Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Robot Technology and Signal Processing with Industrial Economics	Spring semester, 2019 Confidential
Author: Joachim Nising Lundal Christer Aanestad Lende	[illegible] (signature of author) [illegible] (signature of author)

Programme coordinator:

Supervisor(s):

Ivar Austvoll

Title of master's thesis:

Automatic Detection of Features from Atlantic Salmon by Classical Image Processing

Credits: 30

Keywords: Salmon, computer vision, machine vision, object detection, feature extraction, individual recognition, fish, image processing	Number of pages: 128 + supplemental material/other: 73 Stavanger, 14.06.2019

Declaration of Authorship

We, Christer Aanestad Lende and Joachim Nising Lundal, declare that this project with title "Automatic Detection of Features from Atlantic Salmon by Classical Image Processing" and the work presented is entirely our own. We can confirm that:

- The work presented is done partially or completely within candidacy for a research degree at this university.
- If any part of this project has been previously presented for or in another degree, this is clearly stated.
- Where were have cited others work, the citation is always presented. With exception to these citations, the project is entirely our own.
- In any part of the project where we worked with others, it is clearly stated what was done by others and what we contributed with.

Signed: [illegible]

Signed: Christer A. Lende

Date: 14.06.2019

“The problem in this business isn’t to keep people from stealing your ideas: it’s making them steal your ideas!”

Howard Alken

UNIVERSITY OF STAVANGER

Abstract

Faculty of Science and Technology
Department of Computer Science and Electronics

Project in Robot Technology and Signal Processing with Industrial Economics

Christer Aanestad Lende Joachim Nising Lundal

The methods created for this project aim to locate the salmon in the image and extract features from it. The aim of the features are to recognize individual salmon from each other. Individual identification, done by RFID today, is important in the Norwegian aquaculture industry, mostly for scientific purposes. If this could be implemented by machine vision instead, it could be expanded to commercial purposes and tracking of larger masses, which could be economically beneficial for the industry.

Based on reasonable assumptions, some economic scenarios were computed to estimate potential savings that could be achieved by successful implementation of such a system. Based on the assumptions, it is reasonable to believe that this could save roughly 5,04 MNOK every year **per** offshore fish cage applied to. The potential costs are however somewhat uncertain.

K-means clustering was used to extract the salmon from the image. This was successful for all images. It should be noted that the data-set was cleaned of images which did not meet certain requirements.

A method was developed to detect the nose and tail tips of the salmon mainly to estimate its orientation. It worked on all images largely due to the successful cropping done by the k-means clustering.

Another method was created to detect the pectoral fin on the salmon, using segmentation by thresholding, as well as structural measures and area-thresholds. It achieved a best success rate at 99.2% on 537 images from one data-set(main set) and at worst a success rate of 93.5% on 246 images from another data-set(second set).

It was important to detect the gill-opening on the salmon, which would lead to extract a ROI around the head. The method for locating the gill-opening therefore had an important task in detecting the back of the gills, towards the body, such that the area of the head was not cropped too small. The method had an average of detecting 5.32 pixels away from the gill-opening towards the body, which served the purpose of capturing the head well.

Salmon have spots on their heads, which could serve as matching points for individual specimen. Two methods were tested to detect these spots, one of them created specifically for this project. The methods were merged together and achieved a success rate of 49.02% correctly detected spot with 0.6% false detections on the main with strict parameters. With more tolerable parameters it got a success rate of 87.4% correctly detected. Of all possible detections, it also incorrectly detected 52.01% spots. The same method was tried on the second data-set with success rates at 45.67% correctly and 4.32% false detections with strict parameters and 97.58% correct detections with 50% false detections with tolerable parameters.

An individual recognition method was created, which used feature vectors of each image to recognize if it was the same salmon in two images. The feature vectors contained up to 19 features. There were images of 178 salmon to recognize, and all but 16 had two images of the same specimen. Out of 178 salmon, it recognized 65 of them, a success rate of 36.47%, incorrectly classifying 28.8%. 34.7% of the salmon were not recognized. In other words, out of 340 possible correct classifications, 124 were correctly classified, 98 incorrectly classified and 118 salmon not classified at all.

Acknowledgement

Huge thanks go out to Ivar Austvoll for his contribution in this project. His insight and sage advice in image processing has guided us through many obstacles we struggled with. Your weekly input made this all possible.

Another thanks goes out to Johar Lundal, who made the project possible through RFID Solutions and also for his valuable contacts within the business.

MOWI, and especially Janne Kristin Engdal Øvretveit, did a great effort in helping us get a working data-set of images of salmon. Their team was professional, determined to help and efficient. Thank you all.

Contents

List of Figures

List of Tables

Abbreviation

ROI	**R**egion **O**f **I**nterest
HOG	**H**istogram of **O**rienteed **G**radients
ML	**M**achine **L**earning
DL	**D**eep **L**earning
SVM	**S**upport **V**ector **M**achine
IMR	**I**nstitute of **M**arine **R**esearch
CNN	**C**onvolutional **N**eural **N**etwork
PCA	**P**rincipal **C**omponent **A**nalysis
SPP	**S**pacial **P**yramid **P**ooling
RFID	**R**adio **F**requency **ID**entification
GHG	**G**reen **H**ouse **G**asses

Symbols

σ_B	Interclass variance	No denomination

Dedicated to the University of Stavanger

Chapter 1

Introduction

In the last centuries humankind has grown exponentially both in industry and population. This has given birth to countless inventions to better the lives of humanity, but it has also taken its toll on us. A growing population demands an increased amount of nourishment, and one solution to this problem may lie in aquaculture. Fish is the largest resource the ocean has to offer and can contribute to meet the growing demand. The aquaculture produces less pollution, whereas agriculture is the leading cause of increased greenhouse gasses through deforestation and livestock. Especially in Norway, aquaculture is an important industry.

Individual tracking of small fish populations is important for various purposes, but is too expensive for large populations. Recently, the industry has taken an interest in using computer vision to track fish, as a cheap and less brutal alternative to today's methods. This could enable even more opportunities, like individual tracking of every fish's health in a fish farm.

This project implements the use of classical image processing methods to extract structural features from bred Atlantic Salmon, such as distances and angles between points of interest. This project focuses especially on detecting the spots on the gill cover of the fish, which could be used matching fish with each other in later stages. Points were also extracted from the position of the nose, two back fins, pectoral fin and gill opening, as well as the contour of the fish. The long term goal is that these could later be used for individual recognition.

Specification of the Purpose

The aim of this section, is to better describe the purpose of the work in this report.

The long term goal is that extracted features from Atlantic Salmon could be used for individual recognition. This requires the feature detection to be accurate and robust. This is due to the potentially large size of salmon populations and the similarity between individuals.

The specific goal of this project is to:

- **Explore Machine Vision methods for automatic extraction of features from Atlantic Salmon.**
  - The ulterior motive of these features is individual tracking. This needs to be taken into account when developing the feature extraction methods.
  - Experiment with recognition using the features to get an idea of how the extracted features could be used for individual recognition.

To specify: The long term goal is to track fish individually. The goal of this project; extract features, is preparatory work for achieving the long term goal. To better explain the possibilities of the long term goal, follows three examples:

Long term goals

1. Individual tracking in a ”large” population of salmon. This would require extreme accuracy and is probably not achievable in the immediate near future. This could for example be discriminating between breed salmon in Norway, which is over 400 Million individuals. This degree of accuracy could be a fair substitute to tracking by RFID (See 1.1.1, Tracking of fish). It would probably still have chances for missclassifications, but it would be economically possible to track more fish. The need for such accuracy is questionable. It could be useful for knowing who to hold responsible for fleeing bred salmon, but the amount of fled salmon has decreased in the past years. This would also require the ability to determine whether a given fish is in the database at all, which is a tough task.
2. Individual tracking in a ”medium” population, like a fish cage. Salmon cages in Norway typically hold around 200 000 fish[1]. This lowers the demand for accuracy considerably compared to the point above. This could be useful for tracking

individual fish health in the population, which is presumed to be potentially revolutionary for the industry. Such a concept is currently being developed, as is described in 1.1.1 Biosort's IFarm - Tracking fish by machine vision, page 4.

