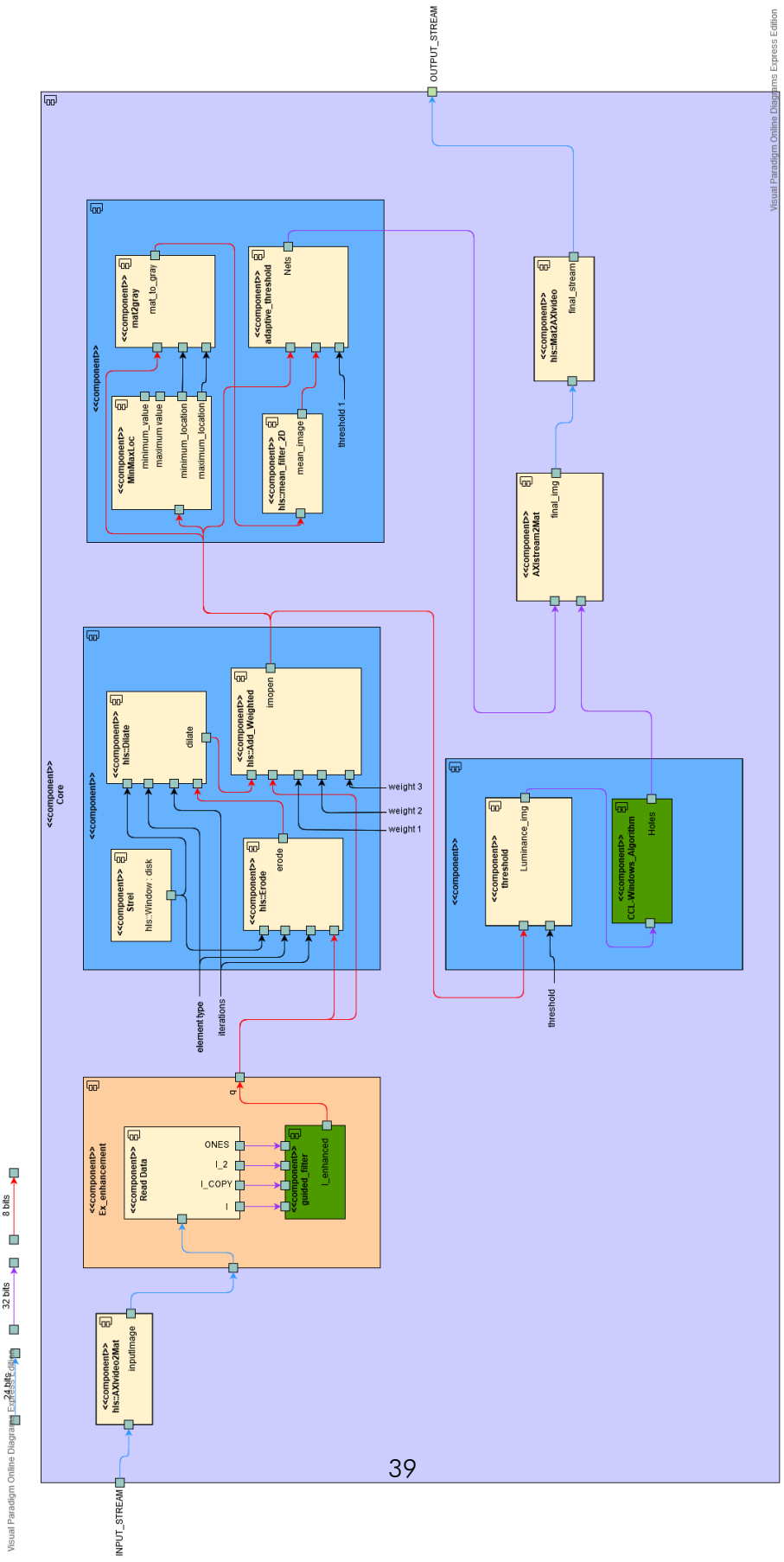
Initially, the block which contains the corresponding MATLAB functions, has to be implemented as software functions in HLS in order to verify its correct operation, before applying RTL conversions and attaching them to the final system. The overall execution on the PC is explained in depth. Instead of MATLAB's ability to expand an input video to frames, HLS uses OpenCV functions in test bench files to read/write frames from/to PC. As a result, frames are exported from MATLAB to PC, and then they are read from a test bench in Vivado HLS. Then, with another function of OpenCV, each frame is transformed into a stream of pixels and is passed to the block. In the same way, when the block finishes computations, the sequence of pixels returns to test bench and is reversed to an image. Results of such images are shown afterwards, and are indeed quite similar with the corresponding ones in MATLAB.

