TECHNICAL UNIVERSITY OF CRETE

DIPLOMA THESIS

Reconfigurable Logic-Based Real-Time Automatic Video Calibration and Processing to Detect Holes in Aquaculture Nets.

Author:
Eirini Ntafi

Thesis Committee:
Prof. Apostolos DOLLAS
Prof. Michael ZERVAKIS
Dr. Nikolaos PAPANDROULAKIS (HCMR)

A thesis submitted in fulfillment of the requirements for the diploma of Electrical and Computer Engineer

in the

School of Electrical and Computer Engineering
Microprocessor and Hardware Laboratory

22 November, 2022

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

Electrical and Computer Engineer

Reconfigurable Logic-Based Real-Time Automatic Video Calibration and Processing to Detect Holes in Aquaculture Nets.

by Eirini Ntafi

As Computer Vision technology is developing more and more in the last decade, the areas of application expand to many research sectors, some of which are for underwater applications, such as in aquaculture. The main goal of this thesis was to create an embedded system that detects defective holes in aquaculture nets from underwater videos in various realistic conditions. First, we evaluate an embedded system that is already build against new video data that were taken in more realistic conditions, including marine fouling on the nets and fish in the background; such conditions were never tested before. We created a classification system according to the specific characteristics and problems that we identified as common in the videos. Subsequently we improve the detection capabilities of the system, either by using parameter calibration, or by adding an extra logic block in the existing system. After testing, the new system resulted on average x6 times better results over a broad range of realistic videos, which with the previous system were not detected at all. The software application was developed with MATLAB and the hardware design block was created using Vitis Unified Platform software (targeting the Alveo U50 Card). In conclusion, the system was improved to work on more realistic underwater conditions and addresses some of the limitations of the previous system.

TECHNICAL UNIVERSITY OF CRETE

Abstract

School of Electrical and Computer Engineering

Electrical and Computer Engineer

Reconfigurable Logic-Based Real-Time Automatic Video Calibration and Processing to Detect Holes in Aquaculture Nets.

by Eirini Ntafi

Καθώς η τεχνολογία Computer Vision αναπτύσσεται όλο και περισσότερο την τελευταία δεκαετία, οι τομείς εφαρμογής επεκτείνονται σε πολλούς ερευνητικούς τομείς, ορισμένοι από τους οποίους αφορούν υποβρύχιες εφαρμογές, όπως στην υδατοκαλλιέργεια. Ο κύριος στόχος αυτής της διπλωματικής εργασίας ήταν η δημιουργία ενός ενσωματωμένου συστήματος που ανιχνεύει ελαττωματικές τρύπες σε δίχτυα υδατοκαλλιέργειας από υποβρύχια βίντεο σε διάφορες ρεαλιστικές συνθήκες. Αρχικά, αξιολογούμε ένα ενσωματωμένο σύστημα που έχει ήδη αναπτυχθεί στο Πολυτεχνείο Κρήτης με βάση νέα δεδομένα βίντεο που έχουν ληφθεί σε πιο ρεαλιστικές συνθήκες, συμπεριλαμβανομένων θαλάσσιων ρύπων(φύκια) στα δίχτυα και ψαριών στο παρασκήνιο, που δεν είχαν δοκιμαστεί ποτέ πριν. Δημιουργήσαμε ένα σύστημα ταξινόμησης σύμφωνα με τα συγκεκριμένα χαρακτηριστικά και προβλήματα που εντοπίσαμε ως κοινά στα βίντεο. Στη συνέχεια τα βελτιώνουμε είτε χρησιμοποιώντας βαθμονόμηση παραμέτρων, είτε προσθέτοντας επιπλέον λογική στο υπάρχον σύστημα. Μετά τη δοκιμή, το νέο σύστημα οδήγησε σε επιτυχία κατά μέσο όσο 6 φορές βελτίωση πάνω σε ρεαλιστικά βίντεο για τα οποία το προηγούμενο σύστημα είχε πολύ χαμηλά ποσοστά επιτυχίας. Η εφαρμογή λογισμικού αναπτύχθηκε με το MATLAB ενώ το μπλοκ σχεδιασμού υλικού δημιουργήθηκε χρησιμοποιώντας το λογισμικό Vitis Unified Software Platform (στοχεύοντας την κάρτα Alveo U50). Συμπερασματικά, το σύστημα βελτιώθηκε για να λειτουργεί σε πιο ρεαλιστικές υποβρύχιες συνθήκες και να εξελίξει το προηγούμενο σύστημα αντιμετωπίζοντας κάποιους από τους περιορισμούς του.

Acknowledgements

First of all I would like to express my sincere gratitude to my advisor, Prof. Apostolos Dollas, for his guidance through every step of this thesis' process. His trust in me and his valuable advice made this thesis possible and helped me grow both personally and academically.

Furthermore, I would like to thank Dr. Nikolaos Papandroulakis and the Hellenic Centre for Marine Research (HCMR) for providing us with the video data for this thesis, as well as Prof. Michael Zervakis for his contribution as a thesis committee member. I would also like to thank all the people in the MHL Lab and the predecessors of this thesis project for their contribution.

Last but not least i would like to thank my family and friends for their unconditional support throughout all these years.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Computer vision is a fast growing field that enables computers and digital systems to derive meaningful information and gain high-level understanding from digital images or videos. This field concerns and analyses imaging data from many areas and for many uses, including Medicine, Neurobiology, Robotic navigation, Autonomous Vehicles and many more. Some of the computer vision tasks include Motion Analysis and Recognition, as in Object Recognition, Identification, or Detection of some specific object, data or condition.

1.1 Problem Statement

There are things that a human eye can not see and situations which could be dangerous or time-demanding when dealing with aquaculture problems and other underwater problems. In aquaculture, the cage nets are a basic core in the creation and the protection of the environment that the living organisms need. In order to do that, either divers or remote vehicles need to take visual material in order to inspect the nets and identify possible problems. One of those problems is the nets wear, which happens either from time or from the conditions(weather, fish). That decay causes tears and defective holes in the nets that need to be fixed so that the environment remains intact. Usually divers have to go underwater to resolve those issues and in this process they come in contact with the sea and its various conditions.

There are specific underwater vehicles such as Remotely Operated Vehicle (ROV) and Autonomous Underwater Vehicles(AUV) which allow us to take

images and videos underwater. Even with those, the divers still have to confront difficulties such as bad visibility, light exposure, marine growth, movement of the waves and more. From the above, we can say that underwater images with nets are complex enough to process.

This thesis shows the different levels of complexity in a variety of underwater videos and helps to get closer in creating an automated underwater net holes detection system in order to find holes in nets faster, cheaper and in a safer way. In order to do that we use image and video processing techniques that are suitable for this task. That way the human power needed for some tasks can be decreased, and also the risks that comes with it.

1.2 Scientific Contributions

This thesis continues the previous work of Th. Zacheilas [1] and N. Badogiannis[4]. While N. Badogiannis' thesis focuses on finding holes in nets of still images, Th. Zacheilas' advances the system to work with videos.

The contribution of this thesis is an advance on the previous system in order to accommodate the needs of new video data provided by the Hellenic Centre for Marine Research (HCMR). There is an evaluation of performance of the previous system for the new data and new changes according to the distinctive processing needs. Specifically, the system was improved in several ways:

- Analysis of the existing system response in more advanced video data.
- A classification system is being created, where the videos are divided in categories according to their different characteristics.
- Parameter calibration is introduced as a way to improve the results of more than one video.
- Creation of automated net color recognition system to help in the defective holes recognition process regardless of the diversity of the new video data given, and also future video data tests.

The most important thing is how the system improves in these new more advanced input video data and can be used even on more difficult and demanding environmental conditions.

1.3 Thesis Outline

- **Chapter 2 - Theoretical Background:** Theoretical analysis of important concepts and algorithms.
- **Chapter 3 - Related Work:** Previous contributions in video processing on fishery nets from our institution.
- **Chapter 4 - Assessment of Algorithms Under Realistic Datasets:** Evaluate results of previous thesis algorithm applied on new data.
- **Chapter 5 - Modeling of the System and Progression:** Algorithmic variations and improvements according to the needs of the system.
- **Chapter 6 - FPGA Design and Results:** FPGA Design of Net Color Detection block and the results of its implementation.
- **Chapter 7 - Conclusions and Related Work**

Chapter 2

Theoretical Background

As the given data is actually underwater videos from aquaculture cages, the problem of detecting defective holes in the cage nets, needs to be approached using video processing techniques. In this chapter we analyze how video processing works and a basis of the algorithms used in the project. In order to fully understand video processing we need to know the basics of image processing as videos are actually images - that we call video frames - put together in a stream. When working in a real-time video each frame has to be processed separately according to the goals of the project and then put together sequentially to create the processed video output.

2.1 Video Processing

In order to process the aquaculture videos and detect defective holes in the cage nets, the basic steps of video processing were followed:

1. Reading a video. Matlab environment has specific functions that allow to import videos from files and read their data.
2. Processing the video. After reading the video, frames are separated and the program goes through each one to do the main processing goal using image processing methods. It is essential to say, that the frames are first converted to grayscale and later binary frames for the holes detection process. There is a loop used to cover each frame the moment it is read.
3. Write back the video as a file or display it on screen. After processing each frame, the program either displays the video right after processing or writes it back and saves it as a video file using Matlab functions.

A very important part of a video processing system is whether the processing is happening in real time or not. If the system is not required to process a video in real time, we can go as far and as expensive it can be in the terms

of algorithmic complexity and hardware use. However, in real time, a video processing system should be able to output the results that are required in the time the video lasts. This requires for the broadcasting devise(system) to perform as well as the processing system. It needs to be both fast - minimum 24fps - and give the correct results.

2.2 Computer Vision

Is a field that studies the interpretation of an image or video from a computer. For a computer to understand and make a conclusion of what it "sees" in that image, how close to reality is this conclusion and how true. One of the most common ways to accomplish this is with Deep Learning techniques where the system is fed with hundreds or thousands of data similar to the image/item they want to recognise and then the model learns on its own the different features that make up the item.

An other way is to apply morphological operations about information of specific characteristics of the image/item so that the system can be able to recognise them. This can be used both for pre-processing systems before a Deep Learning model can be applied as well as in a standalone system for final estimation and interpretation of the needed image/item.

In both cases the system must be able to 'recognise' and label videos, images, or parts of them. In order to do that, we need know how to process an input video/image, extract information from it and classify the image or parts of it.

2.3 Image Classification

Is the process of assigning labels to an image from a predefined set of categories. Image classification is a very important procedure in computer vision, as it assists a system in 'seeing' and recognizing an image or objects in it.

Video Classification

Continuing in the same way of thinking for Image Classification, Video Classification is the process of categorising whole videos or parts of them by putting labels on them according to their characteristics. In video classification, it would be safe to check each frame, classify each one of them into

the pre-existed categories and then choose the most used label for the whole video.

However, if the process takes a lot of resources or a lot of time, this could be very nonperforming. Considering this, when the video classification process is a part of the video process in real time, it has to be easy, quick and inexpensive to do while the video is taken.

2.4 Image Segmentation - Morphological image processing

As an extension of image classification, in digital image processing and computer vision, image segmentation is the process of partitioning a digital image into multiple image segments, also known as image regions or image objects (sets of pixels). The goal of segmentation is to simplify and/or change the representation of an image into something that is more meaningful and easier to analyze.[1][2] Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

As the main process in Video Processing, focuses on using algorithms to alternate each frame in a way that the information needed can be extracted, it is easier to approach separated frames, treat them as images and process them separately with morphological image processing methods. Once an image-frame is subtracted, it is enhanced by getting through a series of filters that depend on the processing goal. Some of them may be used for Noise reduction, Feature extraction, Thresholding and Recognition.

In this thesis, various methods of image segmentation are used in order to extract all information needed for the purpose of identifying defective holes in aquaculture nets. Some of them are *Guided Filter* and *Box Filter*, which are edge-aware and haze removal filters used to enhance each frame before it is used for the holes detection process. These filters are implemented as described in Th. Zacheilas' thesis and have the same use. Some more of these methods are presented below and are fundamental for the following thesis chapters.

2.4.1 Connected Component Labeling

As described in Th. Zacheila's thesis, 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 N. 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, 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.

CCL algorithm gives each hole or "possible" hole a number that is called label for each pixel that is in the hole. Every other pixel of the image that is not a hole is 0. So there is an array that holds all the hole labels and their sizes which later is used to find the defective, or possibly defective holes.

2.4.2 Thresholding

Thresholding in image processing is the simplest method of segmentation that helps in creating a binary image. In this process each pixel of a grayscale image must be replaced with a black or a white pixel so that the final image becomes a binary one. The way pixels are replaced with black or white is through a threshold value comparison. First a threshold value is chosen for the whole image and then each pixel is compared to this value and is replaced with black if the intensity of the pixel is less than this value and white if not. In this thesis it is used to separate the background(sea) from the foreground(nets) in each frame and assign them as black or white.

Adaptive Threshold

The adaptive thresholding technique is introduced in T.Zacheila's thesis for this specific task. It is a thresholding method that separates the foreground from the background in a nonuniform illumination environment. It breaks down the image into windows and extracts local thresholds with mean or median values. It takes a given window size and exports a binary image that clearly separates the foreground which is later taken advantage to characterize is as nets.

It already exists in the previous version of the algorithm and later on, the adjustments made through parameter configuration in the method, are important in order to adapt in new conditions according to the new data.

2.4.3 Flood Fill

Flood fill or seed fill is an algorithm with the purpose of coloring an entire area of connected pixels with the same color and it's often used in tools like the "bucket tool" color fill in the paint program.

The most common version of the algorithm lets the user select a pixel in an image, and then fills all the surrounded same-colored pixels with the color of the users choice. There are two ways to implement the algorithm, called 4-way and 8-way. The difference between them is that in the 4-way flood-fill implementation the algorithm moves in 4 directions while in the 8-way implementation it moves in 8 directions (same way as the connectivity in CCL). Practically this means that the 8-way version of flood fill could be more efficient for narrow spaces like diagonal lines but it can also be useless for spaces enclosed in narrow lines as it will leak from the diagonal pixels, shown in Figure 2.1. The red dot symbolises the starting point and the red X the area that is filled in different situations.

This is usually implemented with a recursive function where the algorithm checks all the surrounding pixels of the selected one and if their color matches the color of the selected pixel, then it replaces their colors with the color of the users choice. And continues to do so for the next pixels in the area in the same way.

Specifically in the 4-way version of the algorithm that is applied on an image, the first step is to set a starting point(a pixel in the area the user wants to fill) and request a color to be filled in the area around that starting point.

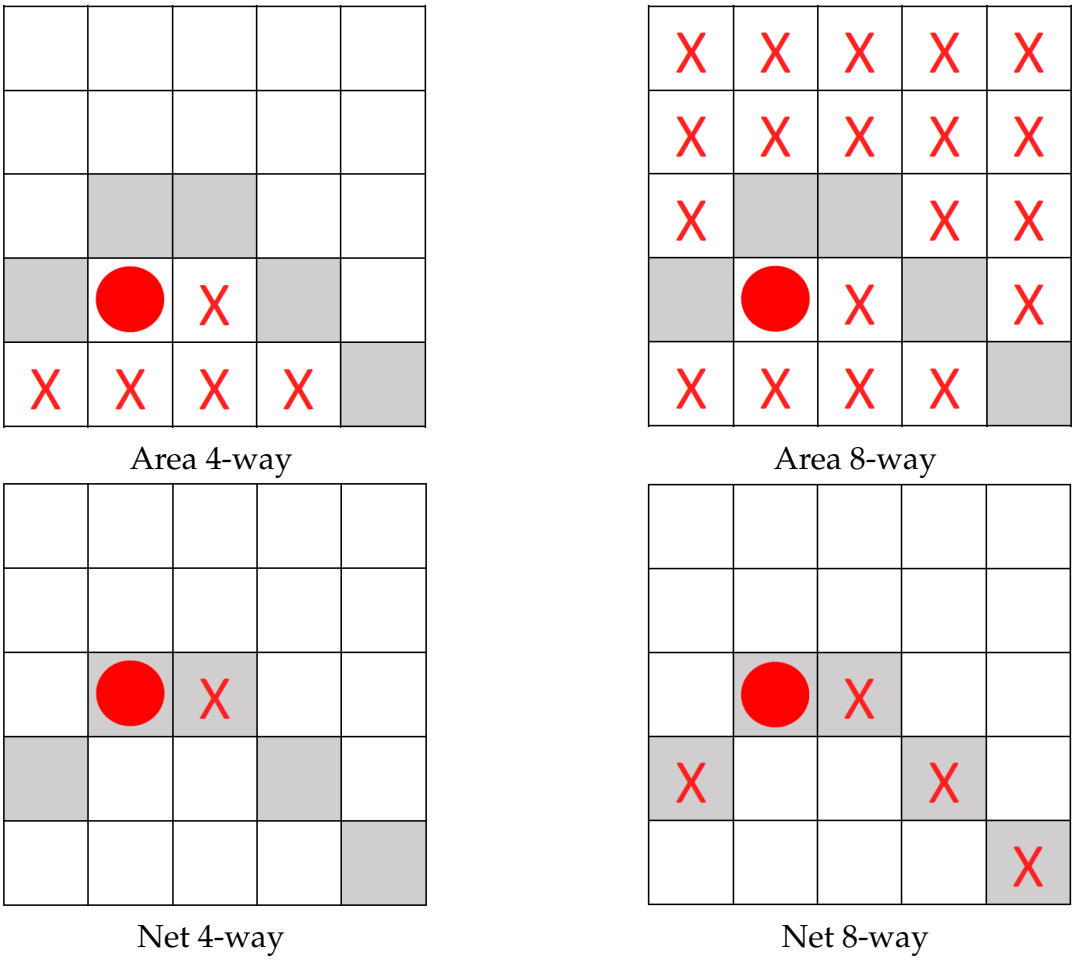


FIGURE 2.1: Flood Fill behavior

The algorithm then checks if the pixel is within the image limits and also if it is not already the requested color, or else it exits. If not, then it changes the pixel to the new color requested and searches the surrounding pixels in four directions - north, east, south, west - for the same color. Each time it's calling the same function and passes the new direction pixel as the start point, checking again for the image limits and the color of each pixel. The function ends when the region of interest is filled with the chosen color.

Matlab already has a function called imfill() that does fill image regions from a user specified point. However, as recursion is not supported in Vivado HLS tools, the algorithm used in this project is based on a non-recursive method with a stack. This works in the same way with the only difference being that instead of moving to the next pixel with recursion, is uses a stack to push every new position and a while loop that continues until that stack is empty.

2.5 Marine Fouling

Marine Biofouling, also known as marine fouling or marine growth, occurs when microorganisms, plants and algae attach themselves to underwater objects like boats, ropes and nets, pipes and building structures.

The primary focus in fish culture relates to the mitigation of net fouling, as this leads to compromised cage structure and detrimental effects on fish health mainly through low flow-through of water, leading to poor dissolved oxygen availability. From this thesis perspective, marine fouling also lowers the visibility in our videos which increases the probability of errors in the defective holes detection algorithm.

There are many methods of preventing the attachment of microorganisms on underwater surfaces called antifouling methods. However, for this project the goal is not to prevent marine growth from occurring rather than accept the existence of marine fouling on the aquaculture cage nets, and recognise the problems that come with that.

2.6 Tools and Libraries Used

2.6.1 For Software Implementation

Matlab tool was used for the initial modeling process of the system. With Matlab tool the basic algorithms and different methods were tested before deciding if the results are satisfactory to proceed to a hardware design.

2.6.2 For FPGA Implementation

For the old hardware design, Xilinx Vivado tools were used and specifically the Vivado HLS 2019.1 for the simulation of the system which later were implemented on an FPGA device(ZCU102).

The new design targeted an Alveo U50 Acceleration Card, using Vitis Unified Software Platform 2020.2 tool and is described in detail in chapter 6.

2.6.3 Libraries

OpenCV consists of Computer Vision Library tools that were used both in software implementation as well as in hardware implementations.

The Vitis vision library has been designed to work in the Vitis development environment, and provides a software interface for computer vision functions accelerated on an FPGA device. Vitis vision library functions are mostly similar in functionality to their OpenCV equivalent.

Chapter 3

Related Work

As Computer Vision technologies evolve, Technical University of Crete in collaboration with the Hellenic Centre for Marine Research took the initiative to study and explore the technological possibilities of underwater videos from fisheries. Each research is done with the goal of creating an automated Net Hole Detection System as a whole. The following works are a basic core we follow to continue and subsequently reach our goal.

3.1 Previous Theses at TUC

Nikolaos Badogiannis' Thesis

Nikolaos Badogiannis' thesis[3] with title "Real-Time Embedded System for Hole Detection in Fish Cage Nets" is the first one to introduce this subject of processing videos from fisheries in our institution. In this thesis, two methods were developed and tested on still images which simulate nets with holes in ideal conditions and low complexity.

The first method used to pinpoint holes is *Template Matching* which works by setting a template image that has the information we need to locate and matching it with our source image to compare and calculate the difference degree of the pixels in the two images. The second method used is *Edge Detection* which first blurs the initial image with a Gaussian filter and uses Sobel filter thresholding to convert to binary. Then dilates the image and uses Connected Component Labeling algorithm to label the possible holes and determine which are the defective ones by their size(at least double size).
In both methods the results are satisfactory as the system actually detects holes in ideal conditions and makes some mistakes in more complex ones.

Specifically the Template Matching is a good method for controlled conditions and repeatable patterns while this Edge Detection method seems to be more flexible and precise where there is not much noise in the images.

Stavros Paspalakis' Thesis

In this thesis[4] a totally different approach is developed with the same goal: to detect defective net areas. There are used two processes a local process and a global process. In the local process the frame is divided in parts and in each part are applied algorithms of a local Hough transform, edge detection and line detection, then uses the Otsu method for detection and then the line distances for discontinuities. In the global process the whole frame goes through a Hough transform where it gets patterns from sets of points in the frame(lines, circles, ellipses). Both of the processes are needed for the different lighting and background.