3. Individual tracking of a small proportion in a "medium" population. This could be used to track the general biomass growth, and health, in a cage. It still requires good accuracy, as the recognised fish should not be confused with other individuals.

Important note on time difference between updating data

Another factor which affects the difficulty of the task at hand, is the time difference between each recognition. How old can the previous images be, so that recognition is still possible? Atlantic salmon grows and changes during their life, making recognition of individuals harder, especially if the frequency of updating images is too low. For the example in point 1, it should be expected that it is unknown when the individual was last updated in the database. For a controlled environment like in point 2 and 3, it is possible to capture images more often, and this is therefore more realistic because the fish will not have changed significantly since the last.

This Project's Contribution

As previously stated, the objective of this work is to start preparatory work that could be helpful towards the long term goal. Beneath is a detailed description of the plan of contributing to the long term goal, consisting of the following four points:

1. Collect and prepare images of bred Atlantic Salmon. For simplicity, the data-sets should meet a standard as described below:
    - Similar age of the fish.
    - One fish per image.
    - Similar scene in each image, as in similar background, camera placement and angle, lighting, fish placement and direction.
2. Automatically extract the fish in the image, and extract features from it. The features should be detected with an accuracy as good as possible. Practical problems such as posture of the fish, should be taken into account. The most important extracted features were:
    - Spots on the gill cover

- Front fin
- Length
- Nose
- Gill opening

3. Exploratory work and comparing of the extracted features.

4. Discuss ideas and methods that could be useful for future work.

1.1 Related work

1.1.1 Tracking of fish

Today, tracking of large fish populations is mainly done by RFID glass-tags, or so-called pit tags. Every pit tag has a unique code. The system is 100% accurate as long as the tag functions and the fish does not loose it. Only in Norway, hundreds of thousands of salmon are tagged every year. The tagging process is time consuming and expensive, and may also be harmful. The high cost associated with tagging is the core reason that not all salmon are already tagged today. There has been proposed political demands that all breed fish should be tagged, but this has been turned down, to prioritize growth in the industry.

Today, pit tags are mainly used for scientific work on "smaller" populations of fish. It is suggested that recognition by computer vision could replace the use for pit tags and apply tracking on a larger scale. The implementation of this technology could not only help save capital, but could even accelerate possibilities in the industry even further.

Biosort's IFarm - Tracking fish by machine vision

Biosort is a entrepreneur company from Norway. They are currently working on something they refer to as the "IFarm". The IFarm concept is of large interest to large fish farming companies. The basic idea is that a health journal is kept for each individual in every fish farm, where the individuals are tracked by machine vision. This is done in order to give every fish the best possible treatment, in opposite as of today, where the population of a fish cage is treated as a single unit. This is inefficient economically, and also stressful for most of the fish who are actually healthy. Typically, only 5-20% of the fish actually need treatment. Harald Takle, head researcher of Cermaq, states that individual records for each fish will be a revolution in the industry. Geir Stang Hauge, founder of Biosort, estimates that the mortality could be cut by 50-75%.[2]

1.1.2 Related Machine Vision work

The work performed by [3] studied more than 20,000 images that were captured in a harsh real-world coastal scenario at the OBSEA-EMSO testing-site in hopes of being able to track swimming individual fish with computer vision(not separating between species). The team created a 10-fold Cross-Validation framework classifier with an average accuracy of 92%. When the fish classifier was used on a live camera it still performed with acceptable accuracy, but fell easy victim to varying illumination, bio-fouling(accumulation of microorganisms algae, pants or animals on wet surfaces or in aquatic environments) and water turbidity(unclear, murky water).

[4] studied a coral reef in southern Taiwan with cameras, trying to detect fish in the image and classify them into different species. The fish detection discriminates between fish and other moving objects, such as sea anemones, drifting water plants and such. [4] proposes a maximum probability, partial ranking method, which is based on sparse representation-based classification. To get the features of each species of fish, eigenfaces and fisherfaces were used. Feature space dimension and partial ranking value are used to optimize the solutions. Respectively, the recognition and identification rate could achieve 81.1% and 96%. Experimental results showed that this approach was robust and highly accurate in use of fish recognition and identification in a real-world underwater observational video.

[5] extracted fish from images with sparse and low-rank matrix decomposition. The CNN(convolutional neural network) needed features to analyse, which were gathered with a deep architecture consisting of a principal component analysis(PCA) in two layers, followed by a binary hashing in a non-linear layer. Block-wise histograms were used to pool features together. Spatial pyramid pooling(SPP) was used to extract information invariant to large poses, before a linear SVM(support vector machine) classifier was used to classify the fish in the image. [5] managed a 98.64% accuracy on detection the various fish in the images of varying background and on different species.

[6] used shape matching for fish recognition. They tried several different shape descriptors, such as Fourier descriptors, polygon approximation and line segments. In tests with four distinct species, their software correctly determined the species with greater than 90% accuracy.

[7] used texture and color to locate fish in video files. To Track fish once they had been detected, a combination of two algorithms were put in use: matching of blob features and histogram matching. 20 different underwater sequences were tested, sampled to 320x240 with a 24-bit RGB camera and a frame rate of 5fps. Each video consisted of 300 frames, summing up to about a minute. The method performed as well as 89.5%

and down to 80%. The method was then tried on 20 movies(about 800 frames). It counted all the fish present in the video and achieved a success rate of 85.72%.

[3], [6] and [4] deal with tracking fish in an image. While [5] also aim to accomplish this, they also separate between species. [7] also located fish in videos and tracked them. Though this project also deals with locating fish, differentiation between species and individual recognition on one species are two different tasks. Most other articles surrounding fish tracking and recognition deal with locating the fish, tracking and counting it, and separating between different species of fish. The method created for the purpose in this project aims to, or build foundation for, recognition of individual fish within the same species. The articles above prove helpful in locating the fish, but offers little on the further tasks. Methods used in human facial recognition might prove more helpful.

1.1.3 Facial Recognition

[8] discusses some techniques for facial recognition, such as PCA, LDA(linear discriminant analysis), BPN(back progapation network) and structural matching methods, where distances between eyes, width of the head, distance from eyes to mouth etc. were used to separate individuals. A structure based method of matching features is preferred when wanting to recognize salmon individuals. [9] also mentions feature-based facial recognition, using geometric relationships between facial features, such as eyes and mouth. Using euclidean distance measurements, a peak performance of 75% was achieved on a database of 20 different people with two images of each person(one for reference and one for testing). This was performed by Kanade[10] in 1973.

Bruntelli and Poggio[11] built on Kanade's method and computed a vector of 35 features from a database of 47 people, which had four images per person. They achieved a recognition rate of 90%. With a feature based approach, a certain tolerance must be given to the vectors, since they can never fit perfectly on the structures in an image. The tolerance cannot be too high, as that will destroy the precision required for individual recognition.

Cox et al. [12] got a recognition performance of 95% on a database of 685 images using a 30-dimensional feature vector derived from 35 facial features. It should be noted that these facial features were manually extracted and it is reasonable to assume that the recognition would be lower if an automated approach for feature extraction was used.

[8] mentions three stages to facial recognition: *a):* face detection, *b):* feature extraction, and *c):* facial image classification. In similarity with face recognition on humans, by

gathering features on each salmon, structural feature-based methods can be applied to classify the individuals.

1.2 Environmental and genetic impact on a Salmon's appearance

When it comes to using Computer Vision on Atlantic Salmon, It is important to have an idea of how the fish's appearance can rapidly change, especially when one is considering recognition over time. Recent study have shown that the environment plays an important role when it comes to the appearance of Atlantic Salmon.