Figure 4.1: Vivado's sequence of computations for video input



## 4.2 Top Level Design

Figure 4.2: Top level design



*Every table is extracted from the synthesis of the hls component, so all the numbers are an estimate for a final FPGA design (ZCU102 is selected in this case, as shown in 'performance evaluation' chapter).*

Table 4.1: Latency and Interval estimates in clock cycles for top level functions

| Module | Latency | | Interval | |
|---|---|---|---|---|
| | min | max | min | max |
| AXIvideo2Mat | 130,953 | 130,953 | 130,953 | 130,953 |
| ex_enhancement | 137,457 | 137,457 | 137,285 | 137,285 |
| AddWeighted | 1 | 130,681 | 1 | 130,681 |
| MinMaxLoc | 130,681 | 130,681 | 130,681 | 130,681 |
| mat2gray | 129,658 | 129,658 | 129,658 | 129,658 |
| mean_filter_2D | 134,536 | 134,536 | 134,536 | 134,536 |
| adaptive_threshold | 129,606 | 129,606 | 129,606 | 129,606 |
| threshold | 129,603 | 129,603 | 129,603 | 129,603 |
| CCL_WindowsAlgorithm | 1,319,968 | 1,581,346 | 1,319,968 | 1,581,346 |
| AXIstream2Mat | 129,603 | 129,603 | 129,603 | 129,603 |
| Mat2AXIvideo | 130,681 | 130,681 | 130,681 | 130,681 |

Table 4.2: Resources usage of top level functions

| Module | BRAM18K | DSP48E | FF | LUT | URAM |
|---|---|---|---|---|---|
| AXIvideo2Mat | 0 | 0 | 258 | 478 | 0 |
| ex_enhancement | 174 | 16 | 32,669 | 37,293 | 0 |
| AddWeighted | 0 | 0 | 187 | 345 | 0 |
| MinMaxLoc | 0 | 0 | 85 | 245 | 0 |
| mat2gray | 0 | 2 | 7,370 | 4,471 | 0 |
| mean_filter_2D | 8 | 0 | 2,509 | 5,376 | 0 |
| adaptive_threshold | 0 | 8 | 381 | 655 | 0 |
| threshold | 0 | 0 | 28 | 160 | 0 |
| CCL_WindowsAlgorithm | 915 | 6 | 126,083 | 99,517 | 0 |
| AXIstream2Mat | 0 | 0 | 29 | 211 | 0 |
| Mat2AXIvideo | 0 | 0 | 115 | 365 | 0 |

The top level design is represented in figure 4.2 which represents a pixel streaming design. Specifically, FPGA receives an input image as a stream of pixels. The main function, Net Holes Detection, consists of substitute functions in a *dataflow manner*. The last statement means that after the conversion of the initial image to a stream, this stream continues to flow through the next functions via other streams. When a function/component is ready to receive a pixel, from the stream connected to its start, just waits for the stream to become non empty, and then accepts it. After the

computations finish, the pixel is sent to the stream connected to function's end. A function is either a variety of other functions, a single function, or a Vivado built in function which is ready for use.

Every arrow represents a stream. Every stream is implemented as a FIFO to connect components. Each FIFO has depth of one or more places, depending on the number of clock cycles it has to wait before sending its data to receptor function. There are various types of FIFOs. FIFOs with blue color refer to 24 bits wide pixels. This is because they consist of the three image channels (R, G, B) and each one has 8 bits. FIFO's with purple color have places for 32 bit wide pixels that correspond to images of one channel, and serve the purpose of calculating more precisely the internal mathematical expressions. FIFO's with red color are necessary for the use of built in functions that only require as input, images with one channel and width of pixels equal to 8 bits, such as the 2D mean filtering. Purple FIFO's receive pixels of 8 bits that correspond to streams (images) of one channel. Finally, black arrows express signals of various types.

The type of data processing is referred with a specific color.

- Yellow color means that when a pixel is received by a function, it gets processed, and is sent to the next function(s). In the meantime, while this pixel gets processed, more pixels can be inserted and receive the same treatment. With the use of pipeline instruction the overall procedure delays only for the number of cycles the first pixel needs to be extracted.
- Orange or green color means that the function needs more than one pixels to arrive from the previous function to produce a single pixel for the next function(s) (there are two colors used because if a subfunction is of that type then its parental one is too). This occurs, because, there is a need for random access to different pixels, so that the current pixel can be produced correctly. Once the computations finish for a pixel, which is stored in a Block RAM (BRAM) or a Look Up Table RAM (LUTRAM) for the purpose of random access, it is sent to the FIFO(s) that waits for it. So, the overall latency is increased by the number of clock cycles that are needed to gather the appropriate pixels from the input FIFO(s) plus the combinations and computations that take place inside the function.

Every component corresponds to a function. with its inputs and outputs. Functions that have substitute functions form dataflow regions for them and control the way they are connected in hardware for the data to pass through. Functions that have blue colour are not implemented and are just represented from their internal functions which interact inside the main dataflow function, Core, in order to take full advantage of dataflow parallelism.

Explanation of functions.

- AXIvideo2Mat converts OpenCL input to a stream of pixels.
- Ex_enhancement is dataflow beetween read data and guided filter, and serves the role of corresponding MATLAB function. The exact flow of information inside guided filter is shown below.