Theofilos Zacheilas' Thesis

The next thesis with title "Reconfigurable Logic-Based System for Image Processing of Fishery Nets"[2] follows N. Badogiannis' Thesis. Here, Theofilos Zacheilas modifies the second method to be implemented in a series of frames(video) and then test it and change it according to the needs of a real video from aquaculture nets. In order to do that, the following things are developed for each frame. Firstly, a haze image model was applied to determine a hazy factor for the brightness between water and nets. Then the image goes through a guided filter that does Edge Detection, Haze removal and Lighting Smoothing for more clear detection of holes and nets. After that, Connected Component Labeling is applied to separate the holes as different components and label them, so later it can brake down the frame in windows to determine the defective holes from their size in each window and then marking of all faulty holes and nets in the final image. Last, a combination and comparison of consecutive frames is done for safer conclusions in the whole video.

3.2 Thesis Approach

While all the above previous work are excellent each one on its own, we are called to work on new challenges and see what happens when the input data(images or videos) are not similar to those already tested. In the first

two thesis the image data used, is controlled or simulated, while the third is tested in a real video from fisheries but with very good environmental conditions. In this thesis we continue and analyse some of the previous thesis proposals in order to improve the methods already used according to new, more realistic and complex data models.

Chapter 4

Assessment of Algorithms Under Realistic Datasets

In order to improve on the previous work, an evaluation of it is necessary. With new data that was given by the Hellenic Centre for Marine Research (HCMR), the existing project was tested on 13 new videos. We name the videos with numbers 0 to 13 and refer to them like this from now on, with Video 0 being the reference video used as a base for the initial system design. This chapter, analyses the characteristics of the videos and the performance we get when we apply the already existing algorithm on them.

4.1 Video Classification System

The first thing needed to be done was the assessment of the new video data. For that purpose, five new categories were created according to some characteristics seen in the videos (Table 4.1).

The first category refers to the color of the nets each video has. Specifically, it is the concept of whether the nets are darker or lighter than the background, and so when turned into binary, they become black or white. In the second column the videos were categorised according to the amount of marine growth that is attached on the aquaculture nets. It varies, from videos with clear nets, to those with very high concentration of marine fouling. Another category is about the camera's position, and specifically if it is close to the nets or not. Last, two more categories were created that indicate if a video has noise(other objects than the nets) in its foreground or its background. These last two categories are important for the videos evaluation result and not the video processing itself, as they help to get more information about the systems errors and successes.

Video	Color of Nets	Marine Growth	Zoom & Camera Movement	Foreground Noise	Background Noise
v0	White	No	No	No	No
v1	Black	High	Yes	Yes	No
v2	Black	High	No	Yes	Ye
v3	Black	High	No	No	Yes
v4	Black	High	Yes	Yes	Yes
v5	Black	Low	No	No	Yes
v6	Both	Low-Medium	No	No	Yes
v7	Black	Medium	No	Yes(hand)	Yes
v8	Black	Medium	No	No	Yes
v9	Black	Medium	Yes	No	Yes
v10	Black	Medium	No	No	Yes
v11	Black	Medium-High	Yes	Yes	Yes
v12	Black	Medium	Yes	No	Yes
v13	Black	Low	Yes	No	No

TABLE 4.1: Video Classification.

4.2 Presenting the New Video Data

In his thesis[1], Th. Zacheilas works on a video (Video 0) which has specific environmental parameters, so the method and the algorithm produced, try to fit and work best with the characteristics of this video.

When a frame from Video 0 (4.3) is extracted, it is clear to see where the nets

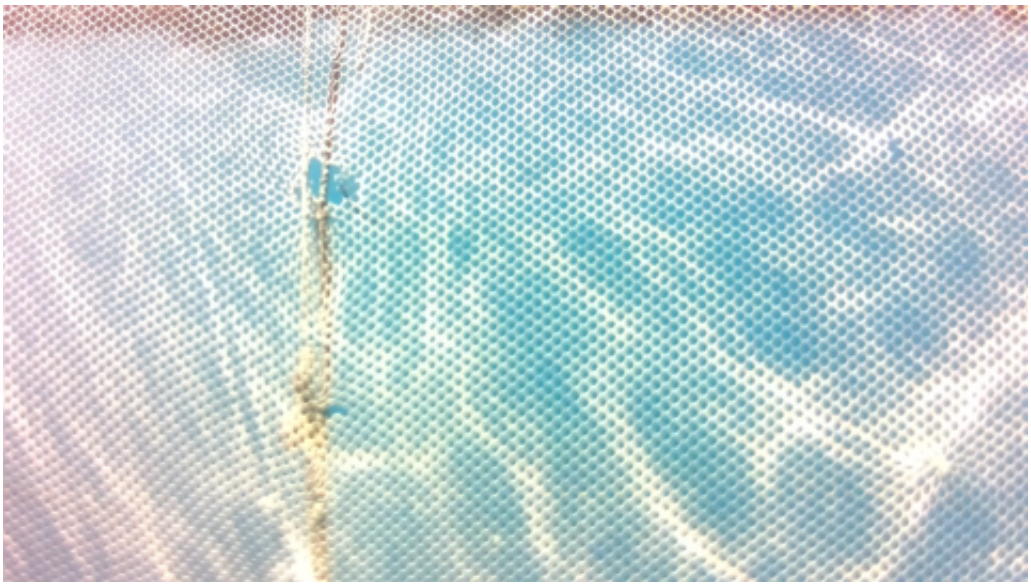


FIGURE 4.1: Frame from Video 0

and the defective holes are. The video has great exposure and the nets do

not have any marine growth on them or other physical interference on the foreground and background of the nets.

The same can not be said for the new video data. One of the first observations made when the algorithm was applied to the new videos is how much the complexity increases. For example in a frame from Video 3 (4.2), the first difficulties arise and are easy to point out just by looking at the image. It is difficult to find the defective hole(s) in the frame even with our eyes. There is a lot of marine fouling attached on the nets which means both foreground and background noise is stronger and the video is taken from an angle which again makes it hard to see a hole-shaped item, whether it is defective of not. It is not easy to determine the boundaries where the actual nets end, and the background begins.

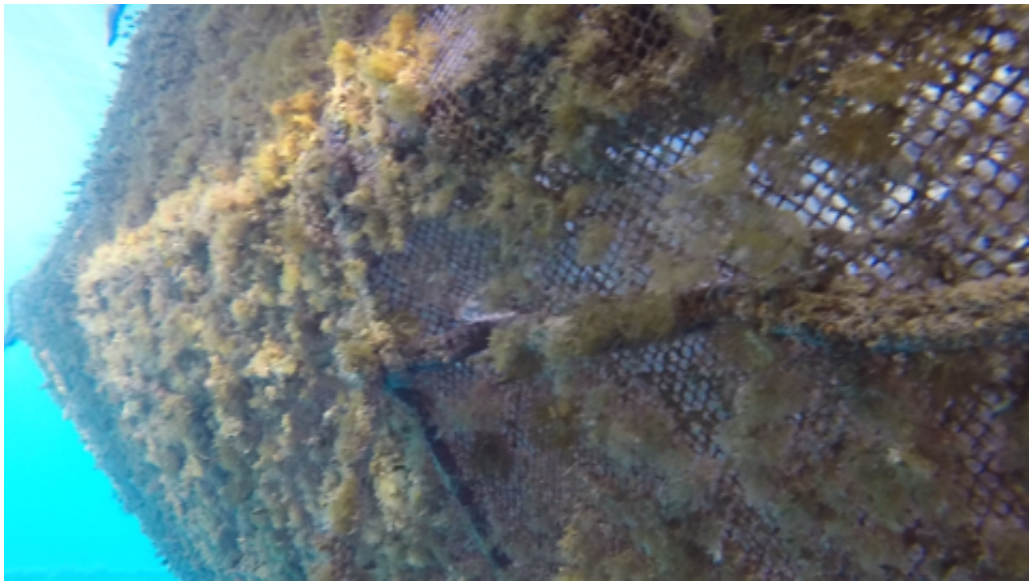


FIGURE 4.2: Frame from Video 3

The same is for Video 1(4.3) with additional noise. Video 1 as one can see, has a lot of marine growth attached on the nets and has a background object on them as well.

A more intense environment can be seen in Video 2(4.4). There are already closed holes and in addition to the marine growth add up to more foreground noise. This concludes to an image that is difficult to read and extract information at first sight. From this specific frame it is not easy to make an observation of whether there is a defective hole or not.

The same goes for Video 4(4.5). The environment is not good to determine whether there are holes in the nets or if there are, how many they are.

FIGURE 4.3: Frame from Video 1

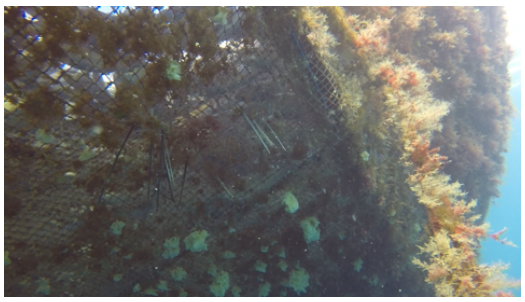


FIGURE 4.4: Frame from Video 2

FIGURE 4.5: Frame from Video 4

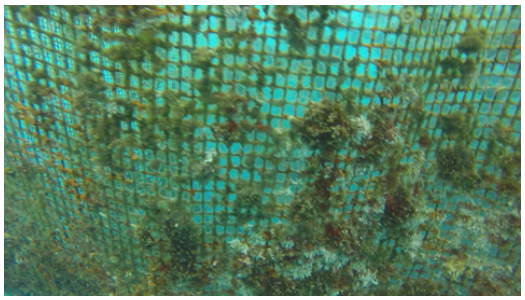


FIGURE 4.6: Frame from Video 9

In Video 9(4.6) however, the visibility is much better. The structure of the nets is clear and with the naked eye it is easy to see that there are no holes in this frame. It does have marine growth attached to the nets, but the quantity is much less than that of previous videos.

Video 6(4.7) and Video 7(4.8) seem quite similar in colors, in noise and in distance from the camera. The camera is positioned further than that of the previous videos and as the videos proceed, it changes and gets different points of view.

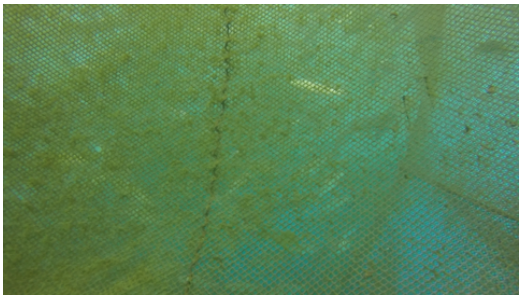


FIGURE 4.7: Frame from Video 6

FIGURE 4.8: Frame from Video 7

Video 5(4.9) is very clear. It does not have any marine growth attached on the nets and apart from the background fish, it is the closest video data to the

original Video 0. On the other hand, Video 10(4.10) has more marine growth noise and the colors are different. Both of them have more sea on the scope than other videos, which changes the uniformity of each frame.

FIGURE 4.9: Frame from Video 5

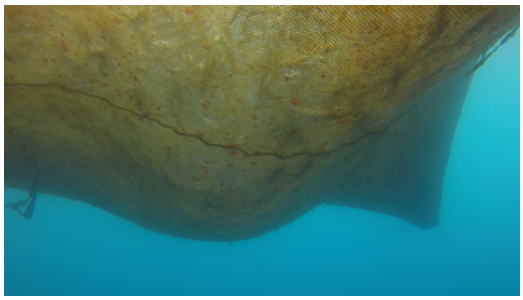


FIGURE 4.10: Frame from Video 10

Video 8(4.11) and video 11(4.12) both have a lot of marine fouling but the colors and the exposure are very different. In Video 8 the structure of the nets is clear while in Video 11 it has more marine growth attached on them. More of the supportive frame of the nets is also visible in both frames which is categorised as foreground noise.

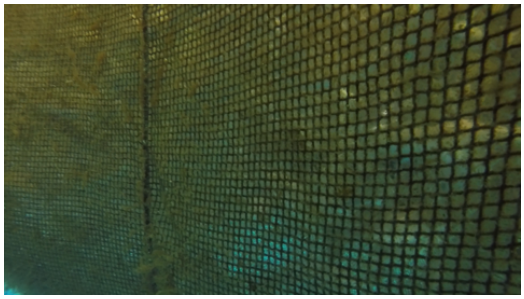


FIGURE 4.11: Frame from Video 8

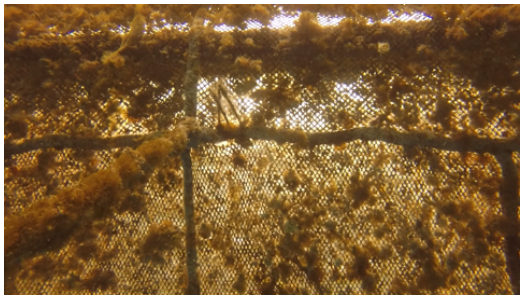


FIGURE 4.12: Frame from Video 11

In the last two videos, Video 12(4.13) and Video 13(4.14) there is a big difference in the vision. The frames are taken from a closer range and even though the nets are not completely clear, they are clear enough for the human eye to detect the defective holes easily.

From the above brief presentation of the new video data combined with information from Table 4.1, it is easy to observe that there is one main characteristic that remains the same in all the new videos and that is the color of the nets. In all new videos the foreground(nets) is darker then the background(sea) and when converted to binary, makes the nets to be black, in

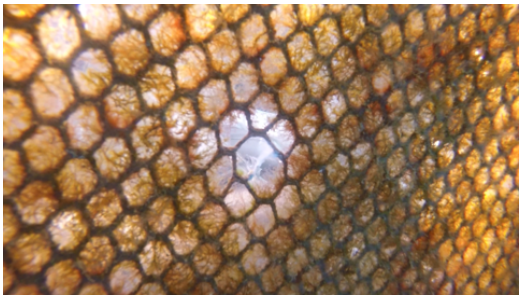


FIGURE 4.13: Frame from Video 12

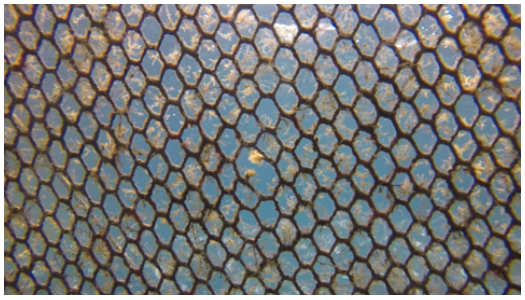


FIGURE 4.14: Frame from Video 13

contrast to the old video where the nets are lighter and so they appear white in the binary conversion. The original algorithm is planned for white/ light colored nets so, where the new videos have black / dark colored nets, the algorithm gets confused and shows wrong defective holes in the place where the actual nets are. For example in Video 13 (4.15) it is clear that the red pixel areas are on top of where the nets are supposed to be and not the holes.

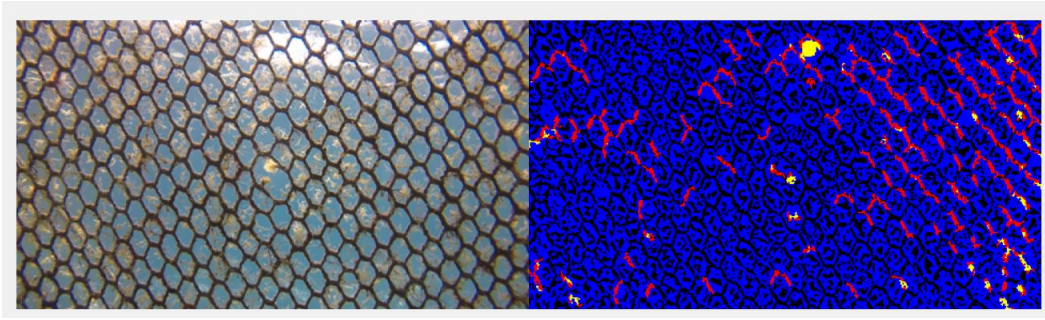


FIGURE 4.15: Frame with results from Video 13
Faulty dark nets

It would be logically wrong to evaluate and make assumptions of how the algorithm works on these new videos without changing this basic detail first as it is no longer detecting holes, but nets. After changing this detail the new videos were tested in a way where the algorithm detects the nets whichever the color and shows a better logical result.

In the next chapter(5) there is a detailed analysis on how this new classification of the nets into black or white is done autonomously and more details about the new video dataset.

4.3 Assessment of the Pre-Existing Method

The first step was to test the existing algorithm from Th. Zacheilas' thesis as it is, for all the new video data(with the additional black to white nets color change). The goal in this section is to introduce the new videos and take a look at the challenges each one presents. Observations also need to be made of how well the algorithm responds to different data situations and how different and challenging the new environmental conditions will be to get the expected outcome. One or more frames are presented of each video with the output results of the initial algorithm that show the difficulties, the errors and the successes of each video. Some of them are common between them, some appear multiple times and some of them exist only in specific moments-frames. In the figures presented in the following section, the defective holes are shown with red color and yellow represents the potential defective holes, that are compared in consecutive frames and either defined as defective holes in the next frames or discarded.

4.3.1 Analyzing Video Data

Video 0

Video 0 (4.16) is the one that was already tested in the pre-existing algorithm in thesis[1] and the results are excellent as we can see in figure 4.4.

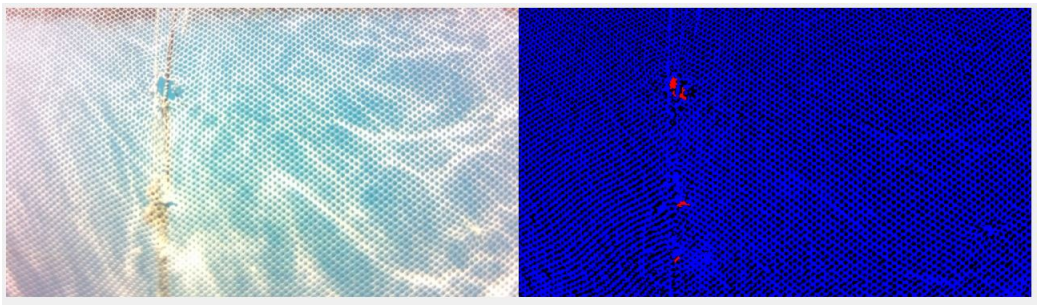


FIGURE 4.16: Frame with results from Video 0

The image is very clear, the nets are clean and the holes in them are easily spotted.

Video 1

A sample frame of the first new video we tested in Figure 4.17. This is the only video that is originally in 568x320 resolution. In this specific frame at first glance, the response of the algorithm is faulty, but it can also be seen that it spotted as red(=defect) in the area where the actual defective hole is. The problem is that it also 'sees' more defective holes, and potential defective holes where there actually are none.

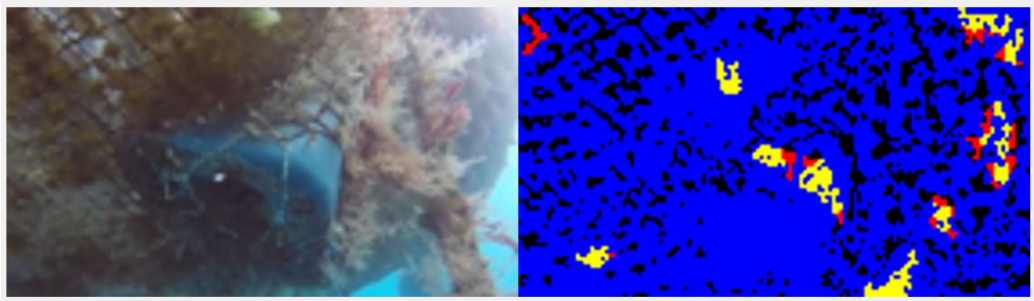


FIGURE 4.17: Frame with results from Video 1

Video 2

In figure 4.18 is a frame of the second new video tested. Even with the naked eye it is not easy to define if there is a defective hole and where. The system responds with the output of many potential defective holes and some defective holes even though there are none.

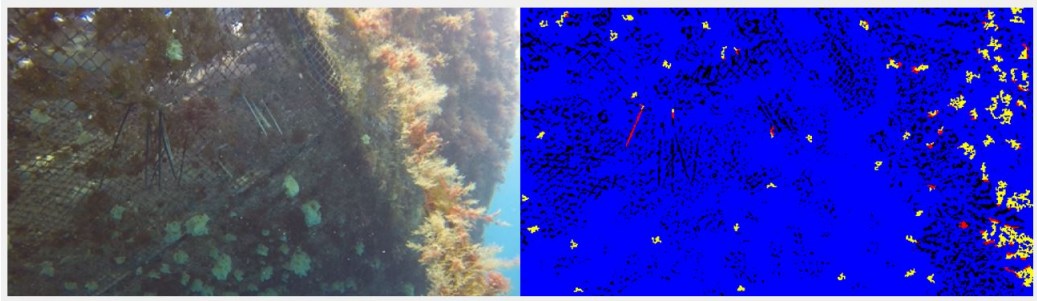


FIGURE 4.18: Frame with results from Video 2

Later, in another frame of the same video 4.19 the results are different. There are fish in the background and the system gets confused and can not recognise the existence of the defective hole. In this frame, the camera has also moved closer to the nets and the frame appears to be zoomed in.

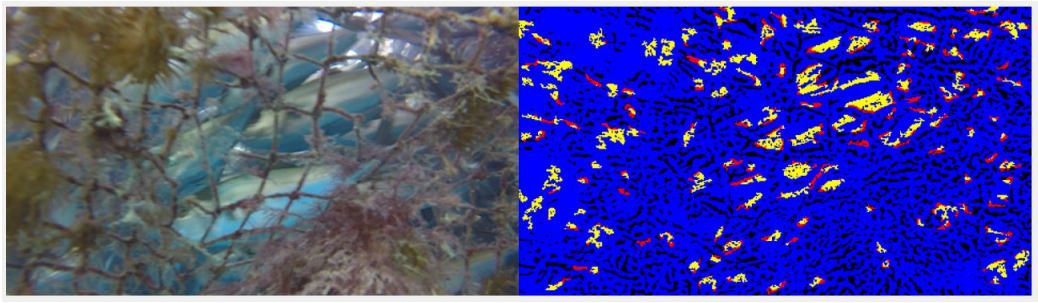


FIGURE 4.19: Frame with results from Video 2

Video 3

In this video it is clear that the system recognises many false positive holes. It also outputs some potential defective holes in the area where the aquaculture nets end and sea begins.