Brightness, Color and growth rate

Both Fig. 1.1 and 1.2 shows Atlantic Salmon smolts (see appendix B for life cycle of Atlantic Salmon). There is a significant difference in the appearance brightness and color. Both salmon were taken from larger populations where the average length of the bright ones were around 21cm, and the darker fish's average length were 19cm. Interestingly, the longer fishes were about 12 months old, while the smaller about 16 months old. B Biology of Atlantic Salmon explains how it is normal that populations in different enviroments grow at a different rate. Further observations to illustrate this is presented below.

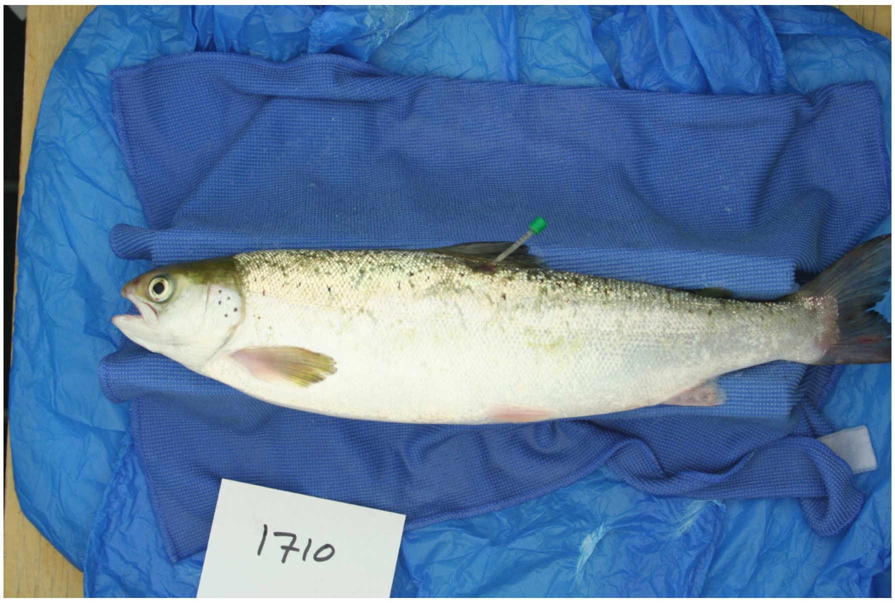


FIGURE 1.1: A 12 month old Atlantic Salmon Smolt, breed by IMR

Ole Folkedal, a biologist from IMR (Institute of Marine Research) explained that the brighter population had been raised in a white tub and illuminated by strong lamp light. This caused their skin to turn brighter. In an experiment, he once forgot to on how the salmon reacted to a strong strobe light, he forgot to turn the light off. After an hour of exposure, the fish had curled up at the bottom of the tub and its skin was all dark.

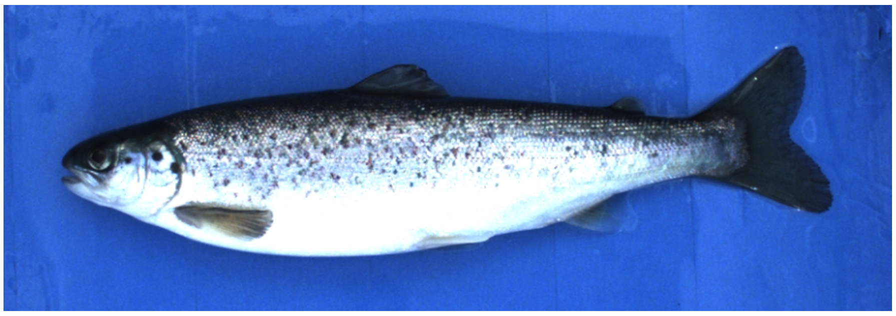


Figure 1.2: A 16 month old Atlantic Salmon Smolt, breed by MOWI

When the light was turned off, the salmon rapidly returned to its normal color and behaviour.

The growth rate of Salmons are especially important when considering recognition over longer periods, to estimate how structural features from the fish, may have changed.

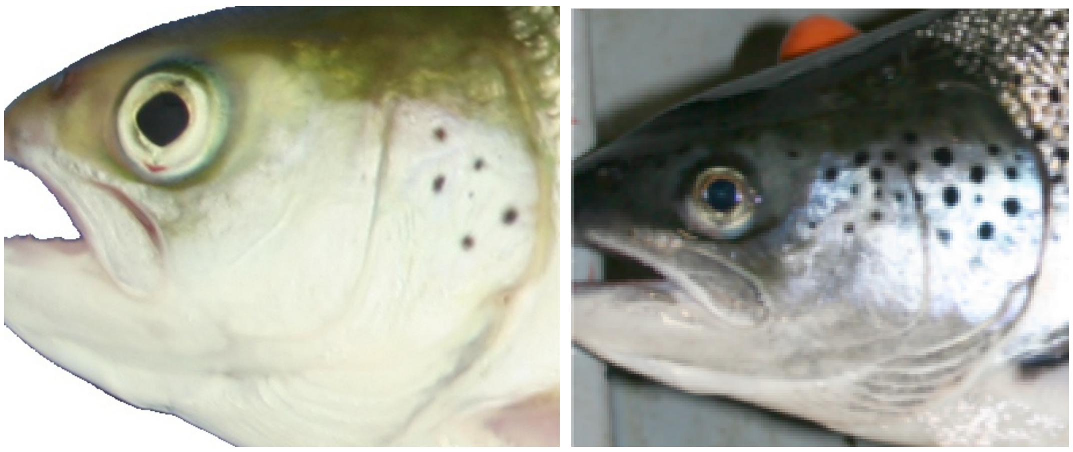
Rapid change in skin brightness is important to consider when working with machine vision, especially with methods involving color and lightness matching. Also, the methods may work differently on salmon of different color. Machine vision methods might have to be adjusted for working well on both images of the types in fig. 1.1 and 1.2.

Spots in the skin

Atlantic Salmon have so called melanophore spots in the skin, which will only be referred to as spots in this report. Their main purpose is to protect the skin from UV damage. Recent evidence also suggests that spottier fish often are more fit and dominant. What causes the distribution and density of these spot patterns are somewhat unknown. It is known that color-changes in the skin is connected with the pigment concentration and the distribution of chromatophores. Yet, in recent studies by IMR, they found that the environment is the main determinant of the spot-pattern development among Atlantic Salmon. For example, Salmon raised in rivers, were shown to develop 1.8 times more spots than domesticated Salmon.[13]

The coat pattern of a salmon will repeatedly change between life stages, like the transition from Parr to Smolt(See appendix B). This will make individual recognition over time difficult. However, concerning the spots, once one has taken shape, it will not disappear. In contrast, it will grow larger, and more distinct. This means they could be used for long term recognition [14]. Unfortunately, the appearance of these spots does not seem to accelerate until the late stage of the smoltification stage (See appendix B). Fig. **??** and **??** shows sample images from a "Spot Recognition" study conducted by IMR. These two images display the same fish at 12 months, and later at 22 months.

It can be seen that the same spots are present at the older stage, and that all existing spots have grown more distinct. Also, many more spots have appeared, but these are less dominant. This was the case for all the 246 individuals in this study.[14]



(a) (b)

FIGURE 1.3: The same Atlantic Salmon at 12 and 22 months. The fish has grown and changed its physical appearance. Still, It is possible to recognize the similarities in the spot pattern. The spots that were present at 12 months, are the most distinct at 22 months, and more spots have appeared.