Figure 4.3: Internal connections of Guided filter

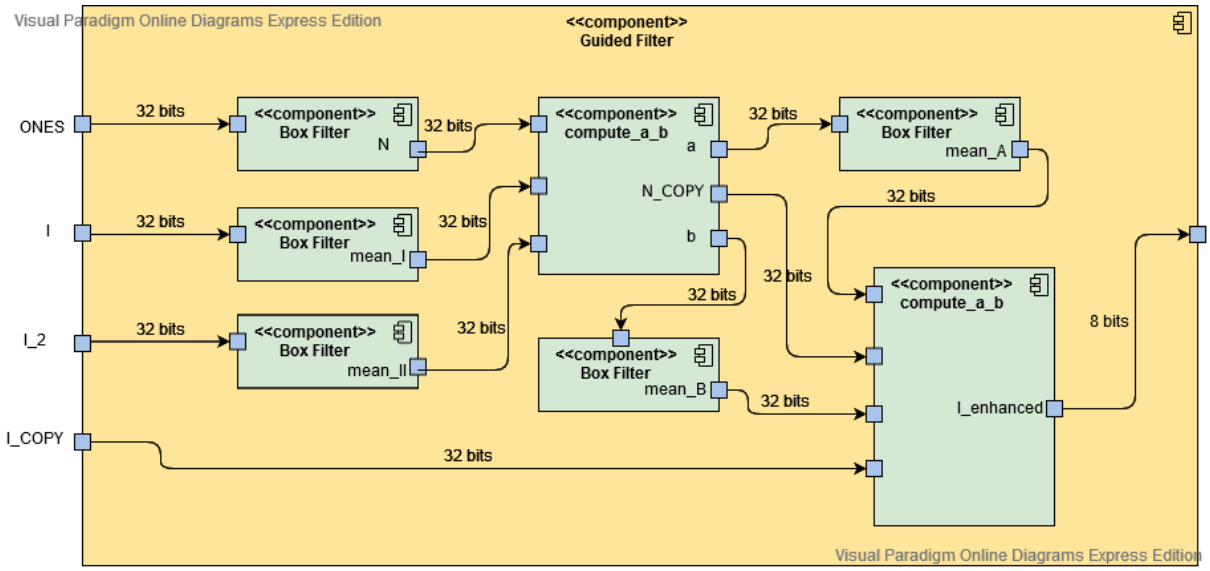



Table 4.3: Latency and Interval estimates of Guided filter's substitute functions

| Module | Latency | | Interval | |
|---|---|---|---|---|
| | min | max | min | max |
| compute_a_b | 129,710 | 129,710 | 129,710 | 129,710 |
| compute_I_enhanced | 129,657 | 129,657 | 129,657 | 129,657 |
| Box_Filter | 137284 | 137284 | 137284 | 137284 |

Table 4.4: Resources used from Guided filter's substitute functions

| Module | BRAM18K | DSP48E | FF | LUT | URAM |
|---|---|---|---|---|---|
| Box_Filter | 34 | 0 | 1212 | 1310 | 0 |
| compute_I_enhanced | 0 | 4 | 10,500 | 8315 | 0 |
| compute_a_b | 4 | 8 | 15,794 | 12,206 | 0 |

- Read Data prepares the streams for guided filter.
- Guided filter is the corresponding function in MATLAB. It performs dataflow between its substitute functions as figure 4.3 shows. Computations on substitute functions compute_a_b and compute_i_enhanced take 129,600 clock cycles to edit their input streams which is equal to the actual size of the input image in pixels (270 x 480), and with pipeline assistance they increase the overall latency just for some cycles. However, box filter delays for another 16 (radius which is translated to lines of image) x 480 (columns : width of image in pixels) = 7,680 more cycles. So, the latency of the whole project increases from 129,600 to 137,280

and is the minimum for these functions. Box filter actually gathers pixels in packs of thirty three (33) lines, using a line buffer. It performs the computations of images 2.2 - 2.5 in chapter two. Height of buffer is two times (2x) the radius of guided filter, while width remains the same with input image. This structure is provided, because, every pixel that is inserted at each clock cycle needs in worst case scenario, a pixel that was inserted two times the radius clock cycles before. This buffer saves a lot of space in hardware resources, either this is BRAMs or LUTs, as its height depends from radius of filter rather than the actual height of image.

The task of these functions is compared to MATLAB model. Compute_a_b computes these two values, as its name reveals. There is a slight difference in computation of a. This value needs the calculation of covariance between streams mean_I and mean_p in each local patch, and variance of mean_I. However, when I is the same as p, it can be seen from MATLAB that these values are exactly the same thing. So, only the variance of mean_I is computed in the end. Compute_I_enhanced computes the corresponding 'q' value in MATLAB. At the very end there is a transformation that has to be mentioned. Because the next functions that receive Compute_I_enhanced result are the built in functions erode, dilate, and AddWeighted. However, these functions accept 8 bit inputs and Compute_I_enhanced produces 32 bit outputs. It was observed that outputs vary inside the range [-3, 3]. So, if they are increased by three they will surely be positive and then they are multiplied by 10 to be spread out better, never exceeding value 255 which is the end of 8 bit signals. Then they are ready to continue.