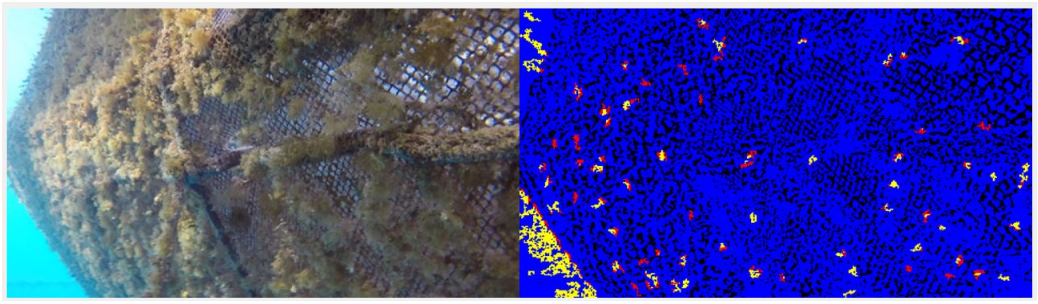


FIGURE 4.20: Frame with results from Video 3

Video 4

The algorithmic result in Video 4 does not seem that bad on each own as it appears to output only a few defective holes and assess the others as potential. However, when compared with the original colored frame, it is made clear that are all faulty and with marine fouling attached the system can not determine the correct results.

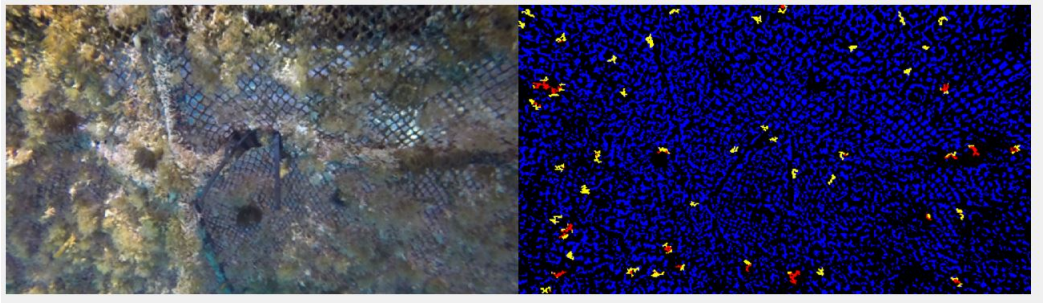


FIGURE 4.21: Frame with results from Video 4

Video 5

In Video 5 the initial results are more positive. The algorithm does not detect defective holes in the biggest part of the nets. The problem arises when the aquaculture cage outline ends and sea begins, where the system finds some defective holes and many potential defective holes.

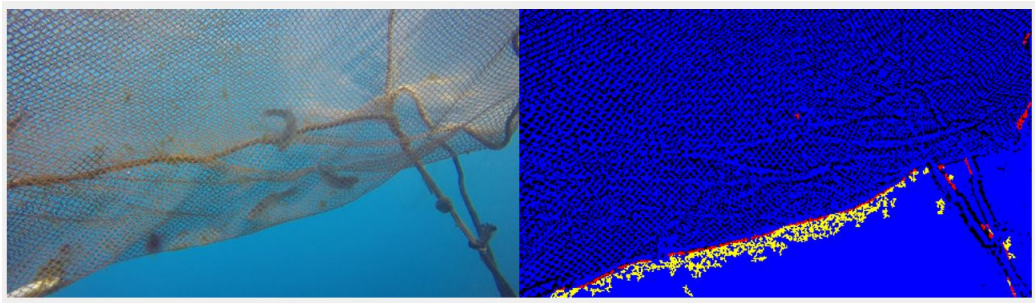


FIGURE 4.22: Frame with results from Video 5

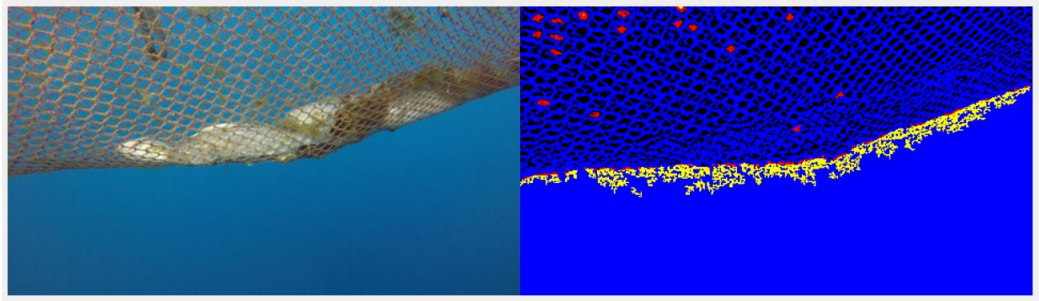


FIGURE 4.23: Frame with results from Video 5

In another frame from Video 5, shown in Figure 4.23 it is easy to observe that as the camera gets closer to the nets, the system outputs more defective holes where the normal holes of the nets are. Also in this specific situation the outcome is not affected by the background fish.

Video 6

In video 6 frame the lower part of the image is closer to the camera, and so it seems zoomed in, which makes the algorithm output faulty defective holes.

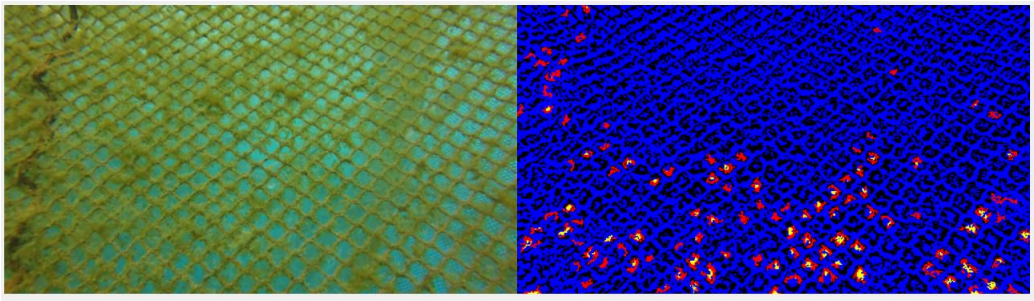


FIGURE 4.24: Frame with results from Video 6

However in another frame of the same video, the faulty defective holes are limited to the area outside the nets borders

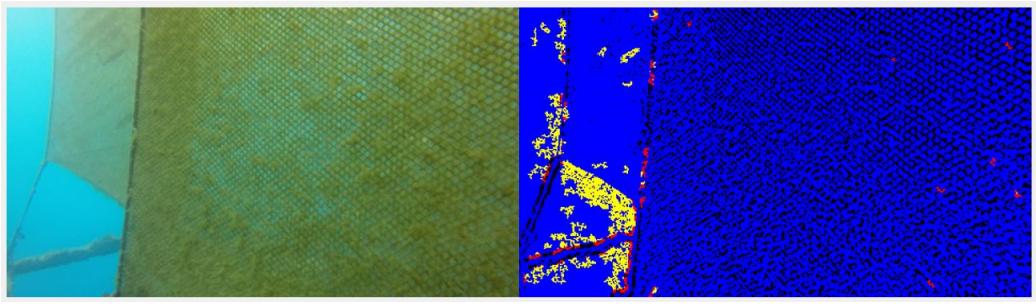


FIGURE 4.25: Frame with results from Video 6

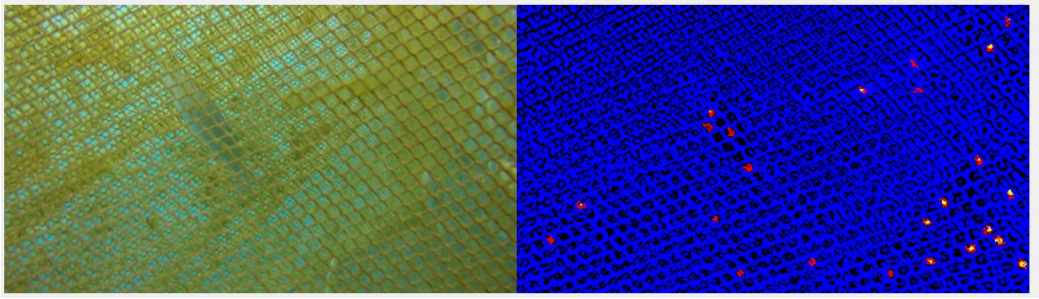


FIGURE 4.26: Frame with results from Video 6

Video 7

The same thing happens in video 7, where the results are more positive. The system recognises only a few faulty holes in the actual nets, and the rest of them near the border with the sea.

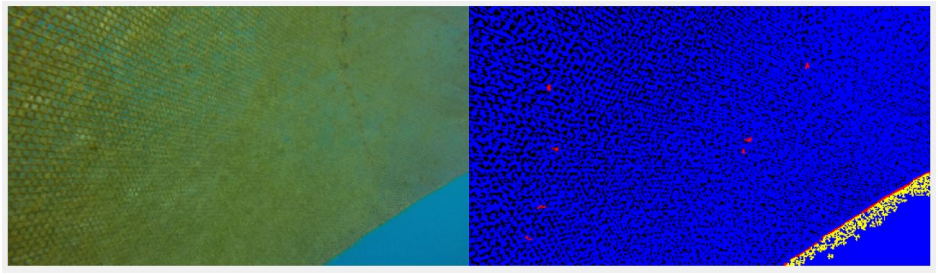


FIGURE 4.27: Frame with results from Video 7

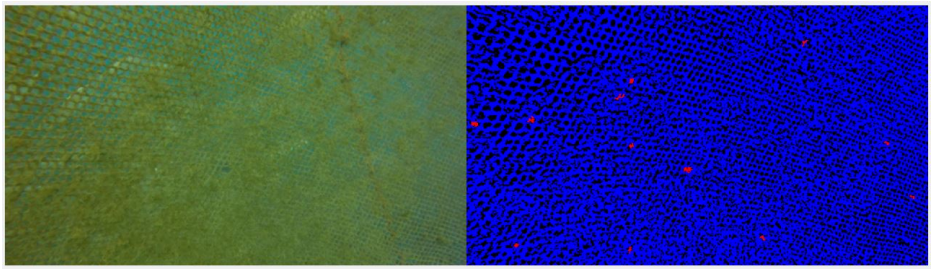


FIGURE 4.28: Frame with results from Video 7

However, later in the video there is external foreground noise - the diver's hand - in the frame, which causes problems in the performance of the algorithm as it outputs many defective holes in its place. In this specific frame there are also more false defective holes detected, as the camera has moved closer and changed the perspective.

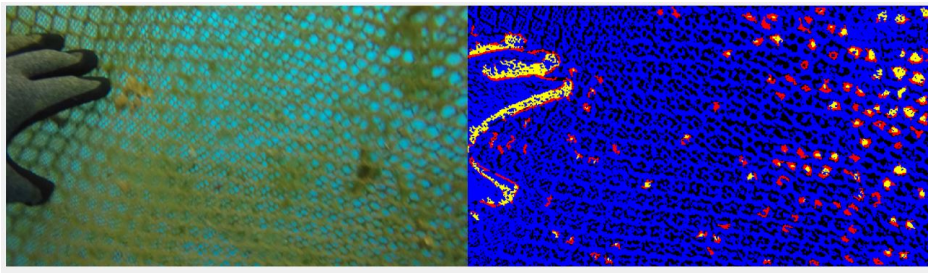


FIGURE 4.29: Frame with results from Video 7

Video 8

Video 8 responds even better in the first frames, as it shows many potential(yellow) defective holes, but does not mark them red as if they actually are.

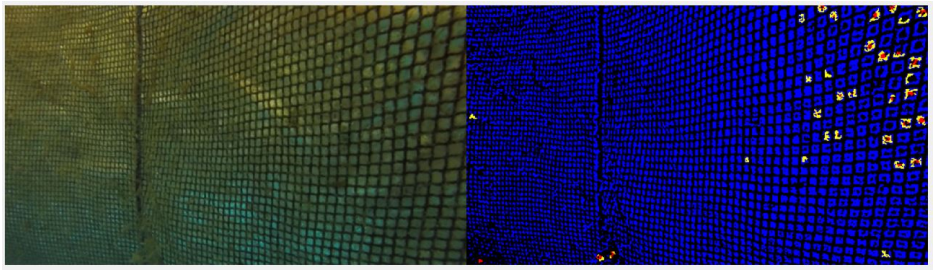


FIGURE 4.30: Frame with results from Video 8

The problem here arises later in the video where, as seen from the frames 4.31 and 4.32, the border structure and the light in the upper part of the frames, create many false positive defective holes.

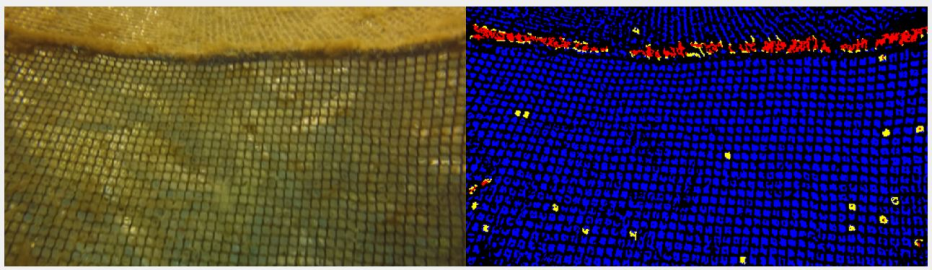


FIGURE 4.31: Frame with results from Video 8

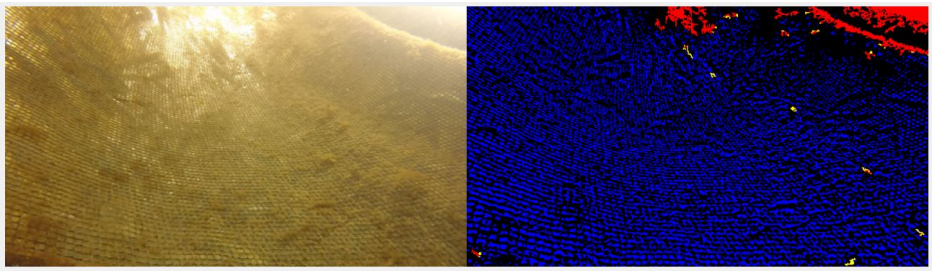


FIGURE 4.32: Frame with results from Video 8

Video 9

Similar problems appear in video 9, in the border between nets and sea, where the algorithm is not able to see it correctly.

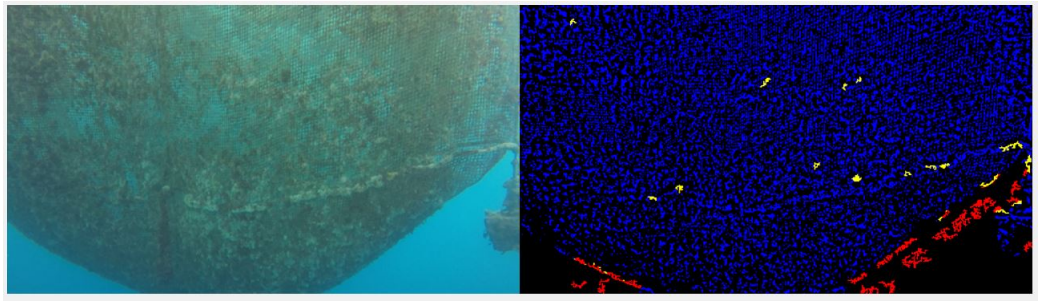


FIGURE 4.33: Frame with results from Video 9

However, the system responds excellent in the rest of the video 4.34, where there are only nets. It outputs only possible defective holes and not definite defective ones and in an environment with a medium amount of marine growth.

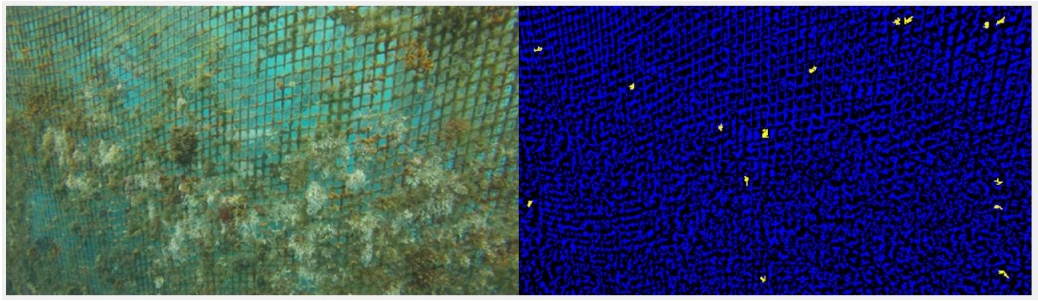


FIGURE 4.34: Frame with results from Video 9

Video 10

Video 10 also has the same problem with the sea around the nets borders. There is also a problem with lighting in this frame, as the lower part of the image appears to have holes, where in reality it is nets that are dark from the lack of light.

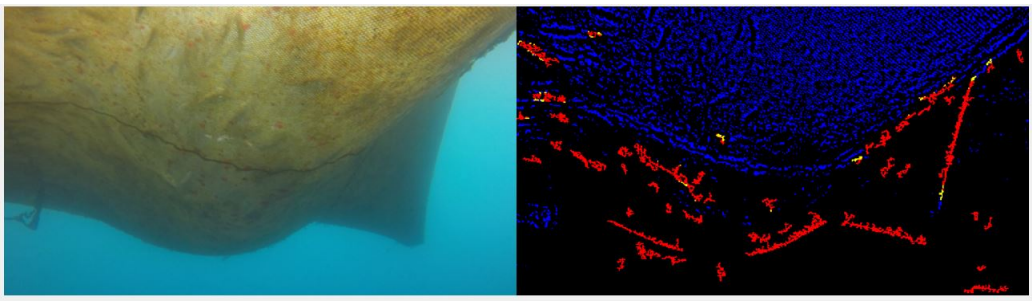


FIGURE 4.35: Frame with results from Video 10

Video 11

In this video we also have a problem that appears in some of the previous videos: the net's frame structure. This is categorised as foreground noise and responds badly as input in the existing algorithm.

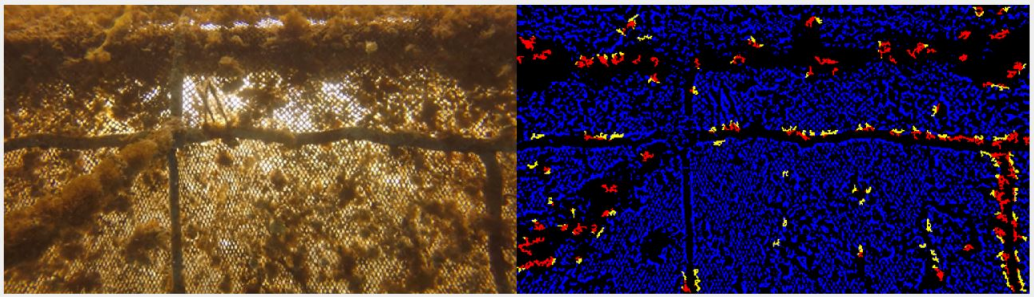


FIGURE 4.36: Frame with results from Video 11

Video 12

In the first frames of Video 12 the output is a positive image, as it detects many possible, but only a few defective holes that are wrong. The main problem in this frame is that the real defective hole is not recognised by the system. It is marked as possible defective hole and not as a real one.

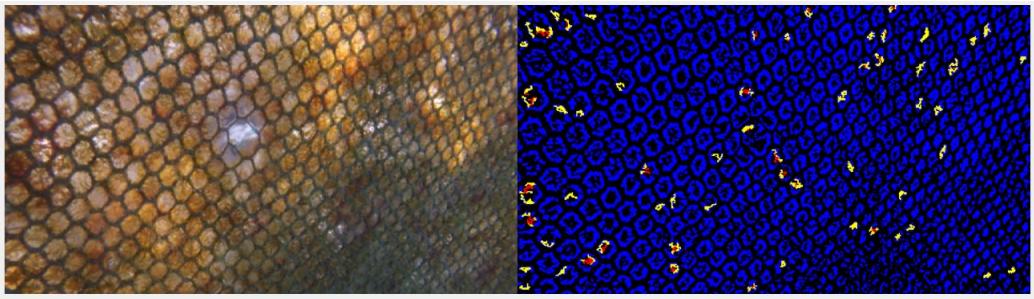


FIGURE 4.37: Frame with results from Video 12

Thankfully, in the next frames the actual defective hole is recognised, but with this achievement came more wrongly recognised holes.

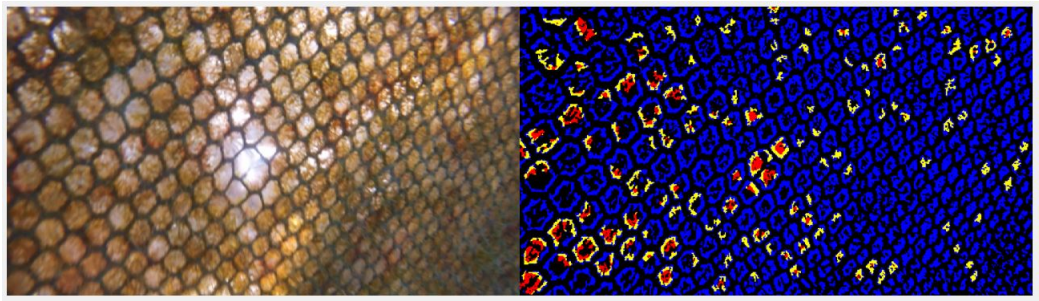


FIGURE 4.38: Frame with results from Video 12

It gets worse as the camera moves closer and the frames are zoomed in. The algorithm detects many false positive holes, when in reality they are regular holes that appear bigger due to the position of the camera.

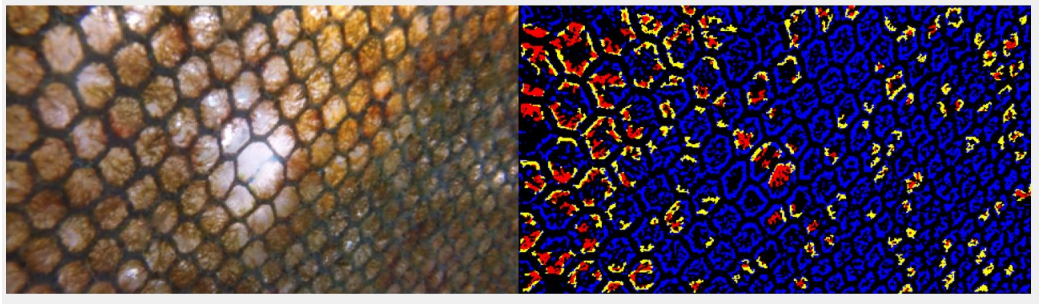


FIGURE 4.39: Frame with results from Video 12

Video 13

In the last video test, Video 13 there are some similarities with Video 12. In the the first frames the results are not so bad, as only a few are false positives, and it also recognises the real hole in the frame.