In early 2019, IMR performed a rough analysis on 300 Atlantic Salmon with an average weight of 3.6kg, that were ready for butchering. The population included four families from the same sea cage, whereas three of the families had an average of around 11 spots on the gill cover, and the last one only had an average of about 2.5.(Information from Ole Folkedal) This shows that also the genes play an important role in spot pattern forming, as these families were raised in the same environment. Fig. 1.4 and 1.5 shows two very different fish from this population.

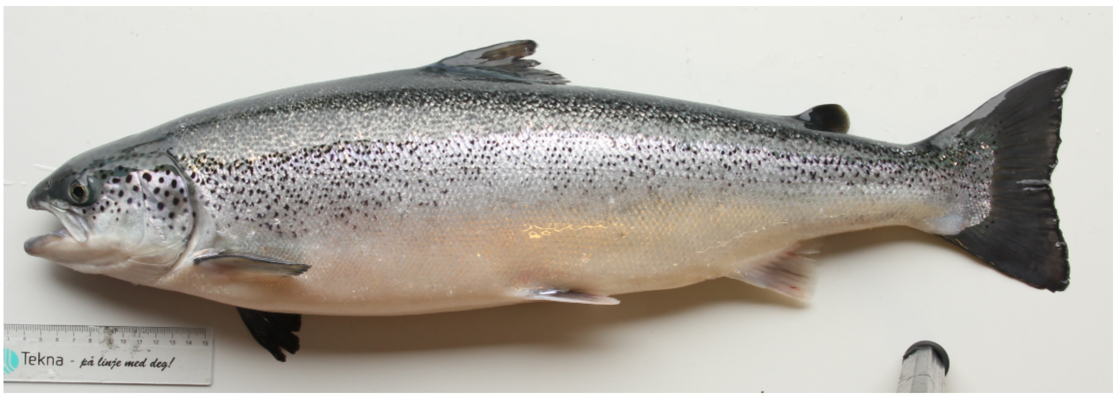


FIGURE 1.4: An adult, bred Atlantic Salmon, ready for slaughter. This fish has a significant amount of spots.

Based on earlier studies such as [14], it is evident that spot pattern could be used for individual recognition, at least to a certain degree.

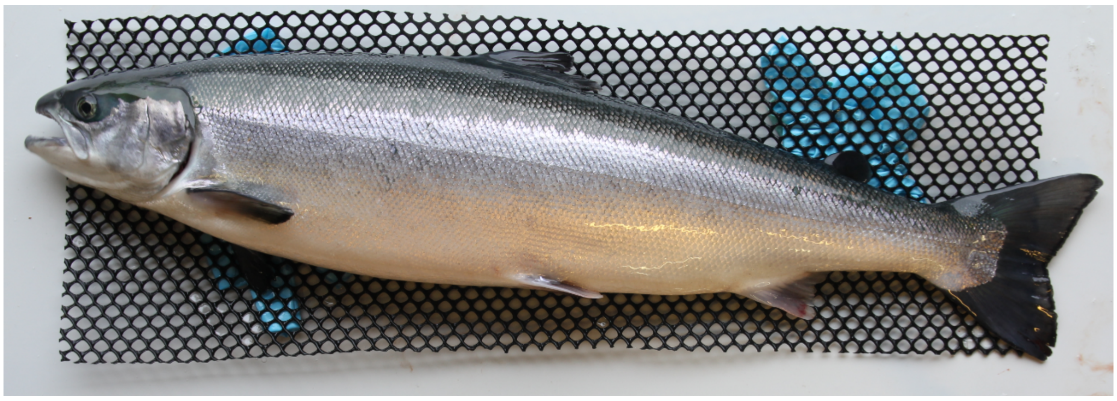


FIGURE 1.5: Another bred Atlantic Salmon from the same sea cage as the one in fig. 1.4. This one seems to have no spots at all.

In some contexts, the spot patterns of fish are referred to as ”fingerprints”. This is a nice description for some of its possible uses, but the description is inaccurate, as a usable spot pattern is normally not developed until the fish is almost adult, and some salmon have very few or no spots at all. The pattern changes over time, in the way that more spots are added as it grows. This does however not mean that the spot pattern cannot be used for individual recognition. Only that one has to be aware of its uncertain nature.

1.3 Methods

In object detection today, two ways of approaching problems are classical image processing methods and machine learning methods. Each of them has their strengths and weaknesses within computer vision and can be valuable assets in object detection and feature extraction.

Classical methods in image processing offers various tools to both extract objects and features within an image. This chapter will bring forth some of these methods that can prove helpful.

1.3.1 Classical image processing VS Machine learning

The use of machine learning(ML) algorithms in object detection and feature extraction is growing with great expanse. Many applications today use different ML algorithms, such as Deep Learning(DL), Haar cascade classifiers, neural networks and so on.

Even though these tools are strong, they are built on classical image processing principles, such as the Haaar Cascade Classifiers is built upon using the haar transform.

There are of course more things going on, but classical methods should be sufficient in detecting the salmon and various features on the fish, or could perhaps outperform the ML algorithms. Article [15] defends classical methods compared with deep neural networks. It is argued that DL algorithms are attractive because they require minimal human design because of their nature with using collected data, but a downside is the finite computing power. GPUs(which DL and other ML methods run on) are power hungry. Hence, using a GPU for each module to run is restricting. To make a proper ML algorithm requires proper understanding of the problem. Without it, many classifiers will be sub-optimal network designs[15].

A vital aspect in using ML algorithms will be the data, which is in this case the images of Atlantic Salmon. The larger the dataset available, the better the ML algorithm will perform. Of course, this also depends on which type of ML is used. [5] used a Support Vector Machine(SVM), a type of ML, with a convolutional neural network(CNN) together with a sparse and low-rank matrix decomposition to detect and recognition. To train the classifier, 27,370 images of fish were used. The algorithm in [16] uses HOG combined with an SVM to detect vehicles. To make the classifiers, 8792 images of different vehicles were used. The face detection classifier made by Paul Viola and Michael J. Jones used 4916 positive images to create the haar cascade classifier[17]. The best suited dataset of salmon that is available for use is 604 images of salmon of 202 individuals. This is generally not enough data to use in ML algorithms and the created classifier would be too weak(meaning falsely classifying too many images) for use.

This project therefore utilizes classical image processing methods to detect the salmon and gather features from it.

Further chapters

The following chapters contain:

- Chapter 2: Theory about relevant themes for methods used for object detection and feature extraction.
- Chapter 3: Implementation of methods to gather different features from salmon.
- Chapter 4: Experimentation and results using methods presented in the previous chapter.
- Chapter 5: Economy: A stand alone chapter, which addresses drivers in the Norwegian and world-wide aquaculture industry. Assumptions are made, and simulations

are conducted in order to determine the actual economical impact individual fish recognition could have for the Norwegian industry.

- Chapter 6: Discussion about the results gathered in the project.
- Chapter 7: The conclusion of the project.

Chapter 2

Theory

How an image is taken is important in regards to image processing. Avoided sharp shadows and getting clear images with a good setup is important. Theory for optics is first presented. Segmentation follows next; how to divide an image into different regions and extract valuable information is crucial when detecting features of an image. The theory for segmenting an image using thresholding and k-means clustering is presented, followed by edge detection. Finally, theory on how to detect spot-like features is presented.

2.1 Optics

In physics, Optics is the area that focuses on the behaviour of lighting and ways to detect it. In Machine Vision, thorough ground work on the optics is often vital for later stages of applications. Unfavourable lighting conditions can cause several problems in images. Especially uneven illumination and sharp shadows are considered troublesome when concerning feature extraction from objects.

A way to minimise both these problems is to have light come from several angles, by several light sources, and to have the light spread by diffuse reflection[18]. This would be especially important when using light sources that emit direct light, such as LED.