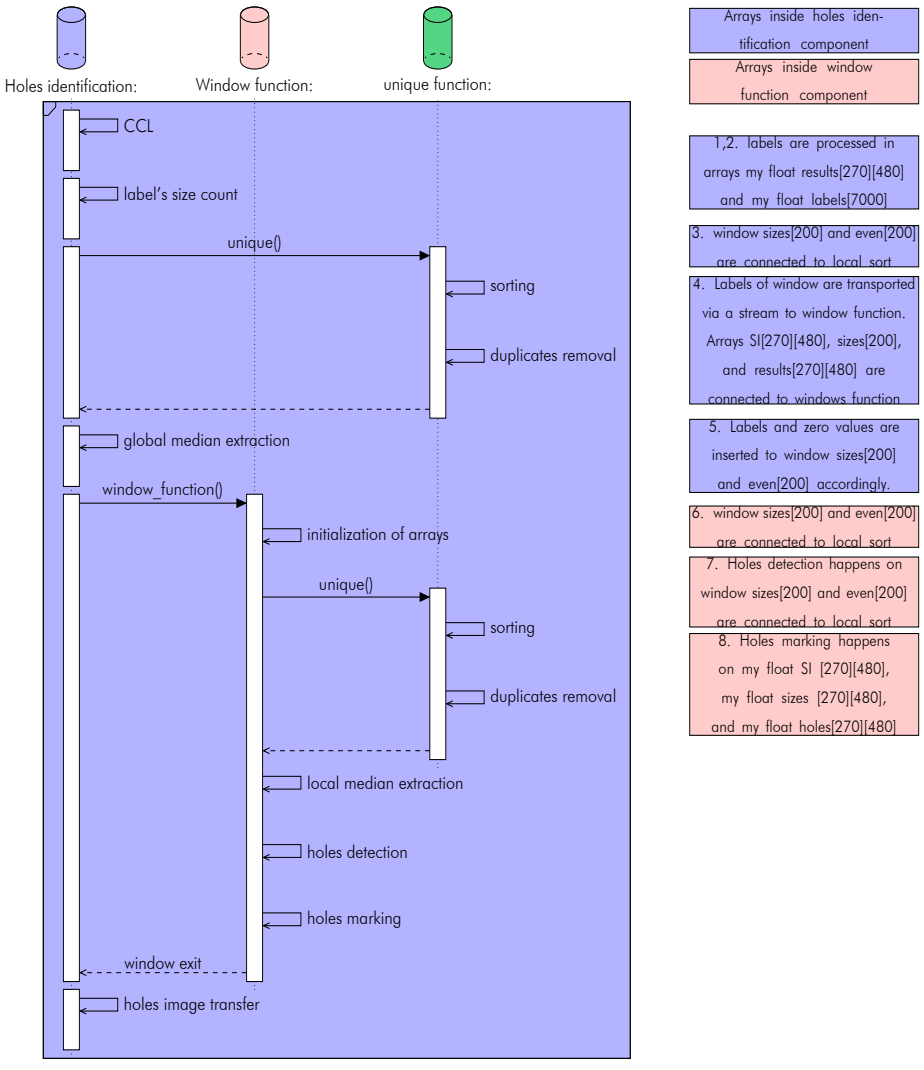
- Strel is a shortcut name for structuring element and passes values in disk which is implemented as a built in HLS hardware structure of type, *window*. Structure, window, is a 2D dimensional buffer.
- Erode and Dilate are Vivado's built in functions which perform morphological opening of the image. They receive as inputs integers, *element type* and the *number of iterations*, which are equal to zero/square (0) and one (1) respectively. Both functions receive as input, the window from function, Strel.
- Add Weighted is a simple linear transformation for subtraction of morphologically opened image from enhanced image. Its model is:
- Min_Max_Loc finds the minimum, and maximum value in entire input image and locates their position. Here, only the values are in central interest.
- Mat_to_Gray converts input image to an intensity image that contains values in the range zero (black) to one (white). Minimum and Maximum are the values supplied by MinMaxLoc that correspond to values, zero and one, in intensity image. Values less than minimum become zero, and values greater than maximum become one.
- Mean_Filter_2D is a Vivado built in function which performs 2D convolution on image.
- Adaptive threshold is the final step for Nets visualization. It performs the following subtractions:

$$Nets = Im1 - Im2 - C,$$

where Nets is the image containing the nets, Im1 is the image of mean filter, Im2 is the Foreground image, and C is the offset value which actually shifts the values of mean filtering in order to distinguish where pixels belong (either nets or holes). Pixels which are going to end up positive or zero are nets, while negative values will just be ignored (they do not represent holes, because holes exist in the image produced in simple threshold).

- Threshold performs the binarization of smoothed image with the involution of illumination threshold value this time.
- Connected Component Labeling - Windows Algorithm performs these two operations on the binarized image from illumination threshold.

Figure 4.4: CCL - WindowsAlgorithm sequence diagram and resources usage




Connected Component algorithm is based on method of two passes over the image, which is explained in chapter three. To achieve this method matrices, *results and labels_sizes*, are used.

Matrix, results, has the same size with initial image and holds the labels of pixels after the two passes. Labels_sizes serves its first designation on this stage of computations. It is used for storage of all the different labels that have been assigned to the entire image, results, after the first pass. It also indicates if two or more different labels should be merged, which is equal to connecting two smaller objects, that are connected together, and producing a larger united object. In second pass, entire image, results, gets passed again to receive the final label value on all its pixels, which may have changed due to the previous conflict phenomenon.

Subsequently, matrix, labels_sizes, is transformed into its next role, which is storing the sizes of objects. Therefore, labels_sizes has to reset to zero. Another pass follows on results, where each pixel label is translated to an index in labels_sizes. The value (size) of an index is incremented by one, every time this index is found during the last pass from results. Using the same matrix, labels_sizes, for storing both labels and then sizes values, saves space in harware resources usage. The size of labels_sizes was chosen to be seven thousand places (7,000), and depended from the maximum number of different labels that could be produced in MATLAB simulation.