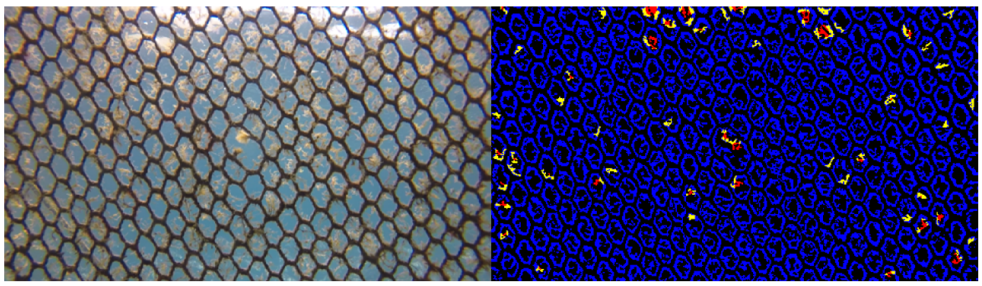


FIGURE 4.40: Frame with results from Video 13

The problem again arises when the camera changes position to get closer, and the system sees the magnified normal net holes, as defective holes.

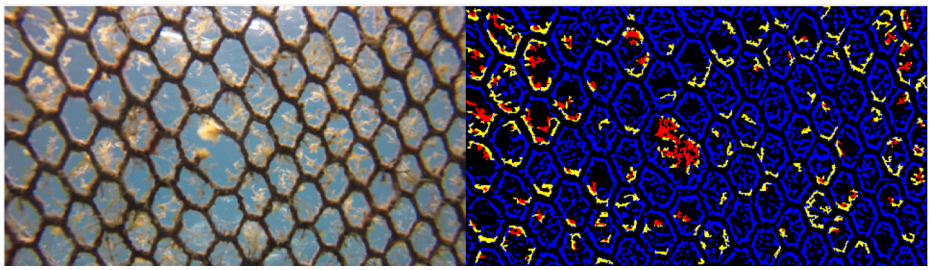


FIGURE 4.41: Frame with results from Video 13

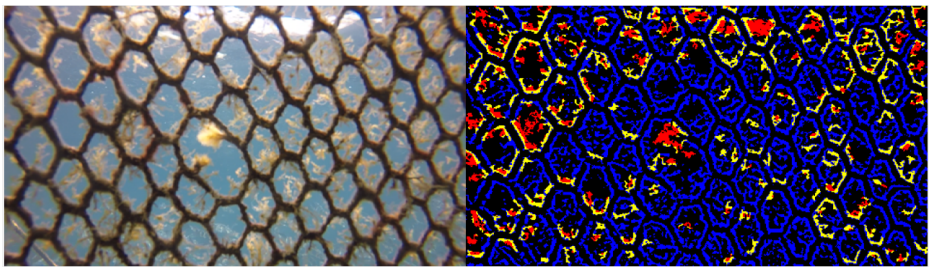


FIGURE 4.42: Frame with results from Video 13

4.3.2 Evaluation Results

The rate data were taken after sampling through the videos and calculating an average.

From every video some basic statistical data was exported which is needed for easier evaluation of the algorithm's performance on the new video data. As some of the numbers that were exported were changing with each frame, for purposes of readability and easier comparison, information was extracted from some frames found in different stages of each video. For example for Video 4, we took 6 frames smoothly distributed through the video in order to get data about the algorithmic statistics from all the different perspectives this video presents. The same thing was done for each video, getting information about the holes this algorithm outputs. Then, a success rate for each of these individual frames is calculated according to the holes the algorithm outputs as defective holes and the real existing holes that can be seen in the videos(with naked eyes). Last, there is a combination of the above to get an average success rate of each video as we can see in Table 4.2.

The 'detected holes' tab in Table 4.2 holds an average number of the holes the algorithm found in the frames, including both the real defective holes and the faulty ones. For example in video 2, the system 'sees' 83.2 holes average when in reality there is only one. For the number 83.2 to be generated, an

average calculation was done from the defective holes result found in each random sample frame from each video. Specifically, the defective holes results of each sample frame of a Video V (V=0,1,...,13) were added and then divided by the number of the sample frames taken from Video V.

video	duration (sec)	real holes holes	average detected holes	average success rate(%)
v0	72	2	2	100
v1	24	1	7.5	19.8
v2	38	1	83.2	1.79
v3	33	1	91.8	1.45
v4	11	0	76.6	1.72
v5	40	0	49.8	3.29
v6	38	1	50.2	7.52
v7	30	1	50.5	4.45
v8	24	0	55.6	2.86
v9	64	0	36.5	4.15
v10	50	0	36.6	2.92
v11	21	1	157.8	0.673
v12	8	1	164.8	0.752
v13	9	1	127.5	1.13

TABLE 4.2: Evaluation Results for new videos (v0 is the reference video from Th. Zacheilas' Thesis.

Video 0 is separated from the rest, as it is the video that the whole previous system was based and build on. As the previous system was build based on the specific environmental characteristics of of Video 0 and was only tested with that as input, it is reasonable to have a 100% success rate. That is not the case for the new videos though, that output an incredibly low success rate.

There is an important note to be made about the success rates calculation. In many of the videos there is exactly one real defective hole in the nets while in some others the nets seem intact, which means that either there is no defective hole, or it is very difficult to determine that due to difficulties in vision. The target output number of the detected defective holes should be zero, and yet in most cases the algorithm outputs a larger number, and as a result the success rate should be zero as well. In that way if the number of defective holes detected is decreased and closer to zero(which is the target), the progress of the detection cannot be seen. In order to show that progress of the algorithmic results - especially for the results in chapter 5 - , when the actual defective hole number is zero, it is assumed to be one. For example if a Video V has 0 real defective holes, and detects also 0, then the algorithm is

100% successful. If in the same Video V detects 50 defective holes, then the algorithm has 0% success and if it decreases that number to 25 defective holes it still has 0% success in the detection. However when assumed that there is at least 1 real defective hole, then for 50 detected holes the rate would be 1/50=2% and when it improves to 25 holes detected, its has 1/25=4%, which shows a 2% increase in the detection and the improvement the algorithm made.

4.4 Video Data Analysis Conclusions

From Table 4.2 it is easy to conclude that the algorithm does not respond well in the new video data. There are lower success rates (a bigger error) in the last 2 videos which for now can only be assumed that this may be due to the camera being closer to the nets as if it is zoomed.

There are some safe observations to be made from the above data introduction. First, the acknowledgment of the increased difficulty to detect defective holes in some of the new data even with naked eyes. In some videos, marine growth makes the net lines hide and the nets are no longer in a pattern. Marine growth also changes the color and the illumination of some videos and so it does make it difficult for the existing threshold to have good results. Also in other videos it is not clear where the edges of the nets end and the background sea begins.

4.5 Problems to be Solved

In order to raise the success rates higher, there are some problems that need to be solved. The fact that all these videos are a lot different, creates the problem of seeing the similarities, compare them and put them in categories with common features as we did in the beginning of this chapter in Table 4.1.

Some basic problems of the system response are:
1. The color of the nets versus the background.
2. The borderline of the aquaculture cages where the nets end and there is a solid sea background area.
3. Movement and position of the camera. Whether the camera goes fast and there are blurs or it gets too close to the nets as if zoomed in, the system can not readjust and recognise the correct defective holes.
4. Marine growth attached on the nets.

In the next chapter some of these problems are solved and in some there is significant progress towards the solution for the best outcome.

Chapter 5

Modeling of the System and Progression

In this chapter there is a progression of the existing algorithm towards a more advanced system and suitable adjustments according to each video, trying to find changes that benefit all or some of the videos. Here there is a presentation of some algorithmic variations and improvements according to the needs of the new dataset.

5.1 Frame and Video Classification

Image classification is the process of categorizing and labeling groups of pixels or vectors within an image based on specific rules. Video classification is the same in a way that when a video is classified, some frames of parts of the frames are taken into consideration to extract a label for the whole video. From the analysis in the previous chapter, we have already established some basic categorization for the video data used in this thesis, shown in Table 4.1.

From Chapter 4, it is concluded that even though the new videos are more complex, with more noise and generally difficult to process, they all have some things in common. One of them is the color of the nets. In all the new video data, the nets are always darker, in black or brown colors, than the background which is usually a light blue or green depending on the lighting and the nets. So, our first attempt is to classify the videos in two categories: white net and the black net videos. This distinction is crucial as the algorithm used to detect the defective holes takes into consideration the color of the nets.

As shown in Table 4.1, the videos were parted in more than one categories. Another one of the classification categories is made by the fact that some of

the videos are zooming in on the nets and the holes are much more visible. For example video 12 and video 13. Also, one more category was made according to the clearness of the nets and the water. Marine fouling attached on the nets plays an important role in the system's response needs to be considered.

The reason for creating those categories is no other than to make the processing easier. putting more than one videos in the same category, means that they have some characteristics that look alike, or that they respond to the system in the same way and can be processed in similar ways. Separating the videos and or parts of them in categories, means that the problems of each category can be solved separately and then apply the solutions to the other same-category videos.

5.1.1 Color of the Nets Category and Identification

As was established above, the first category-video label, was made according to the color of the nets each video has, whether it is due to lighting or to other factors such as marine fouling.

In the original video (Video 0) the nets are white while in all of our new videos the nets are black. That causes a problem in the algorithm as shown in the results of the videos in the previous chapter, and since the new videos have black nets, the nets are recognised as defective holes(5.1). The reason for this problem is that the algorithm recognising the defective holes (Connecting Component Labeling algorithm) takes as input the fact that the nets are white and the holes black.

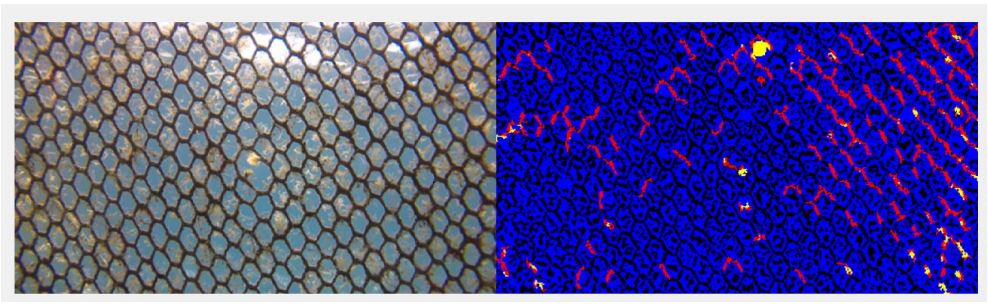


FIGURE 5.1: Frame with results from Video 13
Faulty dark nets detection

When the input is changed, considering that the nets are black and not white the results appear in Figure 5.2. We do not necessarily have better results as a number of detected defective holes, but is is correct now in the sense of logic.

The algorithm detects the defective holes in the real holes of the nets and not on them which is a significant progress on the algorithmic performance.

That was done by changing the input image in the CCL function. This function uses a grayscale frame of the video being processed that is an output of an enhancement method and then it is thresholded to create a binary frame. To change the output, the complement of that binary frame was taken and used from the CCL algorithm.

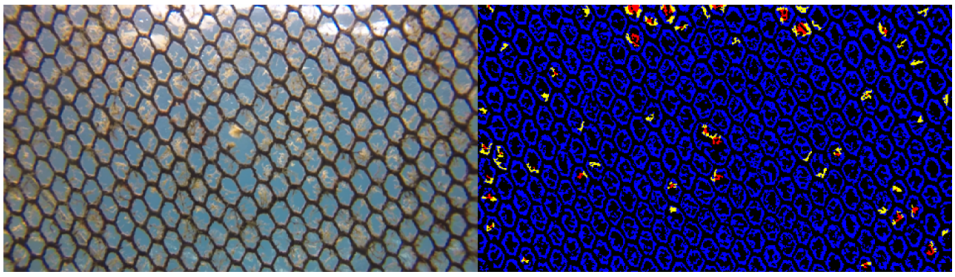


FIGURE 5.2: Frame with results from Video 13
Correct holes detection

As the purpose of this project is the creation of an automated system, there was a need for this task to be done autonomously. So first, the system needs to identify if the nets of each are black or white and give that information back in order to keep the given frame or change it to it's complement accordingly.

This is the first category/classification and it is something that can be done just by the first frame of each video.(so it will be cheap- as we check only one frame. We can also make a variation that checks the video every few frames as some of the videos have complex enough lighting that might change through the video, or have different lighting in the same frame that outputs an image with black nets on one side and white nets in the other. This happens only one time in our system, but it should be considered as a variation for future realistic video models)

When considering a way to output a label for a whole video into Black Net Video or White Net Video, there were some attempts made to find an easy way to label them. The first experimentation was based on the thought that the nets would take less space than the background - whether it be solid sea or just the background of the holes. So after the frame is being converted to binary and before it goes to the CCL function, we counted the number of

black and number of white pixels in the frames. This attempt did not work. In most of the scenarios, the content ratio was about 40%-60% of either color and so the results were ambiguous. If the nets are slim and there is no marine fouling and generally the conditions are ideal, that would give good results as the nets take 'less' space in a frame. But the conditions on our new video data are not ideal and the reality is more complex than that. Marine fouling also played some part in the complexity of the outcome as well as the angle and position of the camera.

Flood Fill and Variation

When proceeding to experiment with another way of recognising the color, there are better results using a variation of the Flood Fill algorithm . The basic logic is that as flood fill actually fills a specific colored area with the color of your choosing, it goes through that area until all the same colored pixels are changed to the new color. When we apply this in a aquaculture nets frame, the pixels that we choose would be either on the nets or on the background area. When the pixel choice is in one of the holes, the flood fill algorithm will fill only this hole with the color choice (Figure 5.3). But when the pixel choice is on the nets, the algorithm will fill the nets in the hole frame as they are connected (Figure 5.4). This determines that when the pixel choice is on the nets it would follow a much bigger and slower path as it goes through more loops to cover the net area.

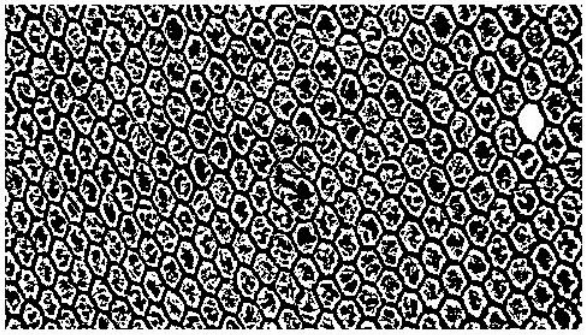


FIGURE 5.3: Flood Fill One pixel Choice in Background

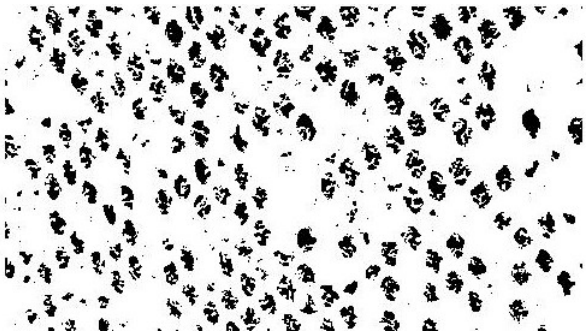


FIGURE 5.4: Flood Fill One pixel Choice on Nets

For the purposes of this thesis, the process of loop iteration for a pixel choice on the nets is what is important. {The general idea is that when you count the iterations that happen to fill a chosen area, the count value in the background holes areas will always be smaller than that of the net areas.}

The Flood Fill variation being used(introduced) in this thesis has a different perspective, as the space does not actually need to be filled with a new color or to have a specific space determined as a result like the original flood fill algorithm. The main goal is to get the 'longer' paths filled that logically have only the boundary of the image. These longer paths will represent the nets while the short ones will be the holes. Contrary to the classic flood fill algorithm, we do not use recursion and the variation is based on a non-recursive 4-way method with a stack. The variation was made for the purpose of being more robust. The original algorithm would work excellent as well.

A function was created for the purposes of identifying whether the nets are dark or not using the flood fill algorithm. This function takes as input a binary frame, that the adaptive_threshold process outputs, and then is used in the CCL algorithm. The first thing the floodfill function does is to select 1% of the total frame pixels at random in order to represent enough covered areas but not too many so that it takes a lot of time to process them all and make a conclusion. Specifically, as the frame's size at this point is 270x480(for the videos that were initially 1080x1920) the total number of pixels is 129600 pixels and one percent of them is equal to 1296 pixels. For each one of those the floodfill algorithm is applied and starts to fill the areas of each pixel. For every repetition of a pixel fill there is a counter -countFlood that counts the amount of times the while loop takes, in order for the area to fill. As we specified above, the longest paths will be of the nets areas while the shorter will be that of holes.

However, as the new video data are very complex, some extra things were done in order to be safe about the net's color decision. While taking into consideration that many frames cover some continuous areas with background sea, and others have marine fouling noise, a simple maximum value of the countFlood array could be the wrong output for the frame. That is why getting the result not only of one max value, but of the color that appears the most among the max countFlood values, is optimal. This also solves the problem of when the nets appear thin and a 4-way flood fill does not go as far as the 8-way version would. In this case, many paths are considered and from the tests that were done it always gets the correct results.

Algorithm 1 : Flood Fill Variation

```
onePercentPixelnum ← numPixelFrame/100
randomPixelArray ← randi(pixels)
for onePercentPixelnum do
    currentPixel ← randomPixelArray(i)
    oldColor ← currentPixel
    while exitflag == 0 And pixelinimage do
        if northPixel is oldColor then
            Set new pixel north
        else if eastPixelisoldColor then
            Set new pixel east
        else if southPixelisoldColor then
            Set new pixel south
        else if westPixelisoldColor then
            Set new pixel west
        else
            exitflag to go back to start
        end if
        countFlood ← countFlood + 1
    end while
    countFloodArray(i) ← countFlood
    oldColor(i) ← oldColor
end for
maxCountFlood ← max(countFloodArray)
if maxCountFlood.oldColor == 0 then
    The Nets are black
else
    The Nets are white
end if
```

Even though the initial testing of the flood fill algorithm was done with Matlab, the main reason for not using the original recursive method of flood fill is

because the xilinx Vivado HLS tools that will be used for the FPGA architecture design do not support recursion. So we designed the algorithm without recursion. However, this also makes the algorithm faster. And as it is not only for one pixel but for 1296 at a time, this more robust version is better.

5.1.2 Marine Fouling Category

A basic problem that arises with the new video data, is the appearance of marine fouling or marine growth attached on the nets. While Video 0 is clean and clear, many of the other videos have low performance due to marine fouling noise which alters basic features of each frame and makes the process of detecting defective holes even more difficult. And that is why a whole category of them was made to label the videos according to the concentration of marine growth attached on the nets.

The goal is to determine if an aquaculture nets video has marine growth on the nets and if it has, how much it is, and measure it with a low, medium or high factor.

A future proposed task would be to find a way to reduce it or even remove it from the frames, in order to make the net's structure clear and visible with the purpose of getting better results in the defective holes detection, or even reducing the extent of the system.

From the evaluation we did in all the test videos in chapter 4, we observe that the videos with marine growth, whether its concentration is high or low, they have some things in common like the color.

5.1.3 Zoom - Camera Positioning and Points of View

Another category made was the Zoom category, which refers to the point of view the videos were taken. It was established as a category, after seeing the results of the system response in chapter 4, as there were many parts of the videos that had a problem with the output when the position of the camera was closer to the nets or it was changing through the video.

This category is essential and the next section's results can be used to significantly improve some of the category's problems.

5.2 Further than Image Classification - Parameters Configuration

Categorizing the videos and changing the nets from white to black is simple and effective but not enough. As we can see from Figure 5.2 even though we changed the colour of the nets, the results are not as positive as we want. The output recognises a lot of faulty holes as in holes that do not really exist.

In the pre-existing algorithm there were some parameters set specifically for the needs of Video 0. The values of these parameters need to be tested and if needed, to readjust for the new video data.

5.2.1 Testing Video 13

Video 13 has many differences with Video 0, but it has a clear structure and is easy to spot when the results are better and that is why it was the choice as the first test subject. Video 13 has 483 frames in total, so the sampling that was done for the video was every 80 frames and get the frames 80, 160, 240, 320, 400 and 480. As some of them have similarities in the test results we show only the frames 160 and 400 from every test to simplify the imaging comparisons and quick assessment of the modification results. The complete numerical results are summed up in Tables after every test made and show the number of defective holes the system recognises as well as the success rate calculated the same way with the evaluation success rates in Chapter 4. As in Video 13 we only have one real defective hole in the nets, the results are better in the cases where the result is closer to one and worse when the defective hole count of the system is larger.

In Figure 5.5 are sampled frames with the original parameter set: offset=50, luminance=0.03, threshold window=9.

The first change to be made, is the offset parameter. It was initially set as 50 which refers to the window size (50x50) and is used to sample the frame after the CCL algorithm in order to detect and define the defective holes.

We first set the values to be one smaller and one larger than the initial and see how the system responds with the change so we can move further accordingly.