Diffuse reflection is when light-waves are spread by hitting a reflective surface or passing through a medium. Any material that diffuses the light, is called a diffuser. There are several specific methods that are used to diffuse light, but a cheap and effective one is to use a matte white surface or medium, because this absorbs little light. The theoretical perfect diffuser reflects 100% of the light it receives and spreads the waves equally in all directions, so that it will appear equally bright from all angles[19]. Fig. 2.1 illustrates a case of perfect diffuse reflection.

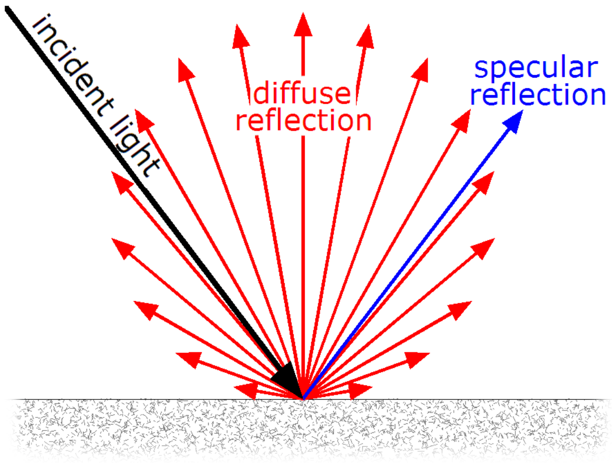


FIGURE 2.1: An illustration of perfect diffuse reflection off a surface. The blue arrow is an example of the opposite, where the reflection is not diffused at all. This is what is wanted from a mirror. Image from: Wikimedia Commons. https://commons.wikimedia.org/

Fig. 2.2 shows a practical example, where a white umbrella is used as a medium, to diffuse sunlight. As a result, sharp shadows on the frog are removed, and colors become more monotone.

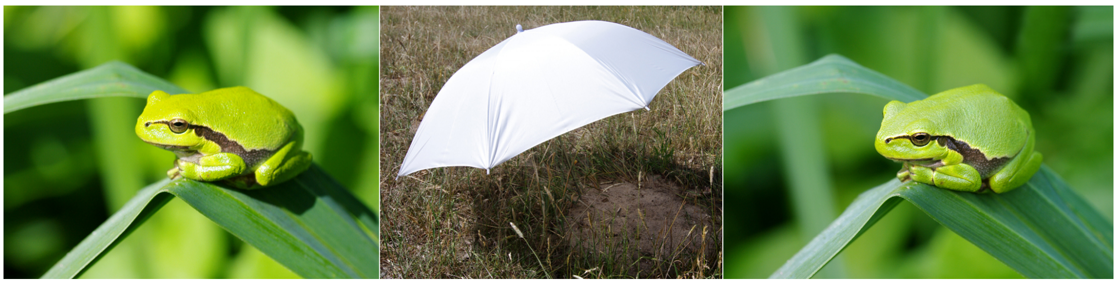


FIGURE 2.2: An example of a practical diffusion solution, and its result. Here, a matte, white umbrella is used as a medium to spread the sunlight. Notice the difference in sharp shadow edges from the image to the left compared to the right. Also, the lighting on the frog itself is more even in the diffused image. Image from: Wikimedia Commons. https://upload.wikimedia.org/

2.2 Segmentation

The purpose of segmentation is to extract information from an image such that the output image holds only a fraction of the amount of information of the original, but leaving the little information in the original much more relevant and valuable for an automatic vision system.

Image segmentation is to divide parts of an image into regions by their outlines. That is to separate the image into areas made up of pixels that have something in common, whether it be similar color or brightness that suggests they belong to the same object or part of an object[20]. Fig. 2.3 is a segmented image of a salmon.

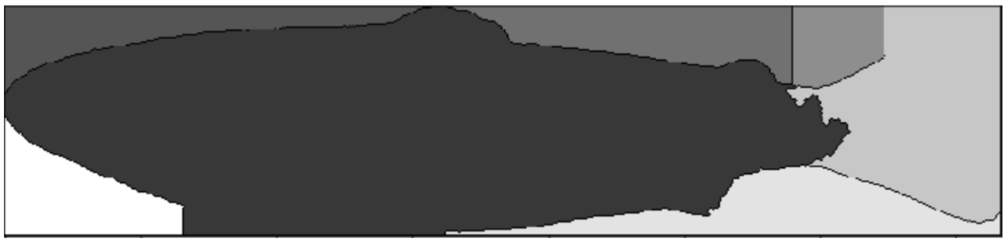


FIGURE 2.3: A salmon segmented into different sections. The body of the fish is clear.

Fig. 2.3 shows the different areas in uniform color. The shape of its body is clear, because pixel in that region had something in common.

2.2.1 Segmentation by Thresholding

One of the simplest ways of performing image segmentation is with the use of thresholding. If the number of pixels with a specific gray value is plotted against that value, a histogram of the image is created. If the histogram is properly normalized, it essentially functions as the probability density function of the gray values of the image.

Assume there is a bright object in an image surrounded by a dark background, and the object is to be extracted. For this image, the histogram will have two peaks, one for all gray values in the dark background and one for all the gray values in the bright object. This is illustrated in fig. 2.4.

In fig. 2.4 there is a valley between the two peaks. If the threshold value is chosen to be the one corresponding to the gray-level in the valley, indicated by t_0, and label all intensities greater than t_0 as the object and those lower than t_0 as the background, the object can be extracted from the image.

What is meant by ”extracting” an object from an image, is to identify the pixels making up the object. To express this information, an array is created with the same size as the

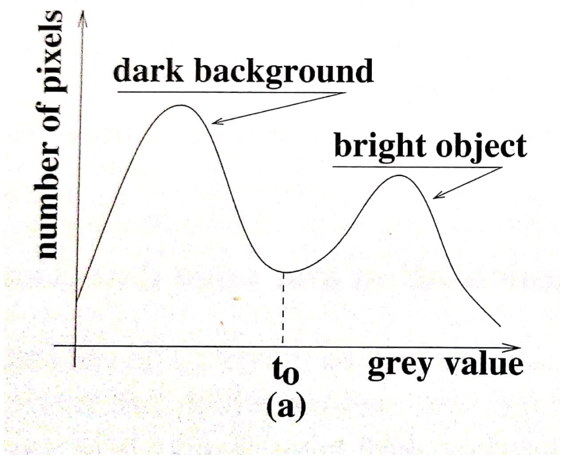


FIGURE 2.4: The histogram of an image with a bright object and a dark background, giving two peaks. t_0, depicts the threshold that separate the image into black and white.
[20]

original image, where each pixel is granted a label. All pixels that make up the object of the image are given the same label, and the pixels for the background another label[20], thereby segmenting the image.

Fig. 2.5 shows a histogram of an image of a random salmon on an originally blue background. The image is bimodal[21], meaning that is has two peaks, also as in fig. 2.4.

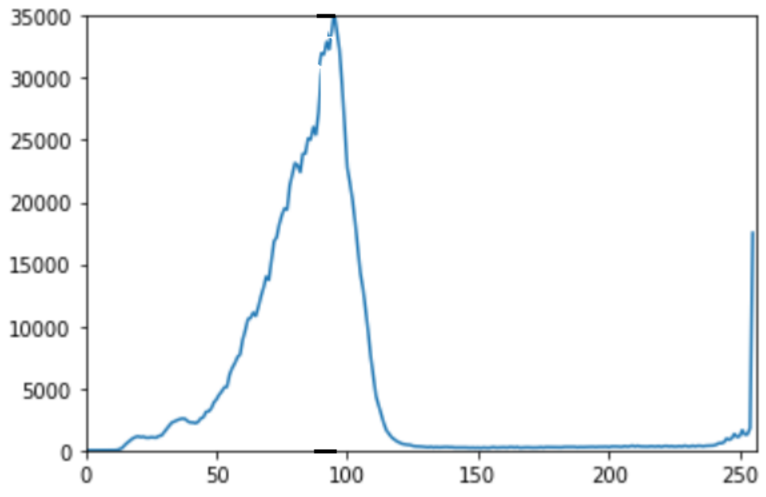


FIGURE 2.5: The histogram of a random salmon on an originally blue background. The grey-level values of the background merge with the gray-levels of the salmon, except for the bright parts of the fish.