Another pass on image, results, takes place for the production of a new image, *SI* (Sizes Image), that contains the size of objects. SI's pixels value is based on their labels. Pixels that belong to the same object receive the size of this object from labels_sizes matrix. The reason for using a new matrix and not reusing matrix, results, by overwriting its values is because of the restrictions of pipelining that cost in clock cycles. Specifically, a value/label of a pixel in results would need to be read, in order to retreive pixel's size from labels_sizes, and, concurrently, be written at the same cycle, so as the retrieved value/size would overwrite the previous value/label on this pixel. The last case forces pipeline to use two clock cycles instead of one to refresh the value of every single pixel on the same matrix. Because, this latency is equivalent with two passes of one clock cycle each on matrix, results, a new matrix in the end is used. The trade off is the usage of more hardware resources compared to faster design plus the fact that labels values of results will be needed afterwards.

First step is calculation of median value, *global median*, on entire SI image. For the purpose of comparisons between sizes, one dimensional array, *labels_sizes*, that was mentioned in chapter two is used. It is obvious that passing all the values would result in an array of same size with initial 2D image, but in one dimension this time. Talking with numbers, this is equal to multiplication between height and width, which is 470 x 280 = 129,600 places. However, utilization of such resources can be avoided. A large volume of pixels is limited to values until five, especially, zero and one. Zero values should be avoided anyway, because, they represent nets and are excluded from formation of objects (imagine that nets commit a large percentage of image). So, positions zero to five of window_sizes are marked With their index values to represent these sizes. Whole array, SI, is then passed and values, that are greater than five, are assigned to next positions of window_sizes. This trick reduced, dramatically, the number of sizes inserted to window_sizes, to around fifty (50) per image. Fifty plus five (0-4) makes only 55 places compared to 129,600.

Values of labels_sizes are randomly placed and appear duplicated from previous step.

As a result, sorting of values and deletion of duplicates is necessary for extraction of correct median value. Hardware component, *Unique*, implements both requirements. Even-Odd type of sorting is selected with maximum number of iterations, half the size of the connected array to the component. Another array, *even*, is used with the same labels_sizes length. Duplicates removal follows with one pass along labels_sizes.

Windows Algorithm begins here. Access is the same with MATLAB, where areas of 50 by 50 pixels are processed in a sequential manner. This strategy favors resource savings rather than time consumption. After all, the part that each window has to pass through is fast enough to be considered in final design.

Windows of 50 x 50 = 2,500 values have to pass into labels_sizes, to extract value, *local median*, after sorting. This is, also, a large value to determine the actual size of this array. Previously labels_sizes was used for storing around 60 values for whole image and now a smaller part of it would require 2,500 values. But, labeling procedure generated labels based on the west and north neighbors of a pixel. So, only pixels with different values from both their neighbors will enter labels_sizes, because, this means that these pixels failed to participate in the same object of at least one of their particular two neighbors. Variety of sizes fell at around 180 different values. The overall length of labels_sizes was finalized to 200. These decisions have the following consequences:

- Sorting, with pipeline usage, reaches maximum latency of 100 clock cycles.
- Duplicates removal requires a full pass along labels_sizes, of 200 cycles.

The rest of windows algorithm is similar to MATLAB's version, with some changes in the way final holes are produced, and is going to be explained at the differences between two models in next section.

- AXIstream2Mat merges 'Holes' image with 'Nets' image on a new image. Overlapping pixels, which provide the regions of uncertainty, are marked with yellow color. Clear holes are shown with red color. Finally, nets are shown with blue color.
- Mat2AXIvideo performs the inverse task from AXIvideo2Mat which is transforming the stream of pixels into an actual image.

## 4.3 Comparison with MATLAB model

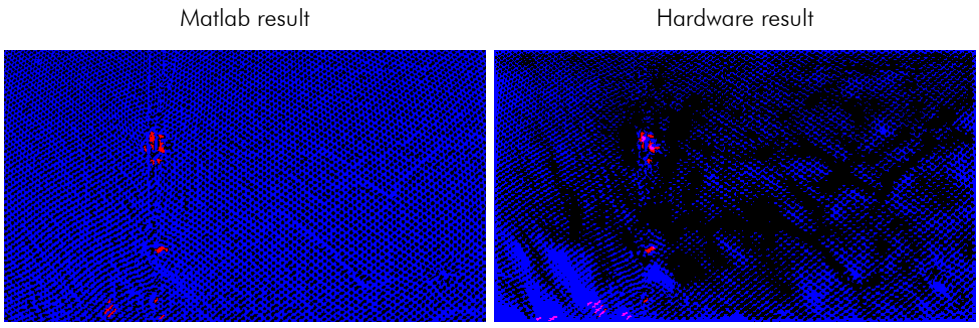
Matlab applies operations by transforming values into high precision floating points which means that they are more time consuming and expensive in use of resources, due to the way that they are defined and being processed. In hardware variables were chosen to have type 'fixed point' with 32 bits width, 16 of which represent decimal places, and the other 16 fraction. This kind of types are used in hardware, because they can be defined to be as precise as the user (based on his requirements) wants, and facilitate the accomplishment of calculations in less clock cycles and smaller use of resources. This way the variables get rid of many bits that represent fraction places, compared to matlab, without sacrificing the accuracy of the final results.