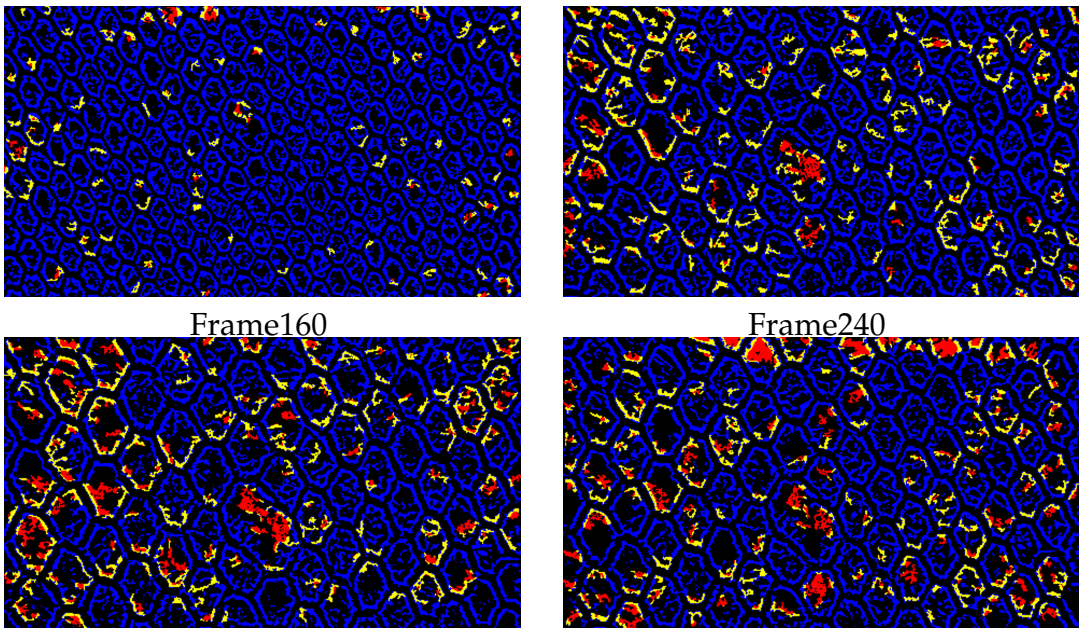


FIGURE 5.5: Frames with results from Video 13 with original settings offset=50, luminance=0.03, thresholdWindow=9

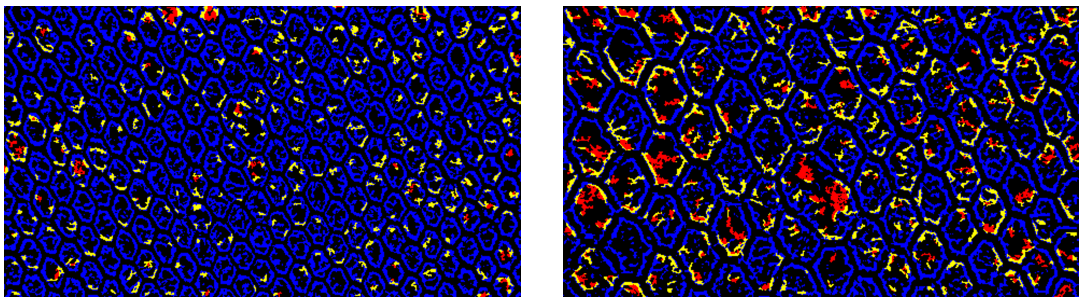


FIGURE 5.6: Frames with results from Video 13 with offset=10

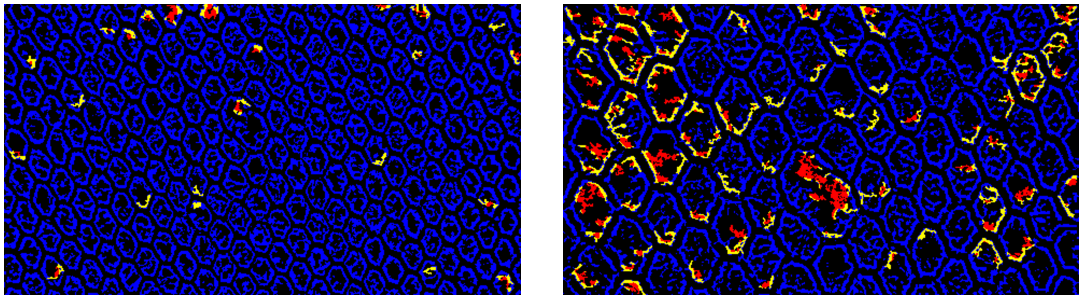
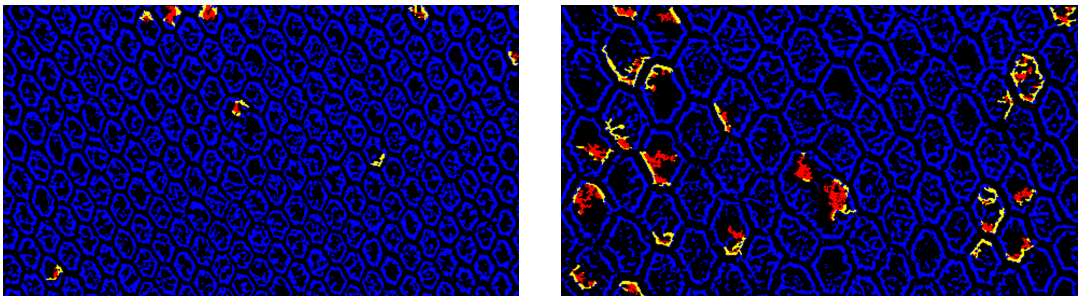


FIGURE 5.7: Frames with results from Video 13 with offset=100

As we can see from figures 5.6 and 5.7, the frames from the video that were exported when the offset is set to 100 have better results, than those with

offset set to 10. We continue to set the parameter higher until we set it to 200(no more because of frame size) and we see the results in figure 5.8.



Frame160 Frame400

FIGURE 5.8: Frames with results from Video 13 with offset=200

In Table 5.1, we can clearly see that when the offset is higher, the false positive results are fewer. If we combine the statistical results with the visual, we can see that the real defective hole is marked as well and so the algorithm does improve a lot with this modification.

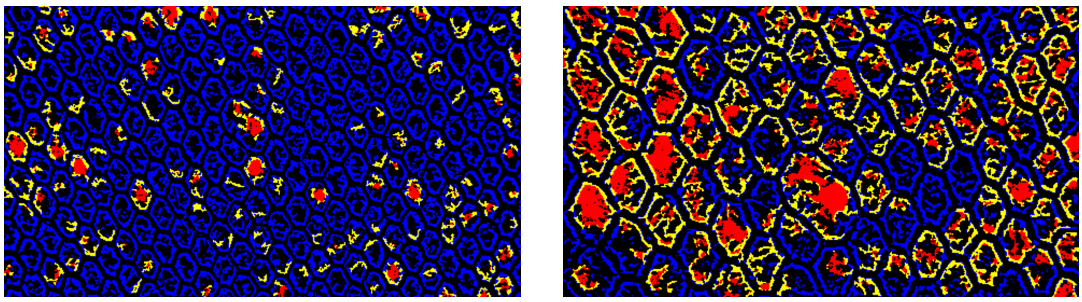
offset	10	50	100	200
frame80	297	52	15	9
frame160	264	84	23	8
frame240	312	148	60	21
frame320	280	151	73	26
frame400	260	162	94	25
frame480	283	168	65	31
success rate	0.4%	1.1%	3.3%	7.9%

TABLE 5.1: Offset parameter in Video 13 frames

Another parameter that could be adapted is the illumination threshold set initially to the value of 0.03. All the video data have differences in lighting and illumination, and all the new ones differentiate a lot with the originally tested Video 0 in these areas, so we need to test how a change in this parameter could help or not in improving the results.

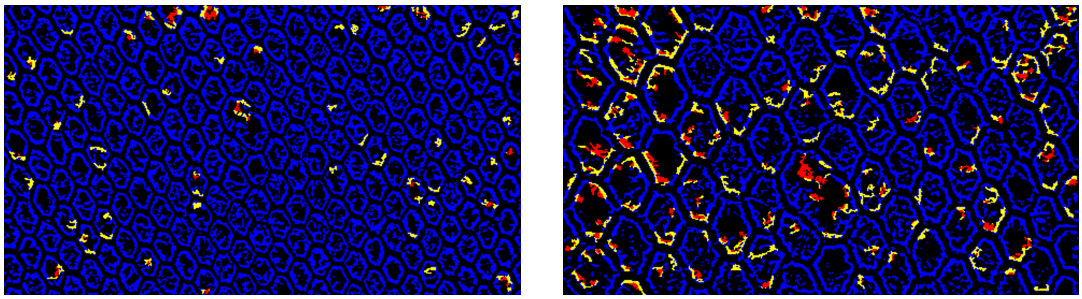
The frame results after testing Video 13 with the original set parameters is shown in Figure 5.5. Continuously, while keeping the offset parameter steady, the illumination threshold alternates between values below and above the initial 0.03.

We begun testing with a higher value set to 0.1 and then a lower set to 0.01. The results shown in Figure 5.10 and Figure 5.10 correspondingly, seem to be better in lower values. So we set a value in between, and see the results in Figure 5.11



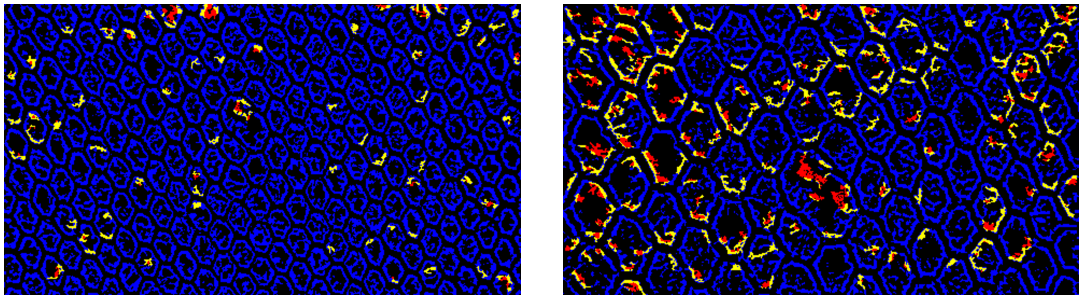
Frame160 Frame400

FIGURE 5.9: Frames with results from Video 13 with luminance=0.10 and offset=50



Frame160 Frame400

FIGURE 5.10: Frames with results from Video 13 with luminance=0.01 and offset=50



Frame160 Frame400

FIGURE 5.11: Frames with results from Video 13 with luminance=0.02 and offset=50

As it is clearly seen from the above testing, when the illumination threshold value is lower, the results are better and the following Table 5.2 confirms those results.

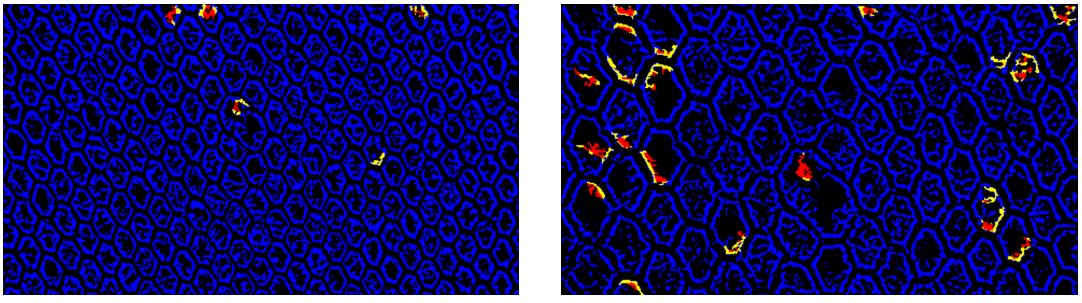
Illumination Threshold	0.005	0.01	0.02	0.03	0.07	0.1
frame80	47	49	53	52	79	99
frame160	54	57	64	84	98	127
frame240	122	124	145	148	177	170
frame320	141	139	148	151	172	159
frame400	129	141	149	162	170	166
frame480	117	126	130	168	157	140
success rate	1.4%	1.3%	1.2%	1.1%	0.9%	0.8%

TABLE 5.2: Illumination parameter in Video 13 frames with default offset=50

The previous testing of the offset parameter(5.1) shows that the algorithm gives best results with offset=200, so the next step is to test the illumination threshold parameter for that optimal offset.

Illumination Threshold	0.005	0.01	0.02	0.03	0.07	0.1
frame80	5	7	9	9	12	21
frame160	5	5	7	8	21	25
frame240	13	14	13	21	39	58
frame320	14	15	15	26	53	61
frame400	21	21	22	25	43	72
frame480	19	22	23	31	51	60
success rate	12.9%	11.5%	9.7%	7.9%	4.4%	3%

TABLE 5.3: Illumination parameter in Video 13 frames given offset=200



Frame160 Frame400

FIGURE 5.12: Frame with results from Video 13 with luminance=001 and offset=200

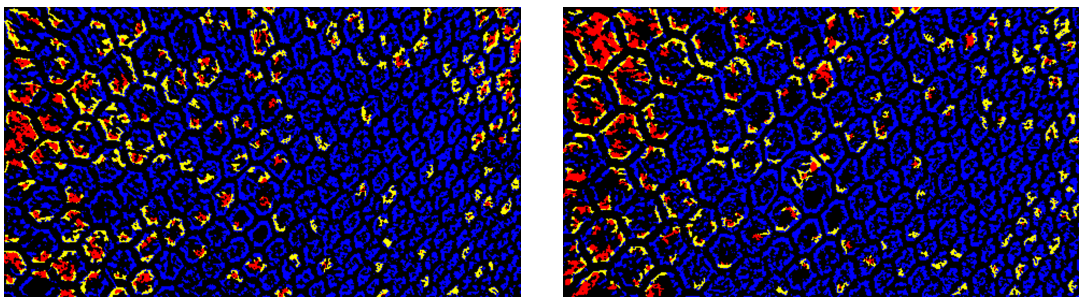
If the focus is only on the success rate data, the answer for the optimal value would be to set the illumination threshold to 0.005. But when the visual result is also taken into consideration, We can see that the defective hole is barely detectable and maybe it would be better to have a bigger value of the threshold in order to have the one correct hole detected and find solution for the rest in another way.

5.2.2 Testing the Remaining Videos

The rest of the video data were also tested for changes with the offset parameter with the purpose of seeing improvement and finding a possible common value that would be optimal for many of them. For that reason, the tests were made with the same values as those of Video 13's testing. In this section, some of the results of the testing are presented both as statistical success ratings as well as visual frame results. The testing was executed in the same way as before, sampling the videos with frames harmoniously distributed through each video. For purposes of readability and easier comparison, only a few frame results of each video are presented.

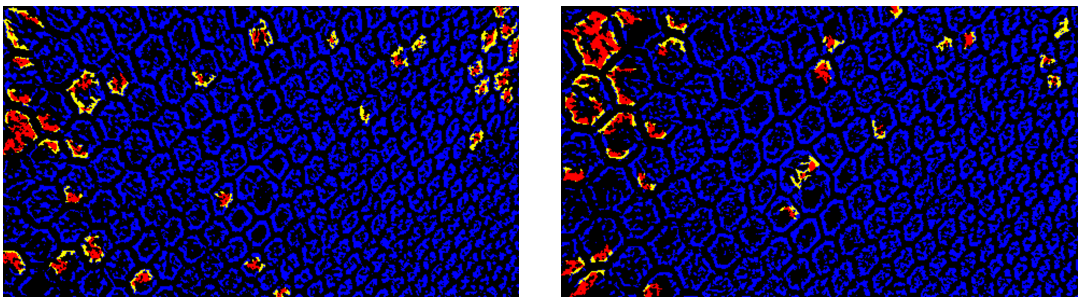
Testing Video 12

Video 12 is similar with Video 13 in terms of perspective and point of view. And that may be the reason why it responds to the algorithm similarly. As the camera goes closer to the nets, more faulty defective holes are detected as seen in Figure 5.13.



Frame140 Frame350

FIGURE 5.13: Frames with results from Video 12 with offset=50



Frame140 Frame350

FIGURE 5.14: Frames with results from Video 12 with offset=200

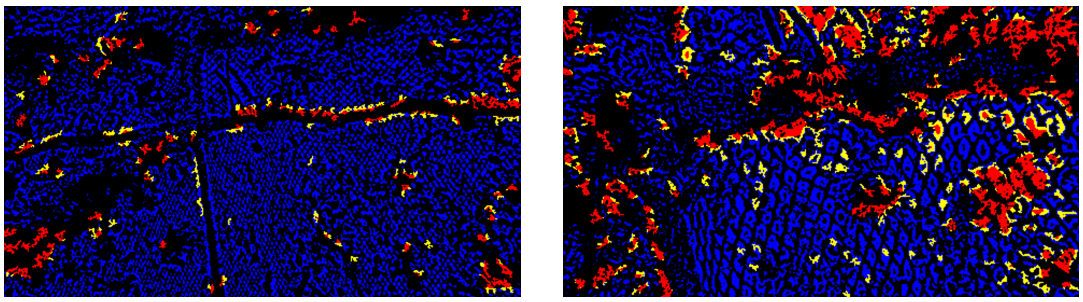
offset	10	50	100	200
frame70	439	113	38	9
frame140	392	172	72	36
frame210	404	190	107	48
frame280	392	185	98	41
frame350	463	180	80	28
frame420	371	149	78	42
success rate	0.2%	0.7%	1.7%	4.8%

TABLE 5.4: Offset parameter in Video 12 frames

The offset success rate shows some improvement and visually as the offset increases (to 200) there is a reduction of faulty detective holes that can be seen in Figure 5.14 for the same frames. However, the statistical data alone are not acceptable as in frame 140, the real defective hole is not detected at all.

Testing Video 11

Video 11 is one of the lower success rated video of the video data, along with Video 12 and Video 13. The reason for that is because it has a lot of cage borders as in foreground noise and also the perspective of the camera changes through the video and appears zoomed in in later frames. As seen from the evaluation on Chapter 4, both of these elements contribute in more faulty detection.



Frame179 Frame537

FIGURE 5.15: Frames with results from Video 11 with offset=50

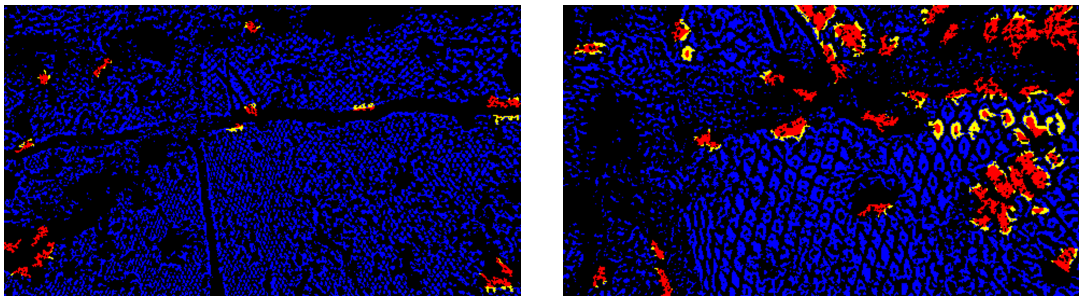


FIGURE 5.16: Frames with results from Video 11 with offset=200

offset	10	50	100	200
frame179	295	106	48	17
frame358	331	124	49	18
frame537	303	174	96	44
frame716	240	187	117	65
frame895	244	198	141	51
success rate	0.3%	0.6%	1.3%	3.4%

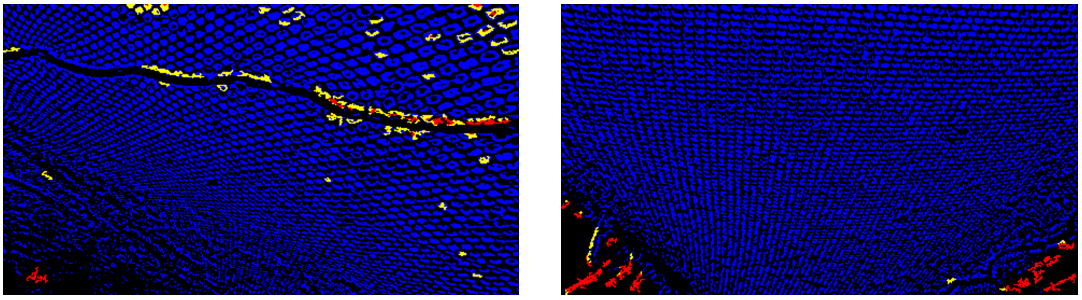
TABLE 5.5: Offset parameter in Video 11 frames

Changing the offset parameter does actually give higher success rate, however the visual results strongly suggest that there should be another way to minimise output errors.

Testing Video 10

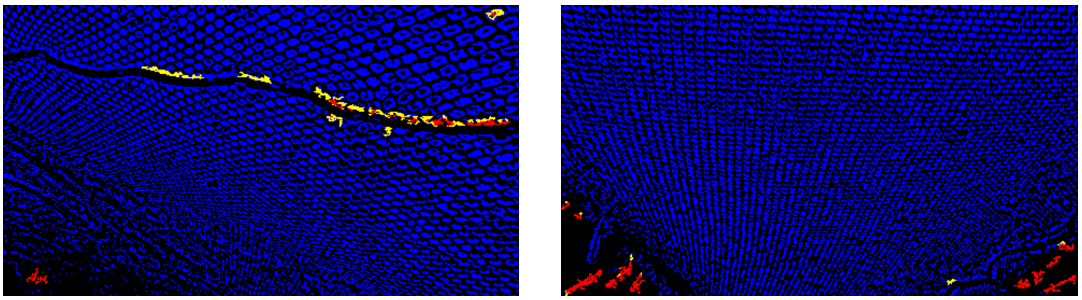
Contrary to the previous video, at first glance, Video 10 has better visual output and of course the success rate data confirm that. The few faulty detected

holes are centered around the cage borders areas and background sea. All the other ones initially detected in the net area, they improve as the offset parameter increases.