A salmon vary in intensity. It has both very dark and bright areas. The dark areas of the fish blend in with the background, since there is no distinct valley for the darker values in the histogram. But there is a spike at around 250 in the histogram, those values being the bright values of the fish.

If there is no distinct valley between the object and the background, hysteresis thresholding can be applied. Hysteresis thresholding is often used when many of pixels of the background have the same values as the object and vice versa. This is particularly a problem near the boundaries of the object, which could be fuzzy and not sharply defined(as is the case with a salmon gray values merging with the background). Hysteresis thresholding uses two threshold values instead of one, t_1 and t_2. The highest threshold is used to define the "hard core" of the object. The lowest of the two thresholds is used in conjunction with spatial proximity of the pixels: if a pixel has intensity value lower than the greatest threshold, but higher than the low threshold, it is labelled as an object pixel only if it is adjacent to a pixel labelled as a core object pixel[20]. A histogram with hysteresis thresholds is illustrated in fig. 2.6

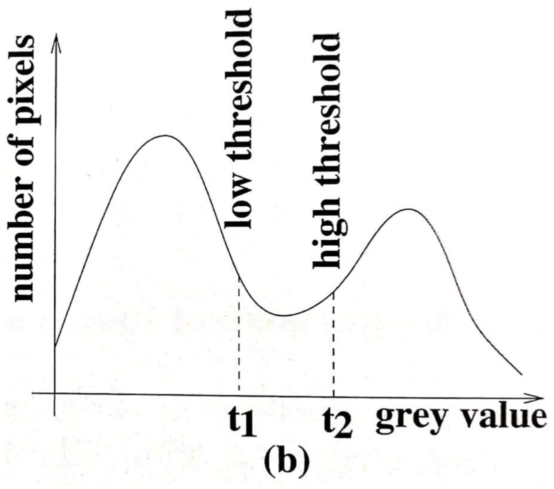


FIGURE 2.6: The same histogram as before, but this time with two thresholds to better extract the object of the image. This is called hysteresis thresholding. [20]

Otsu's Threshold

Otsu's method is a method developed directly in the discrete domain, and aims to find the threshold that maximises the distinctiveness of the two populations to which it splits the image into. Eq. 2.1 is used to compute the distinctiveness, which is the interclass variance.

$$\sigma_B^2 = \frac{[\mu(t) - \mu\theta(t)]^2}{\theta(t)[1 - \theta(t)} \tag{2.1}$$

Where $\mu(t) \equiv \sum_{x=1}^{t} xp_x$, $\theta(t) \equiv limits_{x=1}^{t} p_x$, and in turn where $p(x)$ are the values of the image histogram, μ is the mean gray value of the image and t is the hypothesised threshold. The idea is to start from the beginning of the histogram and test each gray value t for the possibility it is the threshold that maximises eq. 2.1, by computing the values of $\theta(t)$ and $\mu(t)$, and substituting them into σ_B^2. Like this, the t that maximises eq. 2.1 is identified. The method assumes that σ_B^2 is well behaved, meaning that it it only has one maximum[20].

It general, otsu's method places the threshold in between two peaks if the image is bimodal[21].

There are some drawback when using otsu's method:

1. The probability density functions $p_0(x)$ and $p_b(x)$ are described, by the method, only by using their means and variances. This means that it tacitly assumes these two statistics are sufficient to represent them, even though it may not be true.
2. When the two populations are very different from each other in size, σ_B^2 may have two maxima and the correct maximum is not necessarily the global maximum.
3. The method assumes that the histogram of the images are bimodal, meaning it only contains two classes. If there are more than two classes in the image, the method must be expanded so that multiple thresholds are defined, which maximises the interclass variance and minimises the intraclass variance.
4. The method will divide the image into two classes, regardless of the division making sense. It should not be applied directly is under variable illumination.[20]

Adaptive Gaussian Thresholding

While most threshold methods set a fixed threshold for the whole image, adaptive thresholding sets a threshold for each pixel based on the neighboring pixels. To calculate this threshold, $T(x, y)$, the following steps are performed:

1. An NxN region around the pixel location is chosen.
2. The weighted average of the NxN region is calculated, either with using the average of all the pixel location that lie within the NxN area, or by using the Gaussian weighted average. The Gaussian weighted average will weight pixels closer to the center of the NxN region heavier, meaning they get more of a say in the choosing of the threshold, denoted by $WA(x, y)$.

3. Calculate $T(x, y)$ by $T(x, y) = WA(x, y) - C$, where C is a constant subtracted from the weighted average, such that the threshold can be scaled as wanted for different results.

Fig. 2.7 shows an image with clear dark and bright contrast of tables on white paper. However, notice how at the right lower corner there is a sharp shadow. The image is thresholded first with a fixed threshold, then with adaptive thresholding with both weighted average and Gaussian weighted average, shown in fig. 2.8.[22]

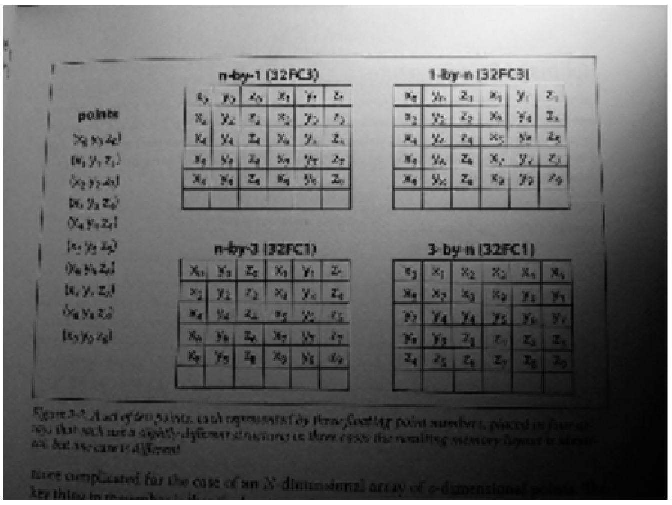
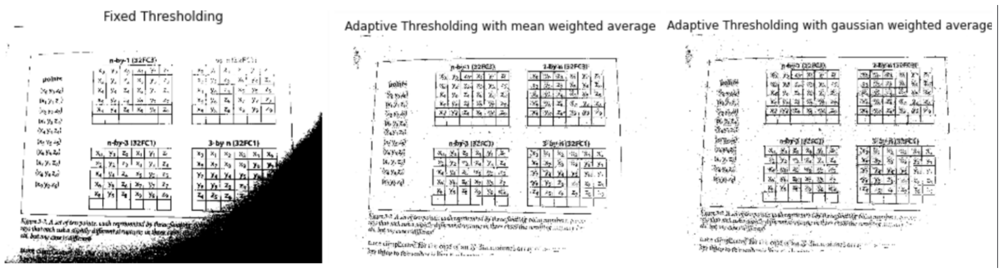


FIGURE 2.7: An image with stark contrasts from dark to bright of various tables on white paper.
[22]



Fixed Thresholding
Adaptive Thresholding with mean weighted average
Adaptive Thresholding with gaussian weighted average

FIGURE 2.8: The leftmost image is fig. 2.7 thresholded with a fixed value of 50. Information in the right lower corner is lost due to the dark shadow. The two following images are of adaptive thresholding, middle image with weighted average and rightmost with Gaussian weighted average, both with a 5x5 window and $C = 10$ and $C = 7$ respectively. The sharp shadow is gone and more information is available from that section. Both images are overall better as well, especially when looking at longer lines in all three images.
[22]

The leftmost image in fig. 2.8 is fig. 2.7 thresholded with a fixed value at 50. Much of the information is kept, except for in the lower right corner, where the shadow was. The two following images in fig. 2.8 are of the same image, but adaptive thresholding has been used, first weighted average and then Gaussian weighted average. The convolution windows is set to 5x5 for both instances and $C = 10$ and $C = 7$ respectively. In both these images the sharp shadow is gone and replaced by white. More information is available, though there are still some resemblance to the shadow(in clear white). The set threshold value could have been changed to try to reduce the shadow, but adaptive thresholding might work better in this instances where the illumination is uneven across the image[22].