In function 'windows' threads access in groups as many values on array 'SI' (Sizes Image) as their number is. When faulty values are found in array 'window_sizes' for a processed window, a copy 'SI_2' of array 'SI' is initialized with zeros and is updated with the values/sizes in the corresponding positions of the window on SI. Each faulty size is checked with sizes on SI_2. Any equation transforms the corresponding SI_2 pixels to '1', while the rest remain at value '0'. The reason why a new array is used instead of SI is that SI may contain same values inside and outside the currently processed window and the latter should not receive any change at this stage at least, because they do not belong at the currently processed window. In the end, with the participation of all windows, all faulty holes will be encountered. However, windows procedure is too slow with this implementation of matlab. Unrolling the comparisons of each faulty size with all the sizes inside a window, could not allow synthesis to finish placement and routing, due to the large multiplexers that had to be implemented. Another approach is used. Array 'holes' is initialised with zeros and passed to 'windows' component (from CCL-Windows algorithm component). Two divisions take place between the width of matrix 'holes' and each faulty size. The quotient of one division is used as the row pointer in 'holes', while the remainder of the other division is used as the column pointer. The combination of these pointers is used as the final, yet exclusive-unique position of a size to be marked with '1' inside 'holes' matrix. Objects from size 1 until 129,600 can be stored in this array, so even if whole image is a faulty hole, still manages to fit in. So, every size inside a window will know whether it will be '0' or '1' in the final image due to its position inside 'holes'.

Hardware model does not include the final, but simple step of combining two sequential frames. This step is not included, because the system was not finalised as a Vivado project. It performs some computations pixel by pixel between two sequential frames. There are ready systems from Vivado itself that can separate sequential frames of the input video to make this step quite simple.

Output of function ex_enhancement, responsible for image smoothing, gets treated in a different way in hardware model. In hardware, in order to take advantage of the built in functions erode and dilate that follow after ex_enhancement, a transformation of its pixels width from 32 to 8 bits takes place. This happens, because these built in functions receive as input matrices with 8 bit wide values. This is the point where results in hardware become less accurate, but only in the appliance of mean filtering for the calculation of nets.

Figure 4.5: Matlab vs Hardware



# Chapter 5

# Performance Evaluation

## 5.1 Run Times in Matlab

MATLAB run times were tested on a computer with 4 threads overall. This means that some MATLAB calls can achieve parallelism in processes of arrays. So, any recorded time does not refer to a, completely, sequential program, but it is supported from threads. Each time represents the overall latency of a single frame (entry 1 is for frame 1, entry 2 is for frame 2...), from the time it enters, image enhancement, function until the time it exits, holes identification, function. The recorded values are shown in table 5.1.

Table 5.1: Matlab execution times in ms.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 863 | 180 | 165 | 97 | 149 | 124 | 73 | 76 | 109 | 82 | 70 | 99 | 82 | 74 | 98 |

These values are the execution times for the whole algorithm in MATLAB, except from the part of multiframe processing, because, it was omitted in hardware design. This part is quite easy to be designed, as it only needs some subtractions and multiplications between two successive frames, in case this method is chosen in a true embedded system, even though this design can support an embedded system itself.

## 5.2 Execution Time in FPGA

The clock cycle period is set to a target of $4_{ns}$ (ns: nano seconds). Target is the desirable clock period that should not be exceeded by any of the designed components. This requirement is accepted from the system, to run as expected, as it automatically sets the actual period at $3.5_{ns}$ with $0.5_{ns}$ uncertainty. Uncertainty implies the deviation of the actual arrival time of the clock edge with respect to the ideal arrival time (ideal time is the target time, set by the user). If uncertainty is negative then this means that target time is not enough to serve the latency of all components, and must be increased. Furthermore, synthesis does not show any warning about excess of the target

time. Because, $f = 1/T$, frequency approaches 285 MHz (Mega Hertz).

Table 5.2: Clock period estimates in ns.

| Clock | Target | Estimated | Uncertainty |
|---|---|---|---|
| ap_clk | 4 | 3.5 | 0.5 |

Vivado tool, also, provides the latency and interval times which are measured in clock cycles, shown in table 5.3. Clock period is the factor, who determines these numbers

Latency field is the number of clock cycles that are necessary for a pack of input data (frame) to complete a full walk through the design and reach output. Copies of arrays, inside Ccl-Windows algorithm, played crucial role at its minimization, because, they allowed parallel accesses to multiple data and reduced the number of pipeline latency cycles.

Interval is the minimum cycles that have to pass, until the next input (frame) can enter hardware processing unit. Here, the use of a line buffer in component, box filter, proves to be the main reason, interval time is reduced and is, finally, smaller than latency time.