Frame840 Frame2100

FIGURE 5.17: Frames with results from Video 10 with offset=50



Frame840 Frame2100

FIGURE 5.18: Frames with results from Video 10 with offset=200

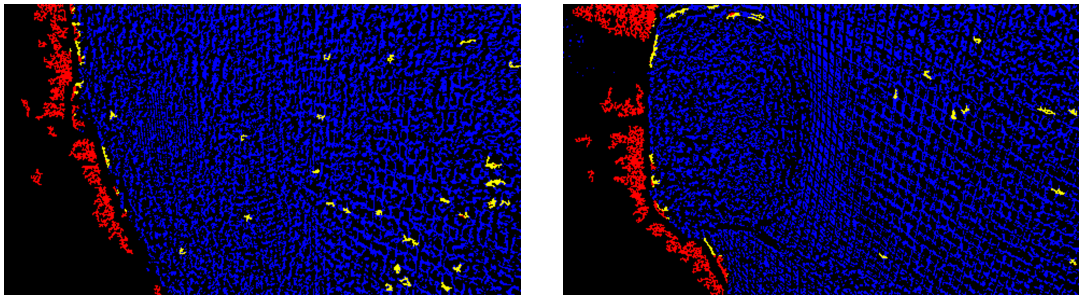
offset	10	50	100	200
frame420	185	49	27	21
frame840	132	49	22	15
frame1260	156	39	31	25
frame1680	125	47	21	20
frame2100	79	19	12	12
frame2520	79	22	12	9
success rate	0.8%	3.7%	6.5%	7.9%

TABLE 5.6: Offset parameter in Video 10 frames

Testing Video 9

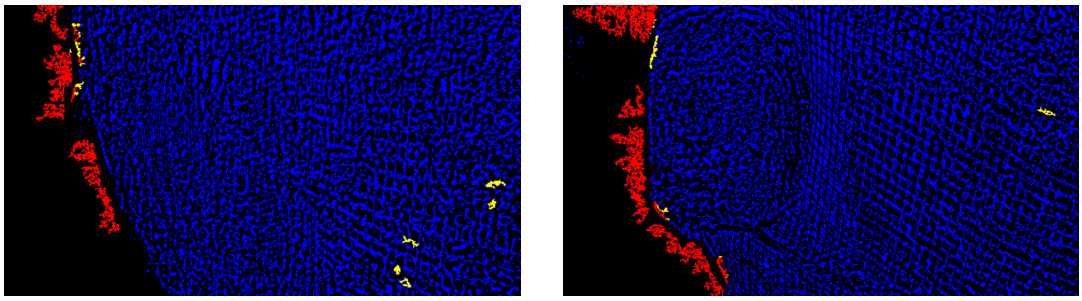
After tested Video 9 for different offset values, that Video 9 respond in similar way to the previous Video 10. The main difference, and thus why it has

bigger increase in the rates, is that many of the wrongly detected holes came from movement and not a foreground object. The movement errors are eliminated as the offset variable increases.



Frame1078 Frame2695

FIGURE 5.19: Frames with results from Video 9 with offset=50



Frame1078 Frame2695

FIGURE 5.20: Frames with results from Video 9 with offset=200

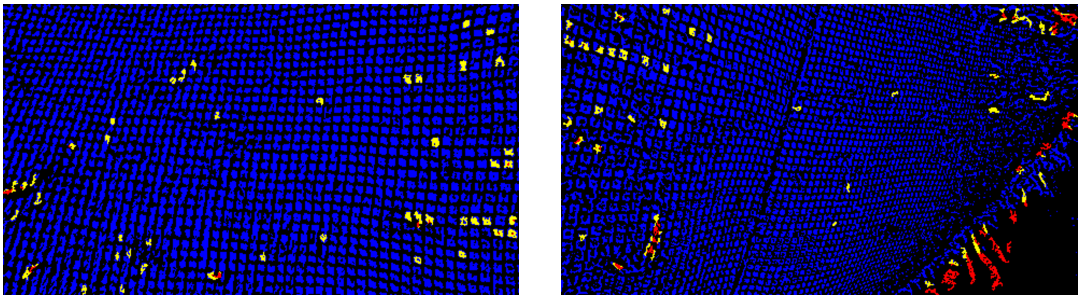
offset	10	50	100	200
frame539	182	12	2	2
frame1078	287	53	26	17
frame1617	265	47	21	15
frame2156	265	37	18	11
frame2695	167	37	19	11
frame3234	200	33	23	13
success rate	0.5%	4.1%	14.7%	17.6%

TABLE 5.7: Offset parameter in Video 9 frames

Someone could say that without the background sea in the left part of the frame, the results would be close to perfect in this video, as the algorithm only detects a few uncertain defective holes. For example the frame2695 in Figure 5.20 with offset set to 200 has only one uncertain defective hole detected in the net area.

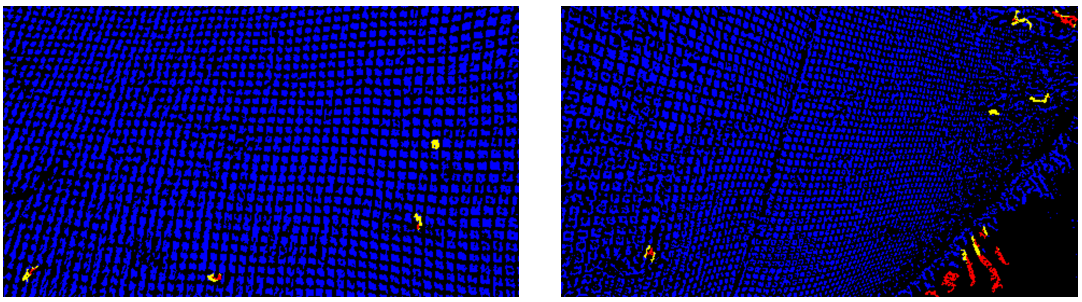
Testing Video 8

In the first frames Video 8 has already good results considering that most of the defective holes shown, are yellow as the algorithm is uncertain. However later the small part that is the background sea, outside of the nets gives some faulty detection and lowers the success rates.



Frame404 Frame1010

FIGURE 5.21: Frames with results from Video 8 with offset=50



Frame404 Frame1010

FIGURE 5.22: Frames with results from Video 8 with offset=200

offset	10	50	100	200
frame202	296	17	8	7
frame404	373	45	10	4
frame606	337	51	25	14
frame808	328	69	27	17
frame1010	355	56	22	12
frame1212	368	96	33	20
success rate	0.3%	2.8%	7.5%	13.1%

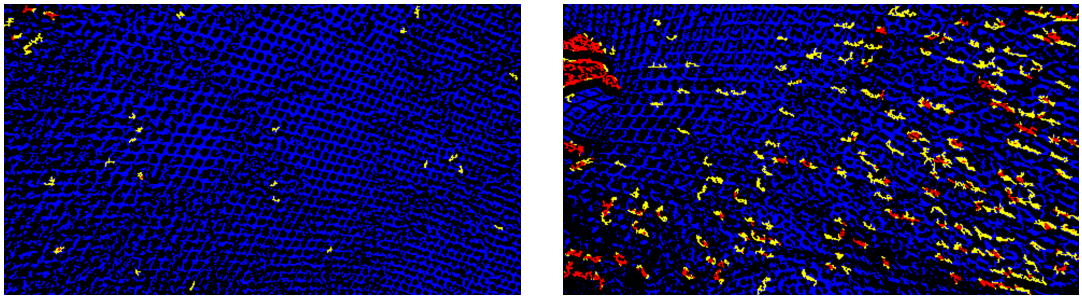
TABLE 5.8: Offset parameter in Video 8 frames

The increase in the rates are significant and the visual results corroborate in the that fact. The faulty results that came from the movement, the perspective

and the zoom, are decreased significantly, which both the statistical and the visual data confirm.

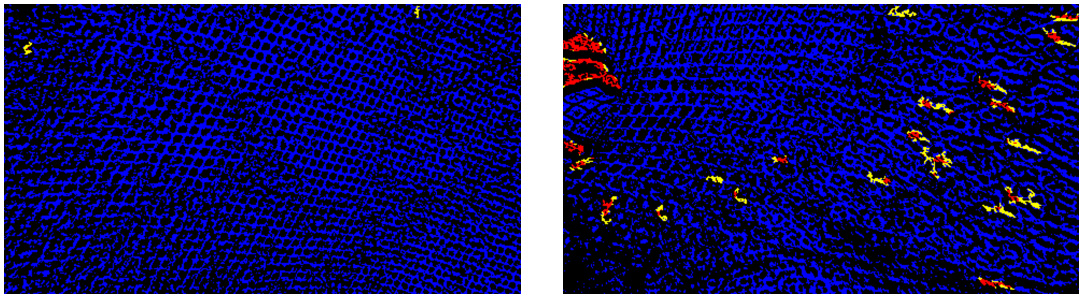
Testing Video 7

The next video tested, Video 7 has generally good result already, as in the beginning the video responds the same way as frame 500. Only in the latest frames the movement is faster and the frames become more hazy, and so the algorithm outputs faulty defective holes.



Frame500 Frame1500

FIGURE 5.23: Frames with results from Video 7 with offset=50



Frame500 Frame1500

FIGURE 5.24: Frames with results from Video 7 with offset=200

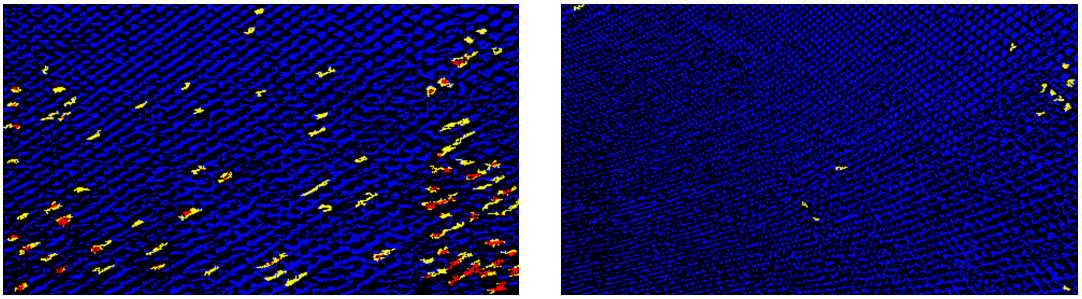
offset	10	50	100	200
frame250	315	29	10	5
frame500	467	27	4	2
frame750	610	92	20	5
frame1000	490	80	26	6
frame1250	430	23	11	8
frame1500	562	154	52	24
success rate	0.2%	2.8%	10.9%	24.6%

TABLE 5.9: Offset parameter in Video 7 frames

The increase in the success rates show how much this small change of the offset parameter actually affects the output results and the visual comes to confirm this. The only thing that stays unchanged is the foreground object - the diver's hand, that stays the same through all the offset changes.

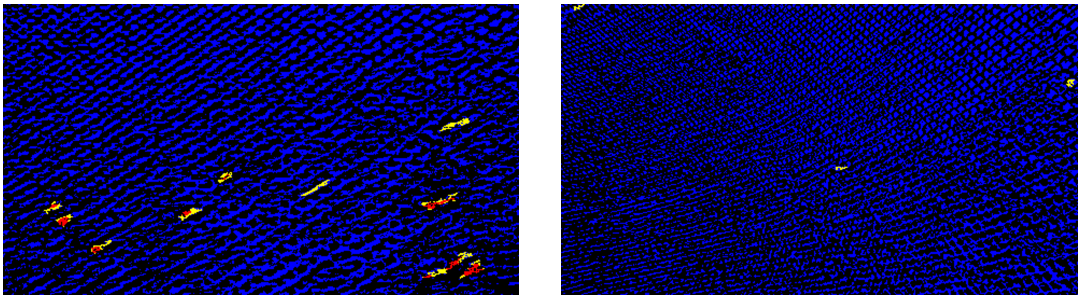
Testing Video 6

Video 6 is the video with the most success rate increase. From the initial setting with offset=50, at 3.4% to 27.3% when setting the offset=200. This is one more video where most of the faulty detected holes is due to movement of the camera and as the visual results present, it is improved a lot with the offset set high.



Frame640 Frame1600

FIGURE 5.25: Frames with results from Video 6 with offset=50



Frame640 Frame1600

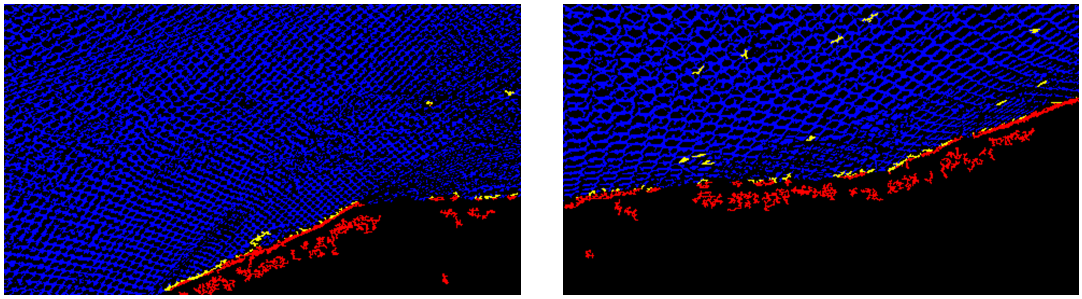
FIGURE 5.26: Frames with results from Video 6 with offset=200

offset	10	50	100	200
frame320	291	25	12	2
frame640	372	81	24	10
frame960	325	49	31	19
frame1280	349	97	41	21
frame1600	262	13	2	3
frame1920	489	70	13	3
success rate	0.3%	3.4%	15.1%	27.3%

TABLE 5.10: Offset parameter in Video 6 frames

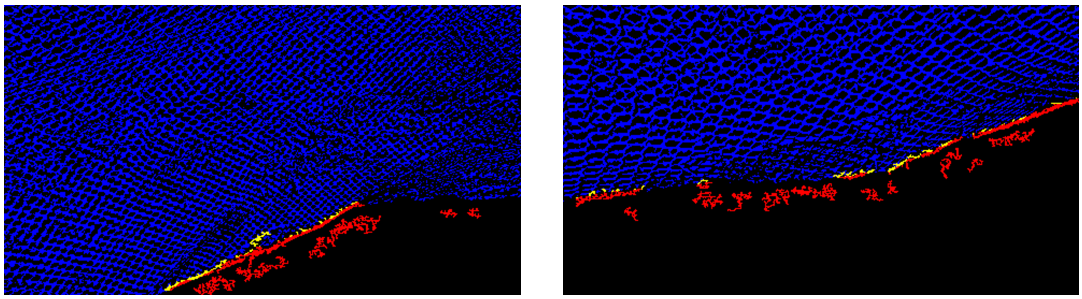
Testing Video 5

Video 5's visual output resembles a lot that of videos 9 and 10, where most of the wrongly detected defective holes are not on the net area but on the background sea around the cage borders. Even though the increase in the success rate is there, it is not as substantial as others, as the cage border remains has almost the same algorithmic response for different offset changes.



Frame674 Frame1685

FIGURE 5.27: Frames with results from Video 5 with offset=50



Frame674 Frame1685

FIGURE 5.28: Frames with results from Video 5 with offset=200

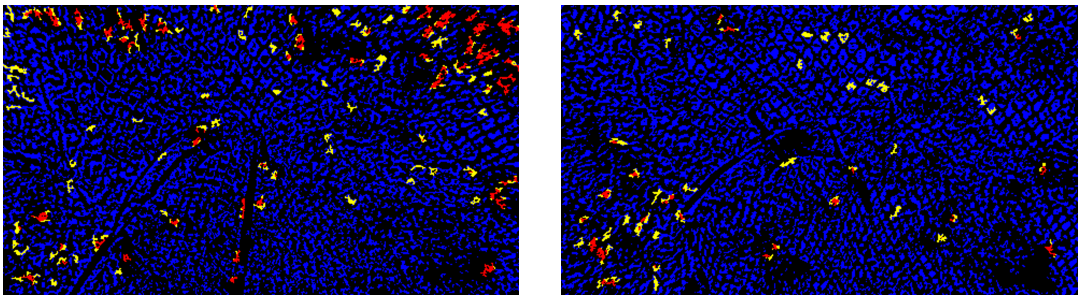
offset	10	50	100	200
frame337	143	48	27	15
frame674	80	30	20	15
frame1011	105	31	21	17
frame1348	94	30	16	15
frame1685	212	50	27	20
frame2022	172	40	29	22
success rate	1%	3.2%	5.3%	7%

TABLE 5.11: Offset parameter in Video 5 frames

Testing Video 4

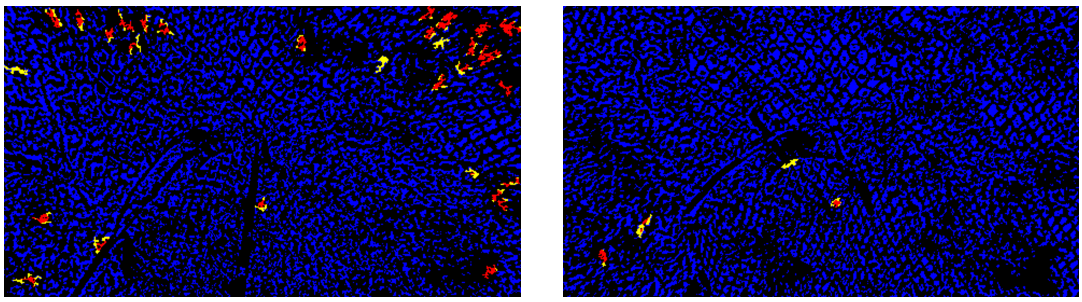
Video 4 is different than the previous videos. The main reason for the faulty holes is the foreground noise. Marine fouling is in high concentration and there are plastic bindings that get in the front and confuse the outcome.

FIGURE 5.29: Frame with results from Video 4



Frame198 Frame495

FIGURE 5.30: Frames with results from Video 4 with offset=50



Frame198 Frame495

FIGURE 5.31: Frames with results from Video 4 with offset=200

The success rate is higher as the offset is increased though, which eliminates many of the faulty output holes and gives the desirable outcome. However, it is probably not good to make assumptions at this point, as when comparing it to the colored frame, whether there are defective holes or not, is uncertain due to visibility.

offset	10	50	100	200
frame99	222	51	19	7
frame198	487	98	45	30
frame297	252	106	41	19
frame396	468	88	27	12
frame495	273	48	16	4
frame594	449	69	24	4
success rate	0.3%	1.7%	4.8%	16.2%

TABLE 5.12: Offset parameter in Video 4 frames

Testing Video 3

Video 3 is similar to Video 4 in the way that the faulty detection is due to foreground noise (marine fouling). The main difference in these two is the defective hole that clearly exists in Video 3.

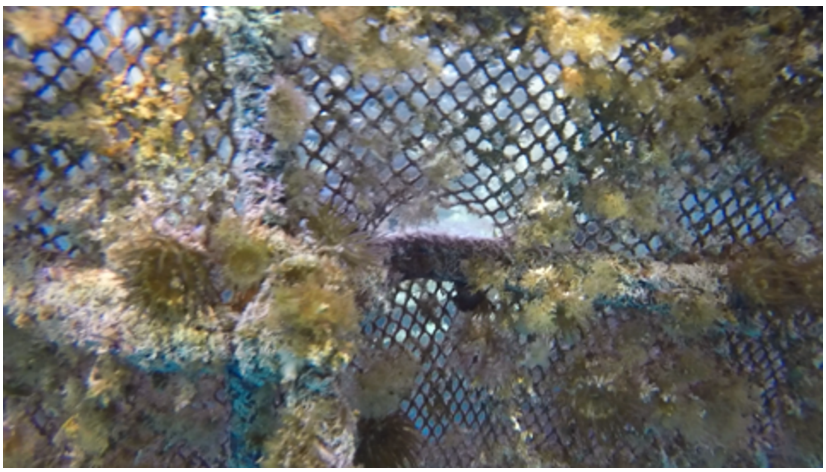
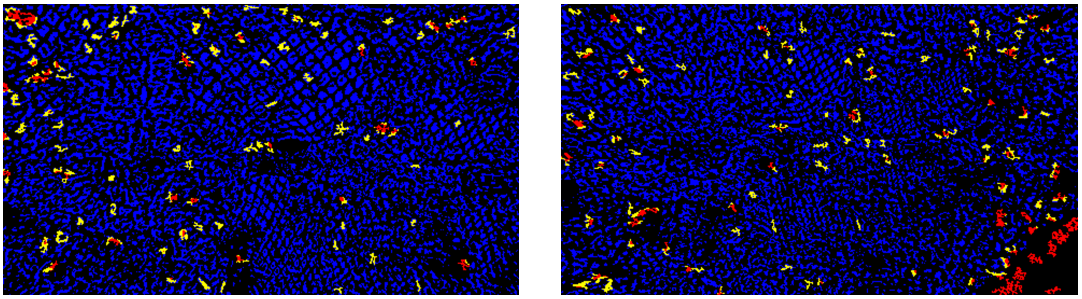
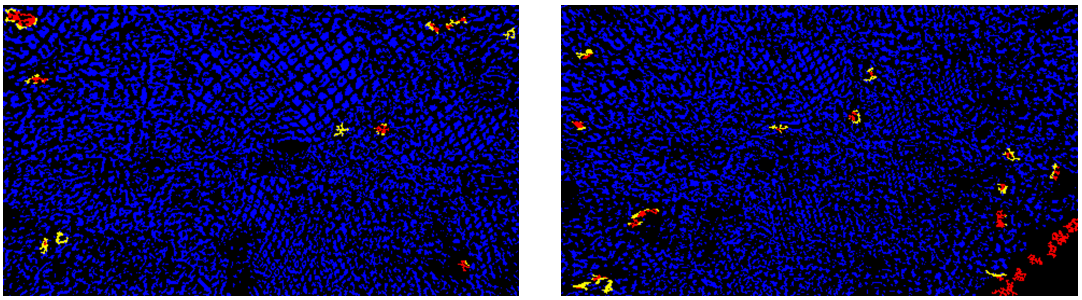


FIGURE 5.32: Frame with results from Video 3



Frame564 Frame1410

FIGURE 5.33: Frames with results from Video 3 with offset=50



Frame564 Frame1410

FIGURE 5.34: Frames with results from Video 3 with offset=200

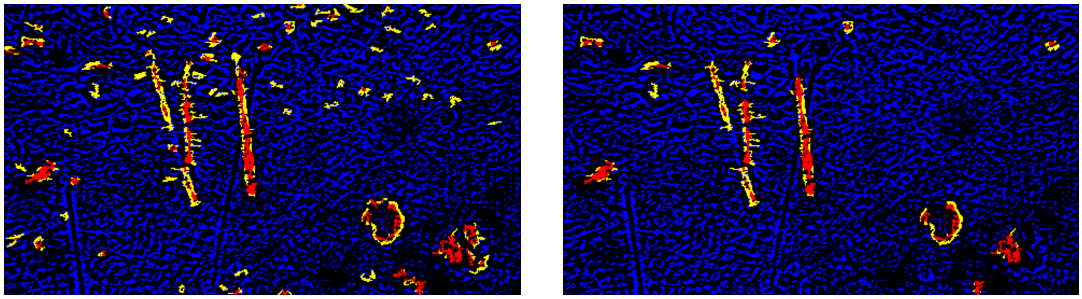
This is where we can see that the success rate alone, does not necessarily mean the correct result. Even though its is increased, which means eliminating many faulty holes, it also eliminates the correct one and thus is wrong. It would be better to keep the offset setting low as the defective hole is still detected and find another way to improve the rest.

offset	10	50	100	200
frame282	470	62	22	12
frame564	469	81	29	10
frame846	473	63	22	5
frame1128	461	81	27	7
frame1410	501	98	31	20
frame1692	449	166	82	41
success rate	0.2%	1.4%	4.1%	12%

TABLE 5.13: Offset parameter in Video 3 frames

Testing Video 2

Video 2 is also similar to the previous two. However the success rate increase is lower and the visual frame results show why. It has many more foreground objects that the offset parameter change does not affect, and also marine fouling noise makes it hard too be sure about the credibility of the rate's output.