Whichever method is used to threshold the image, the new image will have two or more classes. Mostly, the values below the threshold are reduced to 0 and those above to 255. The image is then segmented by whether the pixel has value 0 or 255. All pixels adjacent to another pixel with the same value are given the same label. This is how the image is segmented by thresholding.

2.2.2 Segmentation by Clustering

Clustering is the most common type of unsupervised learning within machine learning. Unsupervised learning can be useful when working with data-sets that include unknown(or unlabelled) data. In a labelled dataset, machine learning will try to predict which label new data should be classified in, but unsupervised learning means that one tries to learn from data without already set labels. A clustering algorithm finds similarities between the unknown data points and groups them together. Thus, it is useful for finding hidden patterns in data-sets.[23]

Clustering in images with the K-Means method

K-means is a commonly used method, mostly because of its simplicity and effectiveness in practise.[24] Images could be defined as a dataset of somewhat unknown data. A large image can contain several million pixels. It would then be strenuous and unpractical to define classes and classify each pixel manually. Using clustering, a computer can both create reasonable class borders and assign pixels to the defined classes.

In image processing, segmentation by clustering can be useful for both finding single or similar objects, but also to compress the image. It can be used to assign pixels with similar color tones and/or lightness together to the same class. This enables the possibility to segment objects or regions in the image by color, or simply to compress the image into fewer distinct colors. This is different to reducing the number of colors linearly, because a clustering algorithm tries to divide the image into colors so that the totality of color change is as little as possible for all the pixels combined. This is typically measured by euclidean distance.

The name K-means originates from the idea to divide n number of observations, into a given k number of clusters. By iteration, the cluster centers are chosen as the mean value of all the objects currently assigned to the cluster, and the objects are reassigned to the cluster with the nearest cluster center. This is repeated for another given number of iterations, or until an acceptable result, given by a threshold, is reached.[24]

The steps are described below and fig. 2.9 illustrates said steps.

1. The cluster centers are chosen as the colored circles. This is done either by random, custom or by chosen methods.
2. All objects (squares in fig. 2.9) are assigned to their nearest cluster center by euclidean distance.
3. The cluster centers are moved to the mean value(or position) of the assigned objects. Notice how the red class only have one object, and the cluster center therefore is moved exactly to the location of that object.
4. Repeat Step 2 and 3 until a given criteria or maximum number of iterations is met.

Figure 2.9 shows an example of the first steps of a simple K-means clustering example.

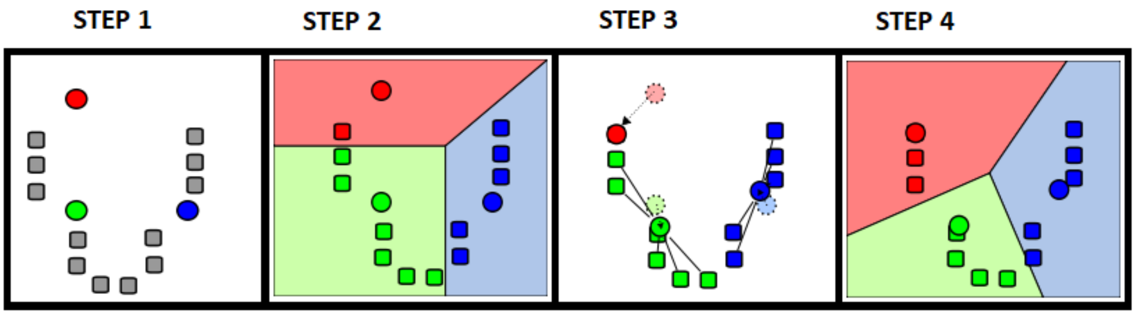


FIGURE 2.9: The figure illustrates the steps in the K-means clustering algorithm. https://commons.wikimedia.org/

The idea of the standard algorithm for the K-means method goes back to 1957[25] and has since then been frequently used. The method is described as relatively effective and efficient, compared to methods of similar purpose. It is therefore especially useful on large data-sets, such as images, which can contain several million pixels and therefore several millions of observations for the algorithm to consider. Nevertheless, the standard K-means method has a pitfall regarding initialization of the cluster means. If these are chosen randomly, it may lead to a poor choice of initial cluster centers, which can make the process take longer and/or lead to a poor result. This has led to the development of several initialization methods over the years.[26]

K-means and clustering by colors, using LAB color space

Clustering in images can be useful when one wants to segment the image by color. Considering color is more complex than only considering lightness. The lightness of pixels can be thought of as a linear "problem". It is harder to segment a picture based on the degree of several individual color values, like RGB. It leads to more possible combinations, and it is more complex to choose a reasonable threshold. In RGB color space, the lightness of the pixels influences all three dimensions. For example, light green and dark green would be located far away in RGB color space, and would be segmented into different classes using clustering.

LAB is an alternative color space, that is commonly used when clustering by colors. LAB also has 3 dimensions, where the L represents the lightness, and the A and B represents the red-green and blue-yellow components of the pixel[20]. This way, it is possible to disregard the lightness and only consider color when segmenting(though, low lighting can conceal the color intensity of an object). This way, different shades of for example green, would be segmented together, regardless of brightness. LAB is therefore a favourable color space to use, when facing problems such as shadows and uneven illumination in images.

To consider, when using the K-means algorithm

The K-means methods, like most method, has its pitfalls. Or at least things to be aware of. Some of these pitfalls are explained in regards to image processing:

1. Colors that take up a lot of area in the image will often affect colors of smaller areas. If there is a lot of blue in the image, and only a tiny bit of green, the green pixels may be assigned to a blue cluster center. The green pixels will of course move the blue cluster center towards green, but since blue is the dominant color, this won't be enough to matter. This could create an illusion of an existing pattern which is not present. The green and blue could be classified as the same color, when they are not.

2. Using clustering to label objects will cause random labelling every time. For example if identifying a fish in an image, it is not expected that the fish will be placed in the same cluster index every time. Actually, the exact opposite should

be expected. Therefore it is necessary to know at least something about the object of interest. For example: its approximate color, where the object is expected to be placed in the image, or the approximate size of the object.

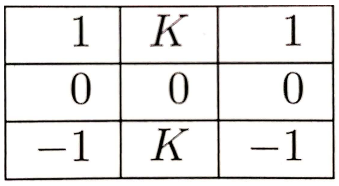
2.3 Edge Detection

Edges can be detected by running a smaller window over the image to calculate certain statistics. To explain this, the smallest possible window is chosen: two adjacent pixels. If the difference in intensity between the two pixels is high, the window is running over an edge. This is an estimate of the first derivative of the intensity function, first in one direction of the image and then in the second direction. Eq. 2.2 shows the first derivative in the x-direction of the image and eq. 2.3 in the y-direction.

$$\delta f_x(i,j) \equiv f(i+1,j) - f(i,j) \tag{2.2}$$

$$\delta f_y(i,j) \equiv f(i,j+1) - f(i,j) \tag{2.3}$$

Many images might contain noise, which would cause every small and irrelevant fluctuation in the intensity value to be greatly amplified when differentiating the image. Generally, an image should be smoothed before finding the local differences. To both achieve smoothing and differentiation in one go, a minimum 3x3 mask is used. This is shown in fig. 2.10.