MATLAB times in table 5.1 are shown in milliseconds and represent various latency times of MATLAB model. As a result, latency of hardware design has to be transformed into milliseconds, so that it can be compared with MATLAB model. Overall latency in milliseconds (ms) is the number of clock cycles multiplied with clock cycle period. But, vivado specifies clock cycle period in nanoseconds (ns), so, latency has to be multiplied with factor, $10^{-6}$, in order to achieve the transfer from nanoseconds to milliseconds.

$$Latency_{ms} = clock\ cycles * clock\ period_{ns} * 10^{-6}$$

Final clock cycles could not be provided by Vivado itself, because, there were regions of code that did not have constant loop bounds. Carefully, these bounds were provided to the tool for best and worst case with the placement of command, loop tripcount, for correct verification of the behavior of final design. The minimum and maximum latency in ms for both MATLAB and hardware are presented in table 5.3.

Table 5.3: Matlab versus Hardware latency for a single frame in ms.

| Matlab Latency | | Hardware Latency | | |
|---|---|---|---|---|
| min | max | min | avg | max |
| 70 | >150 | 5.07 | 5.52 | 5.98 |

The previous table can be translated to frames per second, by dividing 1000ms (= 1s) with each of these times and taking the lower bound of each division.

Table 5.4: Matlab versus Hardware latency for a single frame in fps (frames per second).

| Matlab Latency | | Hardware Latency | | |
|---|---|---|---|---|
| min | max | min | avg | max |
| <6 | 14 | 167 | 181 | 197 |

Finally, the speed-up, that hardware achieves, is analyzed in the extreme situation, where minimum latency of matlab model is divided with minimum, average and maximum latency of hardware. By analyzing only the minimum time of matlab, ends up in safe conclusions about the speed-up of hardware. Speed-up times are shown in table 5.5.

Table 5.5: Speed-up estimates.

| Speed-up (x times) | | |
|---|---|---|
| min | avg | max |
| 11.7 | 12.7 | 13.8 |

## 5.3 Energy Consumption

This system was, initially, going to be designed to fit model, Nexys 4, so as it could follow the previous implementation of Badogiannis, which was quite efficient for energy consumption. Until ccl-windows component, the design could indeed fit Nexys. But, its insertion increased too much the used resources, due to the use of multiple large arrays, and Nexys could no longer serve the requirements. In the end, (ZCU102) was the most acceptable FPGA for the system to fit in. During synthesis, Vivado tool made the following utilization estimates:

Table 5.6: Utilization estimates.

| Name | BRAM_18K | DSP48E | FF | LUT | URAM |
|---|---|---|---|---|---|
| DSP | - | - | - | - | - |
| Expression | - | - | 0 | 36 | - |
| FIFO | 0 | - | 179 | 926 | - |
| Instance | 1,115 | 32 | 179,927 | 184,528 | 0 |
| Memory | - | - | - | - | - |
| Multiplexer | - | - | - | 36 | - |
| Register | - | - | 6 | - | - |
| Total | 1,115 | 32 | 180,112 | 185,526 | 0 |
| Available | 1,824 | 2,520 | 548,160 | 274,080 | 0 |
| Utilization (%) | 61 | 1 | 32 | 67 | 0 |

The previous table represents the resources used until the stage of synthesis in Vivado HLS. There were no actual tests made on ZCU102 itself, but it is clear that there is enough space left for the overall component of this project which includes the production of a single frame, to be inserted into an actual system that combines frames and be used by the actual FPGA.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

This thesis has to able to discover, in the best way possible, if parts of fishing nets are cut off and marine population does not sufficiently get trapped in them. Its main objective is to improve previous attempts that are not capable of recognizing correctly the existence or not of faulty parts in nets when conditions vary. The main conditions that affect the final results are by far illumination variation and different sizes of holes that appear in a single frame. These circumstances have to be encountered in order to produce a final product which will be appropriate to fit in a real time system.

Image smoothing tends to clarify the difference of the absorbed illumination between nets and sea water. A morphological operation follows up to emphasize the characteristics of these regions and distinguish them better (edge detection). Then two images will be formed. One provides the nets while the other the defective holes. The premier one is mainly responsible for providing hazy areas where analysis is still not clear enough to detect nets morphology. Finally, the size of each hole in the latter image is determined with the help of connected component labeling algorithm, and the image breaks down into windows. The sizes in each one are then analyzed and the pixels of the faulty ones are marked into the final image. The nets from premier image are also put into the final image to form the final output frame. To extend the effectiveness of the system, every two consecutive frames are combined to produce better results. The overall procedure seems to work quite well despite the intervention of haze and light variation.

Regarding the performance of the system there are quite good implementations. When it comes to the software, the way that image smoothing is done, where the latency depends from the radius of the filter rather than the actual size of image, significantly improves the running times in software. Also, the use of built in functions ensures the best possible performance. On the other hand, the hardware implementation is effective enough to provide a speed up of about 12x times faster than the necessary time needed for MATLAB. The filter of image smoothing proceeds as fast as possible with use of pixels as early as they enter the matrix used for their storage. The other part which has interest is ccl-windows algorithm where there are two passes used for connected component labeling and windows are processed sequentially. The first part cannot be especially improved, but windows

can be managed in parallel. This of course will provide more needs in resources due to the fact that the function of managing a single window will appear multiple times.