Offset=50 Offset=200

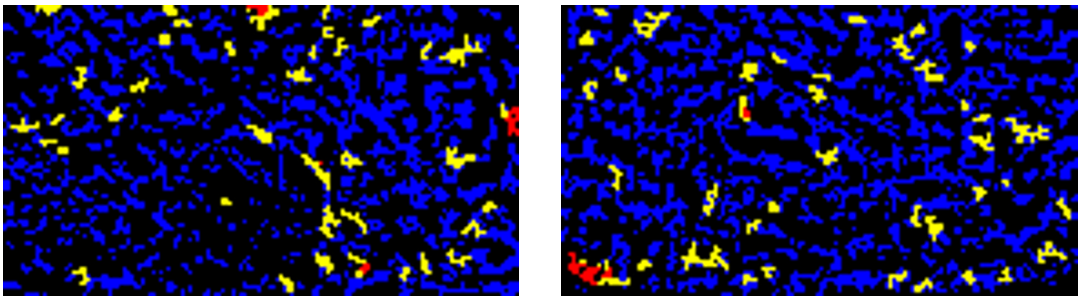
FIGURE 5.35: Frame648 results from Video 2

offset	10	50	100	200
frame324	228	38	22	9
frame648	211	77	33	17
frame972	243	57	29	12
frame1296	218	172	85	31
frame1620	232	60	33	19
frame1944	203	95	58	37
success rate	0.5%	1.7%	3.3%	7.3%

TABLE 5.14: Offset parameter in Video 2 frames

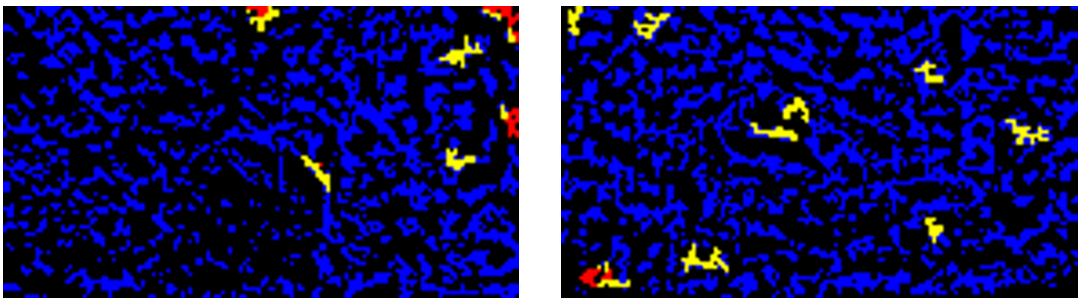
Testing Video 1

As Video 1 has lower resolution than the others, the size frames are significantly lower and thus they cannot be sampled above a window size. Of all the previous values only the fist two are accepted for testing.



Frame240 Frame600

FIGURE 5.36: Frames with results from Video 1 with offset=10



Frame240 Frame600

FIGURE 5.37: Frames with results from Video 1 with offset=50

It seems that the initial offset=50 setting, was optimal and does not need change. It has both better visual and success rate results. This video also has a lot of marine fouling attached on the nets and is difficult to determine many faulty defective holes.

offset	10	50
frame120	37	11
frame240	38	6
frame360	41	5
frame480	50	11
frame600	40	9
frame720	51	3
success rate	2.8%	19.8%

TABLE 5.15: Offset parameter in Video 1 frames

5.2.3 Evaluation of Parameter Testing Results for the Determination of the Changes Needed in the Design

For every one of the above video tests, all outputs were taken into consideration and were compared with each visual video frame output as well as with the initial colored input frame. Bellow, Table 5.16 shows all the statistical success rates of each video with the initial settings (offset=50), the changed offset=200 setting that had the best results and the improvement between them.

Video #	1	2	3	4	5	6	7
offset 50 success rate(%)	19.8	1.7	1.4	1.7	3.2	3.4	2.8
offset 200 success rate(%)	-	7.32	12	16.2	7	27.3	24.6
Improvement Factor (X times better)	-	4.3	8.5	9.5	2.2	8	8.7

Video #	8	9	10	11	12	13
offset 50 success rate(%)	2.8	4.1	3.7	0.6	0.7	1.1
offset 200 success rate(%)	13.1	17.6	7.9	3.4	4.8	7.9
Improvement Factor (X times better)	4.7	4.3	2.1	5.7	6.8	7.2

TABLE 5.16: Success rate improvement in Videos with offset changes

The highest improvement happens in Video 4 with an improvement factor of 9.5, when setting the offset to 200. However, all of the other videos have also a great difference, considering that the initial rates are low and some of them even close to 0. This shows how much can be improved in the results with only simple parameter changes.

Generally it seems that when turning the offset parameter equal to 200 value, the results are better for all the videos. However, the offset parameter is not the only one that needs to be considered in order to make the correct conclusions, since when combined with the visual output the 200 offset value is not always optimal. In many cases like in videos 3 and 4, the success rates are significantly increased, but in the visual output frames, the actual defective hole is not detected and so, it is considered wrong. A combination with other parameter settings may give better results, or for now, just the offset remaining in it's original value at 50.

Also some safe conclusions that can be made, are those of the specific cases

the offset increased the success rates. When a video consists of movement error outputs, the increase in the offset parameter value, gives a significant increase in both the success rate and the visual output. This happens in Videos 6, 7, 8 and 9 in which we also have the maximum increase in the success rate results. The increase of the offset also helps in cases of zoomed , since videos 10, 12 and 13 are significantly improved both visually and statistically.

The only category that is not affected by the offset parameter's change, is that of marine growth. In low to medium concentration, there is no impact on the output results, while in cases of videos 1, 2, 3 and 4 where it is in high concentration, the output results are unreliable visually and occasionally even made worse with higher offset.

Video 10 is a case, that has the lowest increase factor in the success rates results. That may be because it is affected by more than one difficult environmental situations, like many net cage borders, marine fouling attached on them and a zoomed in perspective of the camera.

5.3 Results

Matlab results are important for visual conclusions, as they summarize an analysis in just an image. A variety of frame outputs after video testing is shown and analysed in detail in section 5.2.2.

A low to medium amount of marine fouling does not seem to affect the algorithm on giving results and only the high concentration gives negative results. Another observation is that in most scenarios seen in chapter 4 the good results are affected mostly from the foreground noise and not the background, as in most cases, even though there are fish in the background, the algorithm still outputs correct results.

Figure 5.38 shows the increase on the performance of the algorithm with various offset changes. As clearly seen, all of them give better results when the offset is increased to 200.

All this is done considering the change in the frame color with the Net Color Detection, the floodfill variation. If the color was not changed, the success rates would be 0, as this resulted in detecting holes on the nets.

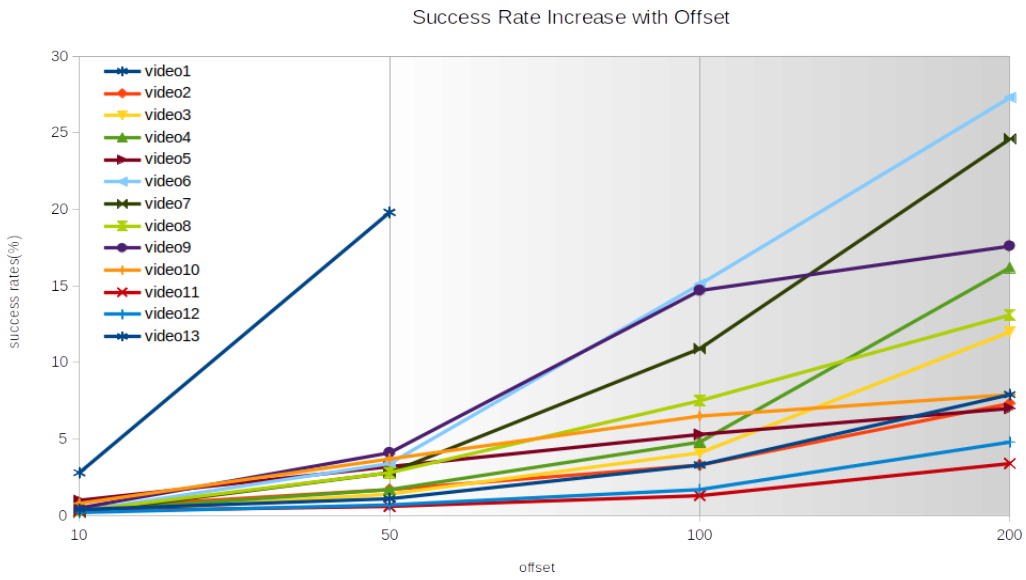


FIGURE 5.38: Offset Success Rates

Figure 5.39 shows the connection between the offset parameter and the illumination and how the results are optimum when the illumination has its lowest value(0.005) an the offset the highest(200).

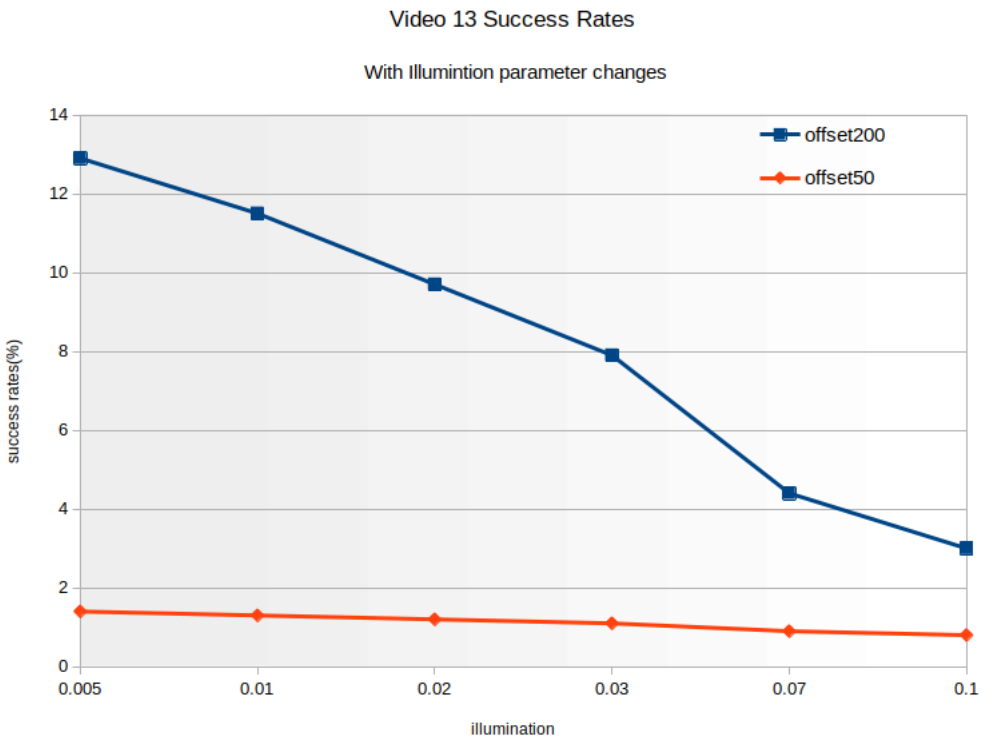


FIGURE 5.39: Illumination Success Rates for Vid 13

In Table 5.17 are presented the execution times of both the system(without floodfill) and the Net Color Detection(floodfill) function in Matlab.

Video #	1	2	3	4	5	6	7
system run time(min)	-	22.8	-	7.3	22.3	21.5	16.9
floodfill run time(sec)	1.2	3.1	2.5	2.6	2.6	2.2	2.5

Video #	8	9	10	11	12	13
system run time(min)	13.4	24.8	31	13.7	5.8	6.16
floodfill run time(sec)	2.6	2.3	3.1	2.3	2.4	2.3

TABLE 5.17: Execution time of floodfill variation in MATLAB (in an Intel i5 Processor)

5.4 Chapter Conclusion and Proposed Methods

This whole chapter's focus was on how to best take advantage of the classification system and eventually increase the output results of the already existing method on the new video data.

In the first category 'Color Nets' the obstacle was overcome, since a function was created for the automated color detection of the nets. The output of this is positive for all the videos as they give excellent output results.

As for further optimisation in the other categories, the results concluded that the 'Zoomed' -generally camera perspective- category, can be improved with increase in the offset parameter. The success rate results were excellent and give up to 29.3% increase, with the increase of the offset.

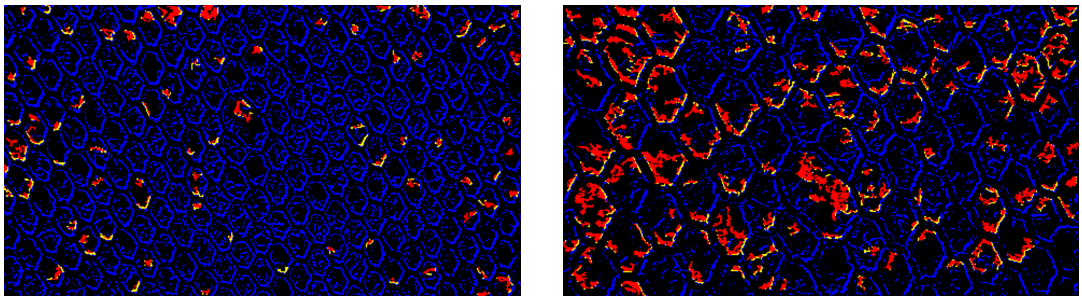
The other categories were not explored in this thesis, but here are some proposals for further exploration:

For the marine growth : first check the illumination parameter and possible changes in the enhancement function's and the adaptive threshold's parameter settings. Another way may be through colored rgb frames before the process .

5.4.1 Extra Testing Appendix

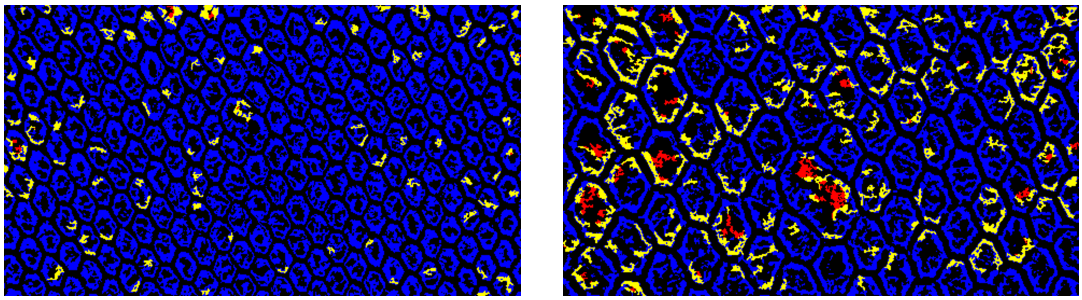
Some extra testing was executed for Video 13, that it was not continued for the other videos, is presented bellow.

-Another parameter is the adaptive threshold window size used in the adaptive threshold algorithm to produce a suitable threshold in order to make the frame binary. In Th. Zacheilas' thesis, it is tested for video 0 with the values of 4, 9, 15 and 20, so we test the same values for this parameter.



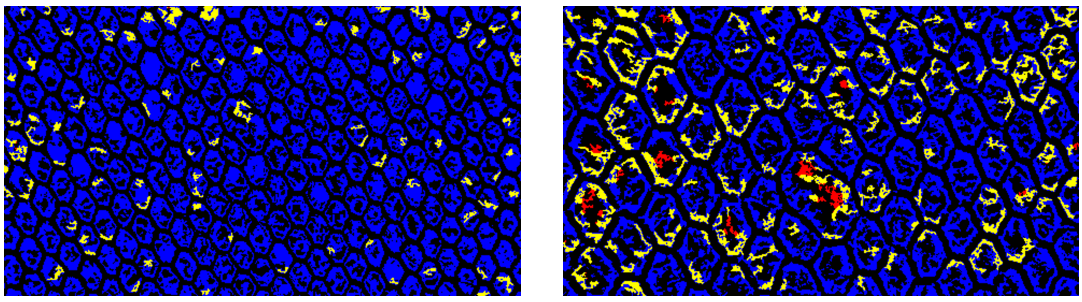
Frame160 Frame400

FIGURE 5.40: Frames with results from Video 13 with threshold window size=4



Frame160 Frame400

FIGURE 5.41: Frames with results from Video 13 with threshold window size=15



Frame160 Frame400

FIGURE 5.42: Frames with results from Video 13 with threshold window size=20

Threshold Window Size	4	9	15	20
frame160	264	0	23	8
frame240	312	0	60	21
frame320	280	0	73	26
frame400	260	0	94	25
frame480	283	0	65	31
success rate	0.0011%	0.0%	0%	0%

TABLE 5.18: Threshold window parameter in Video 13 frames

Chapter 6

FPGA Design and Results

As the previous chapters indicate, the system for the new video data was based on the previous system algorithm in Matlab. The same goes for the hardware implementation. The previous hardware design was taken into consideration to be improved and match the new data needs. Mainly, the new function for the identification of the nets color was created in order to implement it in the system.

For the Hardware Design, the implementation of the algorithm was made in C++ language through the Xilinx tools, that automatically translate it in HDL to be used for the final Design. For this thesis, in every reference of Vitis or Vivado, we are referring to the tools of AMD Xilinx.

6.1 Tools and Platforms Used

The previous system was initially build on a Zedboard device which later upscale onto a ZCU102 Evaluation Board and was implemented using Xilinx Vivado Design Suite 2019.1.

Since the Xilinx tools were upgraded in 2020, for the creation of the new Net Color Detectio(FloodFill) design, the new Xilinx Vitis Unified Software Platform was used(Specifically Vitis 2020.2). The desing was build using an Alveo U50 acceleration card.

Vivado High Level Synthesis (HLS 2019.1)

The initial design was implemented with Vivado High Level Synthesis (HLS) tool. The purpose of HLS is to create an IP block that later can be integrated on the FPGA Design through Vivado IDE. The Vivado HLS Tool allows the user to use C++ language and automatically translates it in RTL.

Vitis Unified Software Platform

The new design was implemented in Vitis 2020.2 which uses a combination of the old Vivado HLS, Vivado IDE and Vivado SDK all in one. Instead of creating a project in Vivado HLS and then importing it as an IP in Vivado, the Vitis tool lets the user write in C/C++ in a kernel code and does the rest on its own, according to the preferences of the designer. Specifically, Vitis Tool combines four elements: a Vitis Target Platform(e.g for Alveo cards), an XRT(API and drivers for the host program), the Vitis core development kit(compiles, analyzers, debuggers) and the Vitis accelerated libraries(e.g. for performance-optimized FPGA acceleration functions in image processing that would be otherwise re-implemented by the designer).

In the Vitis core development kit, an application program is split between a host application and hardware accelerated kernels with a communication channel between them(PCIe or AXI). Typically, the host first transfers the data to be operated on by the kernel(s) from host memory to global memory and after the operations are completed the results are transferred back onto the host memory.

The main operations of an application program are designed in the kernel(s) which then are compiled and build onto the system. The Vitis Compiler provides three different ways to build the design: Software Emulation Build, Hardware Emulation Build and Hardware Build.

One of the key aspects of the Vitis Unified Software Platform are the Vitis Accelerated Libraries. For this thesis we are interested in the Vitis Vision Library that provides us optimized functions for image processing.

Platforms and Devices

For this project, two different platforms were used for the two different systems. The first is ZCU102 Evaluation Kit with Zynq® UltraScale+™ MPSoC(change from initial zedboard to due to space capacity) used through the Vivado 2019.1 when implementing the previous System. And second the Alveo U50 Acceleration Card in the new Vitis 2020.2 environment for the new system implementation.

6.2 Net Color Detection (Floodfill Explore) Subsystem Design

The goal of this thesis is to work and improve on the Previous System Design for better results in the detection of defective holes in aquaculture nets. The main addition of the FPGA implementation focuses on the Flood Fill Variation Algorithm Design that detects the color of the nets and classifies them. As described in Chapter 4 and later in Chapter 5, there is a solution to identify the color of the nets and categorize the frames accordingly, in order to change the frame as needed. This solution is given through the floodfill variation algorithm (Algorithm 1) that was tested in Matlab. However it is imperative to present the way this is used in the whole system.

6.2.1 Previous System Design

The Previous System Design Flow is displayed in Figure 6.1 and was implemented in vivado hls in the same way as in Th.Zacheilas' Thesis[1] - without the red-coloured part. The top level is implemented with a DATAFLOW pragma optimization, that enables task-level pipelining, allowing functions and loops to overlap in their operation, increasing the concurrency of the RTL implementation, and increasing the overall throughput of the design.

Specifically as presented in Figure 6.1, the input and out output streams are coming from an external source and are imported using OpenCV functions with a test bench(AXI streaming). Each component represents a different function, either a Vivado HLS build in function using HLS Video Library or a custom made. In more details:

- *hls::AXIvideo2Mat* and *hls::Mat2AXIvideo*, are build in functions that Converts from a AXI Stream to the HLS::Mat format and the opposite, respectively.
- *ex_enhancement* applies image filtering with a guided filter. This function is used as proposed in thesis[1] and prepares the input frame for further analysis.
- The *hls::Duplicate* function simply creates two stream copies and passes them along.
- *Erode* and *Dilate* perform morphological opening of the image using the structuring element *strel* as input.

- *mat2gray* converts input image to an intensity image

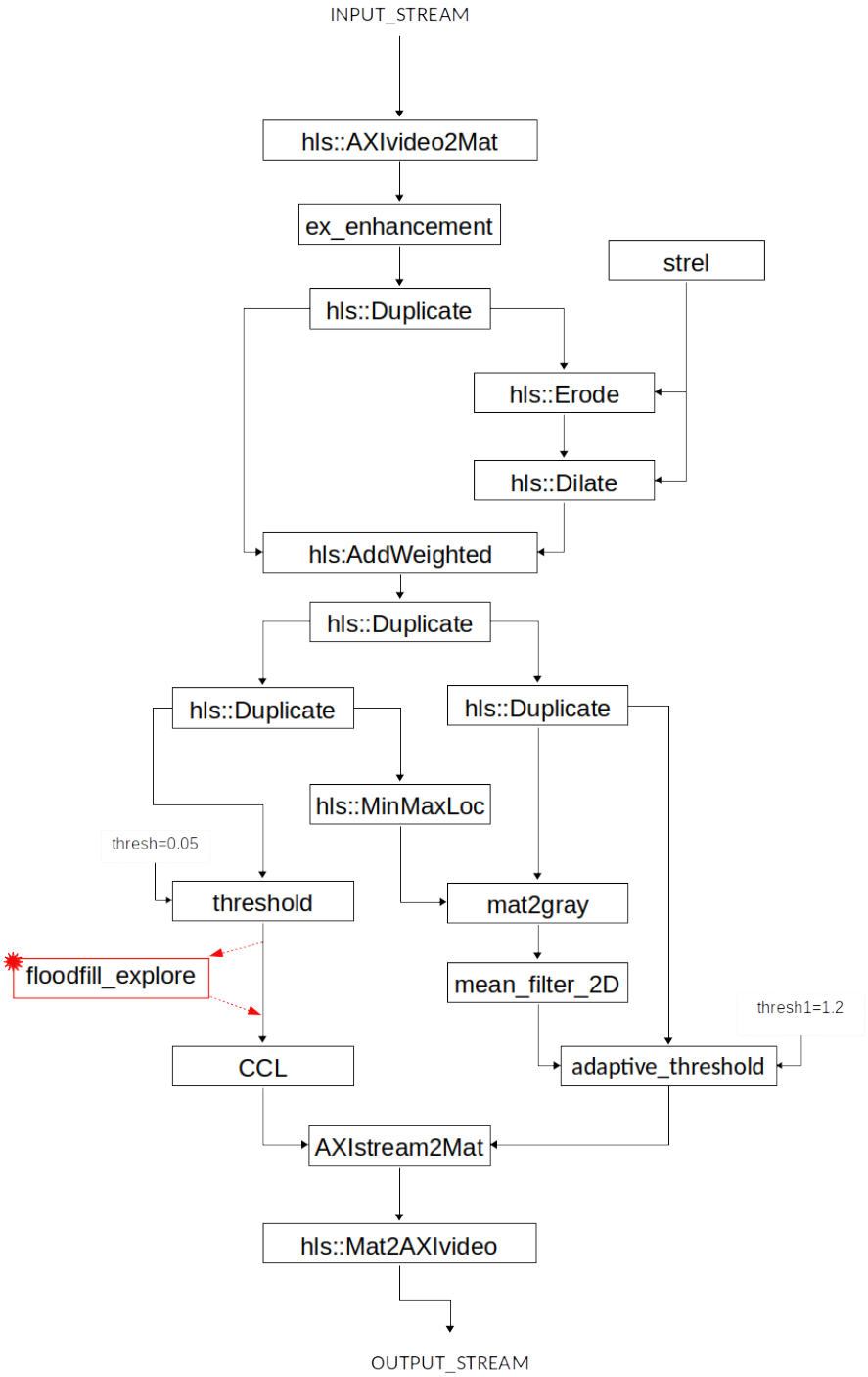


FIGURE 6.1: Top Level System Design - floodfill_explore is the module which was developed in the present thesis

- *mean_filter_2D* is a Vivado built in function which performs 2D convolution on image.
- *adaptive threshold* helps in Nets visualization. Keeps the background(nets).
- *threshold* function outputs the binarized image.
- *CCL* - Connected Component Labeling and Windows algorithm for the hole detection in the binary image. spots the holes (forground)
- *AXIstream2Mat* merges holes and nets (foreground and background) to create the result frame.

The system above is tested and works only with Video 0 as input as shown in Figure 6.4.

FIGURE 6.2: Input Frame

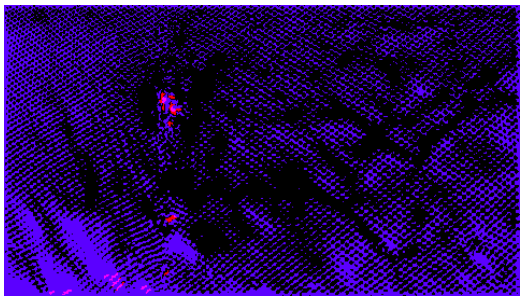


FIGURE 6.3: Output Frame

FIGURE 6.4: Video 0 test in previous System Design

Progressing on Previous Design

After the whole System is created and tested initially with the old video, Video 0, the new video data were loaded as input to test the response. With the current system, in order to get the visual results from the Vivado HLS the input images must have dimensions 480x270 and resolution 72 dpi.

The previous system is based on a specific input video and is specialized in the morphology of that video(Video 0). There are a few specific points where the design is adapted in the video's morphology like the sizes of the arrays created and some limitations specifically made for video 0.

The design needs to be adapted to the characteristics of the new video data and that means that major components of the design need to be changed altogether. The goal was to change the parameters that had good results in software and compare them, but after the above observation, we could not proceed any further with the old design system testing.

The focus in this thesis was shifted to the Net Color Detect Component that is presented in Figure 6.1 in red and is analysed below.

6.2.2 Creation of Net Color Detect Design(Floodfill Variation)

For the detection of the nets color and the classification of them in black or white in the FPGA, the process is similar to the one in matlab. The matlab algorithm is translated in C++, in order to be suitable for the Vitis environment.

When migrating the floodfill function in an FPGA Design, there are some initial observations that need to be made about the differences of the Vitis tools versus the older Vivado. First, all Vitis Vision kernels are provided with C++ function templates with image containers as objects of xf::cv::Mat class. In addition, these kernels will work either in stream based (where complete image is read continuously) or memory mapped (where image data access is in blocks).

As in floodfill implementation we need to access specific pixels of the image, a memory mapped kernel was created to facilitate that. It is more suitable than the stream based kernel as the images are written in memory blocks and can be accessed anytime in contrast with the stream input that can be processed in a FIFO manner.

The whole process begins with the host application program(Fig: 6.5), where the input image frame/image is read through the openCV function *cv::imread()* with the flag *cv::IMREAD_GRAYSCALE* to indicate that the image read only has one color channel. After that, the function cv::threshold is used in order to binarize the image. The input we tested is already only black and white but the cv::imread() function can not process it directly as binary, and it outputs the pixel values as 0 or 255 and that is why the threshold function is needed. After binarizing, the image is ready to be sent to the kernel application. First the device is set, and then the kernel is created with the variables needed through the openCL *enqueueWriteBuffer()* function. Then the kernel is launched using the *enqueueTask()* function and finally, after the kernel has done it's task, the output is passed back to the host with the *enqueueReadBuffer()* function. The last step is to display the output image with imshow() or write it with imwrite() in order to review the results of the process.

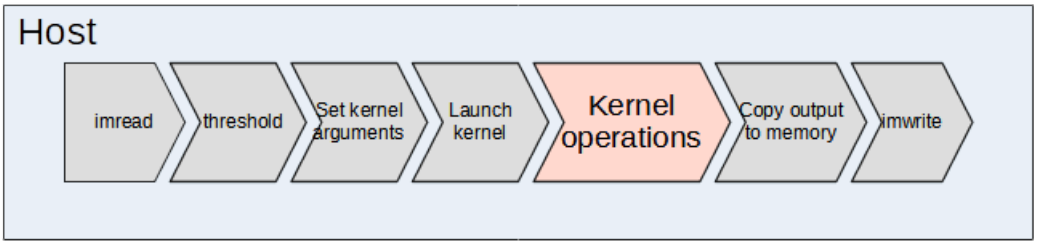


FIGURE 6.5: Host Application Program

Inside the Kernel

Each frame in the kernel is defined as an object of xf::cv::Mat class in this way: *xf::cv::Mat<XF_8UC1, HEIGHT, WIDTH, NPIX> in_mat(rows, cols, img_inp);*, where XF_8UC1 defines the datatype for the mat container. For this specific datatype it means 8bits/pixel and 1 color channel. The HEIGHT and WIDTH variables have the values of the image height and width. Also the NPIX variable is defined with the suitable NPC type which indicates the pixel per clock cycle processing either XF_NPCC1 for 1 pixel/clock cycle or XF_NPPC8 for 8 pixel/clockcycle.

A function for the main process of floodfill was created, called *floodfill_explore()* that takes as input the binary image and outputs either the same image or the opposite one, depending on whether the input has white nets or black. The whole process is shown in Figure 6.6.

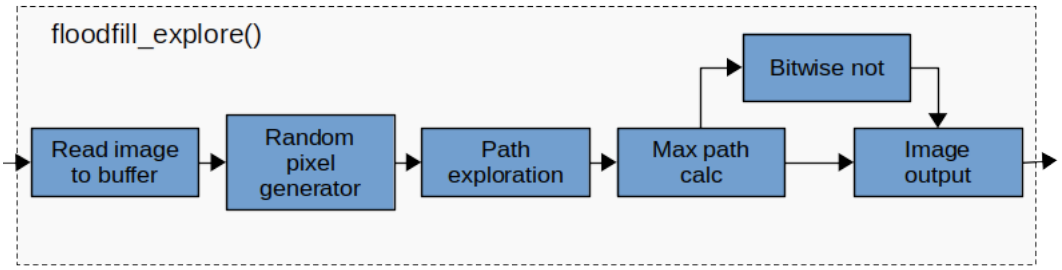


FIGURE 6.6: Net Color detection (Floodfill Explore)

The components in Figure 6.6 are described below:

- *Read image to buffer* uses Vitis Vision template functions to pass the input image into the function.

- *Random pixel generator* function was created using two *LFSR(Linear Feedback Shift Register)* registers in order to generate random input pixel coordinates.There were two LFSR created, one to generate a number for x axis and one for y axis of the frame that are both analysed below.
- *Path exploration*. is the main flood fill algorithm that traces the *Path* for each of the random pixels and finds the longest. The code is written in the same way described in the pseudo-algorithm [1], where it takes the randomly selected pixel and checks its four neighbors for their color. If they have the same, it continues setting them as the next pixel in the path.
- *Max path calc*. The maximum path is calculated and considered as the decisive factor to define the color of the nets.
- *Bitwise not* function is created with the purpose of changing the color of the pixels to the opposite one, if the nets are black to output the opposite frame of the input.
- *Image output*. This block decides whether to pass the input frame as it is, or output the *Bitwise not* frame.

All the loops used in the function are pipelined except the loop that traces the path(path_loop). The path_loop consists of dependencies that could not be resolved either with logic, or by using the *#pragma HLS dependence* directive. The dependence that is created resonates to fact that every pixel checked, depends on the check of its previously neighbor pixel.

Since the floodfill algorithm is designed in a high level language through Vitis, the code resembles that of the software in Matlab with only a few differences being the LFSRs and the bitwise_not from Figure 6.6.

Linear Feedback Shift Register(LFSR)

The random pixel generator component, the second of the figure 6.6, consists of two calls for two different LFSR register functions, *randompixel270()* and *randompixel480()* in order to generate random pixels for each frame.

The LFSR is a shift register that the input bit is fed by the other bits selection through XOR gates. This outputs a sequence of numbers that is defined by the specific feedback function that sets the new bit and is bound to be repeated after a specific amount of outputs. It is necessary to initialize the LFSR with a seed value before starting to generate the sequence numbers.

In our design, in order to have many different cases in the random pixel generator, we chose a 32-bit LFSR and tap into bit 32,22,2 and 1. The pseudo-algorithm is shown below:

Algorithm 2 : Random Number Generator for Pixel Coordinates

```
max_rand ← 2^32
function RANDOMPIXEL(seed, load)
    #pragma HLS inline
    static ap_uint<32> lfsr;
    if load == 1 then
        lfsr ← seed
    end if
    bool b_32 ← lfsr.get_bit(32 − 32)
    bool b_22 ← lfsr.get_bit(32 − 22)
    bool b_2 ← lfsr.get_bit(32 − 2)
    bool b_1 ← lfsr.get_bit(32 − 1)
    bool new_bit ← b_32 ⊕ b_22 ⊕ b_2 ⊕ b_1
    lfsr ← lfsr >> 1
    lfsr.set_bit ← new_bit
    unsigned int pix ← lfsr.to_uint()
     return pix/(max_rand/axis_dimension)
end function
```

We choose an lfsr of 32 bits and then scale it to 270 or 480 that only need 9 bits, because the random sequence that repeats has a grater repeat period than the 9-bit LFSR and we want to increase the randomness. The scale is different for each of the LFSR created, as the one is for height dimension and the other is for the width. The difference in the algorithm is only in the variable axis_dimension that is set 270 or 480 for each dimension.

For the LFSR components, we use *#pragma HLS inline* directive so that the Vitis tool recognizes the functions in the same level of hierarchy in the RTL and not as separate entities.

Integration to the main System

Even though the above application can be used as a standalone project to determine the color of the nets and simply classify the videos, it was created with the thought of being used in the previous System. If we want to implement the above process in the top level design of the initial System, this

would be a block in the dataflow after the threshold function and before the CCL function begins as shown in Fig:6.1, as the input is a binary image that comes after threshold and the output is necessary for the CCL.

6.3 FPGA Results and Utilization

This section shows the FPGA results after testing, of the new Net Color Detection component that we created, and the timing and resource estimated taken from Vitis HLS tool after the stage of Synthesis. We did not proceed to load the design on the actual board.

6.3.1 Verification Testing

The Net Color Detection implementation takes as input a binary frame of a video and outputs the same or its complement depending on the color of the nets. Figure 6.8 shows the output for video 13. The same is for the other videos that have black nets.

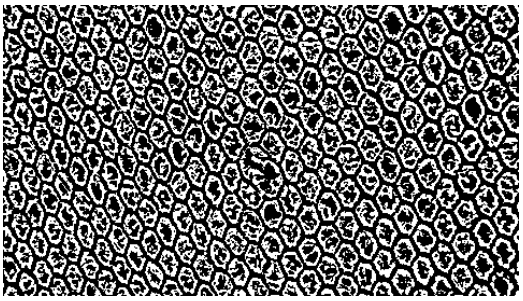


FIGURE 6.7: Input Frame

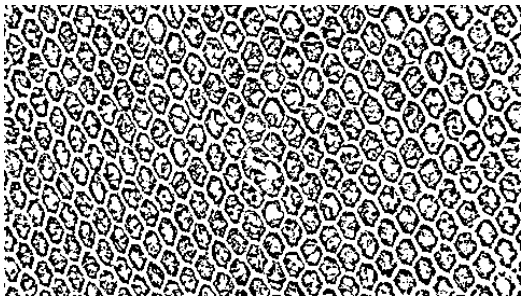


FIGURE 6.8: Output Frame

FIGURE 6.9: Video 13 test in Net Color Detection Design

To verify the output results, we run the simulation several times for each video, for multiple input frames each time. Each time the algorithm outputs the correct result, characterising the frame with the correct net color. The only ambiguous results were that of video 6, which was expected (as mentioned in chapter 5) as the lighting allows to have two different net colors in the same frame. This resulted in having both the two different outcomes with the most dominant being the black nets characterisation.

For every other video, the results were 100% correct and considering the randomness of the sample pixels, it shows how accurate and useful is this component for the task we need.

6.3.2 Performance

FPGA Resources

Table 6.1, presents the resources needed to create the Net Color Detection(floodfill) component(targeting Alveo U50). It shows that the amount needed is really small and can be easily integrated in the old system design without the need to change the original platform(ZCU102) as the design space is enough. Vitis HLS Synthesis report, also shows the utilization in percentage(%) but it is too low and is shown as 0% for all the resources.

Resource Usage	BRAM	DSP	FF	LUT	URAM
Units Used in Floodfill Component	18	0	4,478	6,362	0

TABLE 6.1: Resource Estimates (from Synthesis) for Net Color Detection (floodfill) Component

According to Th. Zacheilas[1], the Top Level Design that was targeting ZCU102 Evaluation Board, outputs the resource estimates of Table 6.2

Resource Usage	BRAM	DSP	FF	LUT	URAM
Units Used in Previous system	1,115	32	180,112	185,526	0
Utilization(%)	61	1	32	67	0

TABLE 6.2: Performance and resource estimates for Previous System

Combining the above tables, we can conclude that the Net Color detection component does not need many resources and can be easily included in the top level design without any FPGA resource problems when targeting the ZCU102 Evaluation Board which was used in the previous design.

Execution Time

Synthesis report timing estimates in Table 6.3 shows the clock period that the synthesis report of Vitis HLS tool estimates.

Clock Period	Target Set by the HLS	Estimated after Synthesis	Uncertainty in ns
Th. Zacheilas' Design (Previous Work)	4	3.5	0.5
Floodfill Design (Present Thesis)	3.33	2.433	0.9

TABLE 6.3: Clock Period Estimates in ns

From Table 6.3 we note that the current design does not impair the performance of the complete system, and hence the real-time capacity of the system is maintained.

The Vitis HLS tool also outputs an estimate for the latency in clock cycles and from that we can calculate the estimated time required for this task.

$$Execution\ Time = Period * Latency = 0.03537sec = 35.537msec$$

We report this estimated execution time for the processing of a complete frame in order to compare it against software execution in MATLAB of the same algorithm. Although in MATLAB there exist a few more calculations vs. the estimated hardware execution time, the difference 35.5msec vs. 2.26sec as reported in 5.17 shows that the hardware speedup of almost 64X renders a hardware solution necessary for the problem at hand; even if we coded the algorithm in C or another language the results would still be orders of magnitude apart.

6.4 FPGA System Setup

Setting up the system required some settings regarding the specific Vitis 2020.2 and how OpenCV and Vitis Vision Library all corespond together.

Opencv

After downloading OpenCV in our local system, adding path of OpenCV local location in the build settings of every project component is required.

Vitis Vision

The Vitis Vision Library does not need further installation or inclusion, but for many of the templates, opencv needs to be included.
Read of Mat() streams is not supported with "«" operator as it was in previous versions of Xilinx tools and need to be read with read() template function.

Useful Links

After Jan. 1st 2022, when using a previous version of the Vitis tools, an IPexport error appears that is solved with the patch in the following link: https://support.xilinx.com/s/article/76960?language=en_US

Chapter 7

Conclusions and Future Work

This chapter summarizes the work and contributions this thesis presents. It also proposes ways to further explore and improve the current system.

7.1 Conclusions

As the goal is to advance the system to work on different and more challenging input data, the evaluation of the video data(Chapter 4) plays an important factor in the process of improving the system results. It shows where the system is lacking and can be improved as well as different aspects and different conditions that were not considered before. The classification system that was created, contributed in the process of assessing the need of the video data not separately, but by turning the focus in the common factors and features of the videos.

This thesis also presents two improvements in the system. The first one refers to the first of the classification category, with the goal of detection and change of the color of the nets, which improves the system a lot, in finding less faulty results as defective holes in the nets. The second improvement is about the category of the movement of the camera that decreases further the amount of faulty results by changing the offset parameter as described in Chapter 5.

The Net Color Detection is also implemented both in software with MATLAB and in hardware on an Alveo U50 using Vitis Tools. In both, the results are satisfactory as they detect the color of the nets and change it accordingly.

The most important result from this work is that in an existing system we changed a subsystem with a new one, in such a way that neither the target FPGA technology nor the speed of the entire system were impaired in any way. At the same time the new system substantially improves the results of

the previous one over a very broad range of realistic datasets from aquaculture nets.

7.2 Future Work

The system is far from perfect yet, as the goal is to create a whole autonomous underwater net holes detection system. There are many possible paths to explore and improve the system and are proposed below:

- System integration: the scope of the current work was to improve the results from a previous system under more realistic conditions, however, the new design has not been fully integrated with the previous system, let alone downloaded on actual hardware. This is probably the first step towards future work.
- Additional parameter Testing. As seen from the offset testing, the results improved a lot. There are some parameters that may contribute a lot regarding the new data.
- Marine Fouling is present in many videos and affects the systems response a lot. It would make a great difference in the system results if the videos processed have clear nets as many of them output faulty holes for that reason.

  Could determine the HSV color range of marine growth in order to remove it or change it or set it as an area of uncertainty before the main processing. This process should happen when initially the frames are extracted from the video and are still in RGB color form.

  Or for videos with low to medium growth noise like Videos 6, 7, 8, 9, 10, 11, 12 and 13 could work by reducing it as proposed in the paper "Fishing Net Health State Estimation Using Underwater Imaging" by Wenliang Qiu
- Zoom factor. While the offset parameter calibration made significant progress in the videos that belong in the zoom/movement category, it is not yet optimal.
- Improving the hardware by reviewing the system as a whole. The FPGA design needs to be revised to accommodate the different input video data.

- Review and analyze more data in realistic and challenging conditions other than those 14 videos.

As many of the image and video processing applications around the globe use more and more AI and Machine Learning, that could be another possible path in the defective net hole detection system. It would be lacking not to propose it as the research is growing the last few years in that field and with it the hardware Design tools like Vitis AI and the various platforms that support it. (choose the hardware wisely- GPU/FPGA/VPU)

Bibliography

[1] T. Zacheilas, K. Moirogiorgou, N. Papandroulakis, E. Sotiriades, M. Zervakis and A. Dollas, *"An FPGA-Based System for Video Processing to Detect Holes in Aquaculture Nets," 2021 IEEE 21st International Conference on Bioinformatics and Bioengineering (BIBE), 2021, pp. 1-6, doi: 10.1109/BIBE52308.2021.9635351.*

[2] Th.Zacheilas, *Reconfigurable Logic-Based System for Image Processing of Fishery Nets*, Technical University of Crete, Chania, 2021.

[3] Bradley, Derek Roth, Gerhard. (2007). Adaptive Thresholding using the Integral Image. J. Graphics Tools. 12. 13-21. 10.1080/2151237X.2007.10129236.

[4] 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.

[5] Stavros Paspalakis et al. *Automated fish cage net inspection using image processing techniques*. In: vol. 14.10.2020, pp. 2028–2034. DOI:10.1049/iet-ipr.2019.1667. URL: https://doi.org/10.1049/iet-ipr.2019.1667.

[6] 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.

[7] 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

[8] Amit Shirsat and Bharat Bhargava. *Local geometric algorithm for hole boundary detection in sensor networks*. Published online 11 March 2011 in Wiley Online Library

[9] 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

[10] 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

[11] Adam Taylor. *Using HLS on an FPGA-Based Image Processing Platform*. Published Online May 31, 2018

[12] 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.

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

[14] 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).

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

[16] 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).

[17] 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).

[18] 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).

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