## 6.2 Future Work

All in all, this thesis provides very good steps to be considered as a practical system to fit in a real time embedded system. However, there are improvements that can be made. When it comes to software part:

- New methods can be used inside the areas of uncertainty to make a better statement about what is included inside them.
- A more stable model can be found which will be able to analyze the conditions of the environment and be adjusted automatically to them, rather than requiring the user to change any parameters (the threshold and offset values in this occasion).
- Connected component labeling method (which is thought to be an integral part for this type of problems, by determining a specific label to each hole that helps in the calculation of their size) can be done with one pass on the image rather than two which is the most common case used, just like in this project.

In hardware, there is space for improvements for both sectors of performance and results' quality of the algorithms:

- Connected component labeling can be parallelized furthermore. Windows can be processed in parallel rather than sequentially with the trade off at more resources usage.
- The part of image opening, which is done with the contribution of Vivado's built in functions (8 bit wide pixels as inputs), can be achieved with new user functions that manipulate 32 bit wide pixels. This will improve hardware results at the section of calculating the nets where some gaps appear compared to MATLAB results.
- The overall hardware module can be expanded to a regular real time hardware implementation, alongside all the components needed (DMA(s), controller(s), memory module(s) etc) to hold a real time embedded system.
- Estimations about power consumption can be made when a real time system is implemented.
- Power consumption can be improved if any new specifications can fit in a smaller FPGA.

# Bibliography


[1] N. Badogiannis, K. Moirogiorgou, M. Zervakis, A. Dollas, N. Papandroulakis. *Real-Time Embedded System for Hole Detection in Fish Cage Nets*. IEEE International Conference on Imaging Systems and Techniques (IST), Dec. 8-10, Abu Dhabi, UAE, IEEE, 2019.

[2] Kaiming He, Jian Sun, and Xiaoou Tang. *Single Image Haze Removal Using Dark Channel Prior*. IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 33, No. 12, December 2011.
https://github.com/IsaacChanghau/OptimizedImageEnhance/tree/master/papers
http://kaiminghe.com/publications/eccv10guidedfilter.pdf

[3] Ahmad Shahrizan Abdul Ghani, Nor Ashidi Mat Isa. *Underwater image quality enhancement through Rayleigh-stretching and averaging image planes*. International Journal of Naval Architecture and Ocean Engineering, Volume 6, Issue 4, December 2014, Pages 840-866
https://www.sciencedirect.com/science/article/pii/S2092678216302588#!

[4] Amit Shirsat and Bharat Bhargava. *Local geometric algorithm for hole boundary detection in sensor networks*. Published online 11 March 2011 in Wiley Online Library
https://onlinelibrary.wiley.com/doi/pdf/10.1002/sec.271

[5] Wenhao Zhang, Ge Li, Zhenqiang Ying. *A New Underwater Image Enhancing Method viaColor Correction and Illumination Adjustment*. School of Electronic and Computer Engineering, Peking University Shenzhen Graduate SchoolLishui Road 2199, Nanshan District, Shenzhen, Guangdong Province, China 518055

[6] Sonali Sachin Sankpal and Shraddha Sunil Deshpande. *Nonuniform Illumination Correction Algorithm for Underwater Images Using Maximum Likelihood Estimation Method*. Research Article, Open Access, Volume 2016, Article ID 5718297
https://doi.org/10.1155/2016/5718297

[7] Adam Taylor. *Using HLS on an FPGA-Based Image Processing Platform*. Published Online May 31, 2018
https://www.hackster.io/adam-taylor/using-hls-on-an-fpga-based-image-processing-platform-8f029f

[8] Valery Sklyarov, Iouliia Skliarova and Alexander Sudnitson. *FPGA-based Accelerators for Parallel Data Sort*. University of Aveiro/IEETA, Portugal, Tallinn University of Technology, Estonia,

Volume 16: Issue 1, Published online: 27 Jan 2015.
https://doi.org/10.1515/acss-2014-0013

[9] Stephen Neuendorffer, Thomas Li, and Devin Wang. *Accelerating OpenCV Applications with Zynq-7000 All Programmable SoC using Vivado HLS Video Libraries*

[10] M. Park, K. Brocklehurst, Robert Collins, and Yanxi Liu. *Deformed lattice detection in real-world images using mean-shift belief propagation*, IEEE Transactions on Pattern Analysis and Machine Intelligence (2009) (english).

[11] Vaibhav Gadewar Radhika Chandwadkar, Saurabh Dhole, Deepika Raut, and Prof. S. A. Tiwaskar. *Comparison of edge detection techniques*, 6th Annual Conference of IRAJ (2013).

[12] Kurt Schwenk and Felix Huber. *Connected component labeling algorithm for very complex and high resolution images on an fpga platform*, SPIE Remote Sensing. International Society for Optics and Photonics (2015).

[13] N. Senthilkumaran and R. Rajesh. *Edge detection techniques for image segmentation – a survey of soft computing approaches*, International Journal of Recent Trends in Engineering, Vol. 1, No. 2 (2009).

[14] Bruce A. Draper, J. Ross Beveridge, A.P. Willem B¨ohm, Charles Ross, Monica Chawathe, and Jeffrey Hammes. *Accelerated image processing on fpgas*, IEEE Transactions on Image Processing (2003).

[15] Ilkoo Ahn and Changick Kim. *Finding defects in regular-texture images*, Korea Advanced Institute of Science and Technology (2009).