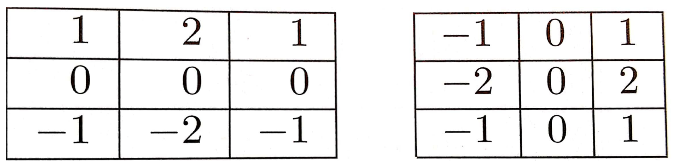


1	K	1
0	0	0
-1	K	-1

FIGURE 2.10: A 3x3 mask which is used for both minimum smoothing and edge detection. In a cascaded manner, this applies first a 3x1 smoothing mask and then a 1x3 differentiation mask, or the other way around. In general, a 2D 3x3 mask will have this form.
[20]

If the K in fig. 2.10 is replaced with 2, it will become the Sobel masks, which is often used to differentiate an image along two directions. These are shown in fig. 2.11

It must still somehow be decided whether a pixel is part of an edge or not. Edges are positions where the image function changes drastically, compared to a uniform surface or background. As the image is a 2D function, the gradient is calculated to find these



1	2	1
0	0	0
−1	−2	−1

−1	0	1
−2	0	2
−1	0	1

FIGURE 2.11: If $K = 2$, fig. 2.10 becomes the sobel masks. [20]

positions, $\nabla f(x, y)$. The gradient of a 2D function is a 2D vector. If, for example, the sobel masks are moved across the image, it will create a gradient vector associated with each pixel. Edges in the image will be places where the magnitude of the gradient vector is a local maximum along the direction of the gradient vector. The local value of the gradient magnitude has to be compared to the values of the gradient calculated along this direction and at unit distance on either side away from the pixel. In most cases, these gradient values will not be known, because they are at positions "in between" the pixels. Then, either a local surface is placed upon the image and used to estimate the gradient magnitude at all interpixel positions where it is required, or the value of the gradient magnitude is computed by interpolating the values of the gradient magnitudes at the integer positions that are already known. This is known as non-maxima suppression, and after it has taken place the values of the gradient vectors are tested against a threshold. Only pixels with gradient values above the threshold are counted as an edge[20].

Fig. 2.12 is an image where a sobel mask has been convoluted across the image to detect vertical edges in the image.

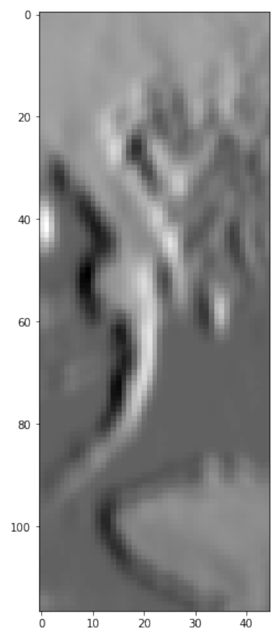


FIGURE 2.12: A sobel mask was run over a salmon to detect vertical edges in the image. The image is of the gills of a salmon.

2.4 Detection of spot-like structures

A Spot-like structure in an image, is a type of blob. A blob is defined as a point or region in image space, that has a higher or lower intensity value than the surrounding region. Every blob should therefore contain at least one local extrema[27].

By definition, blob detection can be regarded as a type of interest point detection[28]. Interest points are used in image matching, to find corresponding points in images of the same object. Historically, interest point detection started with corner detection to extract robust features from images, in order to track moving objects in video. Interest points could however also be used to match and identify objects over longer time, and in different scenes.

Quoting T. Lindeberg in [28]: Interest points should:

1. *have a clear, preferably mathematically well-founded, definition,*
2. *have a well-defined position in image space*
3. *have local image structures around the interest point that are rich in information content such that the interest points carry important information to later stages,*
4. *be stable under local and global deformations of the image domain, including perspective image deformations and illumination variations such that the interest points can be reliably computed with a high degree of repeatability*
5. *be sufficiently distinct, such that interest points corresponding to physically different points can be kept separate*

Blob detection has not allways been regarded within the class of interest point detection, but it has gradually been realized that there is no solid reason to exclude it, especially because most blobs has a ”well-founded” position in the image space, often defined by a local extrema or centroid.[28].

2.4.1 Blob detection methods

There are several approaches to blob detection. Some common approaches is the use of Laplacian of Gaussian, Grey-level trees, difference of Gaussians (DoG) and the determinant of the Hessian.

In this project, especially the ideas of the LoG and Grey-level trees are used. The spesific methods are however not directly implemented, but the methods that are used, are based on the same ideas.

The method developed in 3.7.2 Dark Spot Detection method shares the same basic principle of LoG, by applying a Gaussian blur for enhancing blob-like structures, and the method in 3.7.1 Simple Blob Detection Method shares the same approach of using multiple gray values to define blob regions, such as Grey-level trees.

LoG

The LoG, is one of the oldest and most common approaches for blob detection[29]. A Gaussian kernel is applied to an image to enhance blobs, and the local extrema in every blob is detected by the Laplacian. This becomes the interest point of that blob. If the kernel shape and size fits the blob, and the blob is dark or light, a local extrema will be located towards the center of the blob[29].

Fig. 2.13 shows an example of a Laplacian distribution. This distribution is circular, and would therefore perform best when applied to circular structures of the same size.

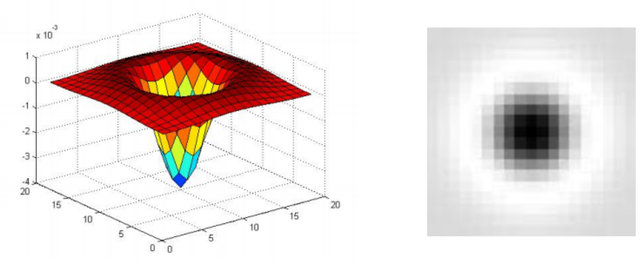
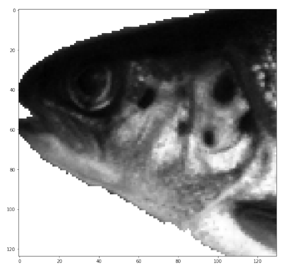


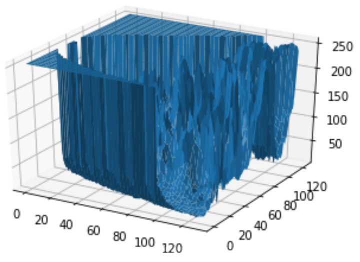
FIGURE 2.13: A 3D and 2D representation of a circular Laplacian distribution. Image from: www.uio.no

Hence, a basic problem of using a Gaussian kernel in LoG and similar methods, is choosing a fitting kernel size. The best results occur when the kernel size fits the size of the object that is being detected. If the goal is to detect blobs of different and/or unknown sizes, a multi-scale approach is preferred[30].

Fig. 2.14 and fig. 2.15 illustrates the enhancement of spot-like objects when a Gaussian blur is applied.

Comparing the graphs in fig. 2.15 to the ones in fig. 2.14 it is evident that the amount of local extremas are reduced. This makes the remaining extrema enhanced. By filtering only the dark extrema of these images, features such as the dark spots of the fish can be extracted as interest points. However, a too strong/large blur could wipe out small extrema and fuse nearby blobs together. This is why scale variations are important.


(a) Original Image


(b) 3D plot of values plot

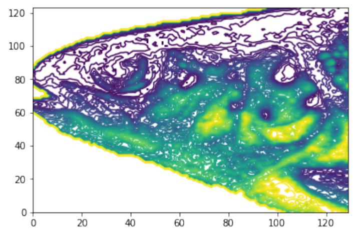
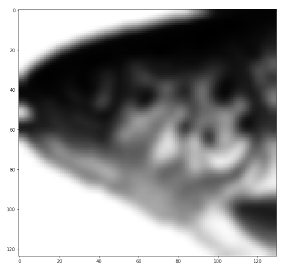
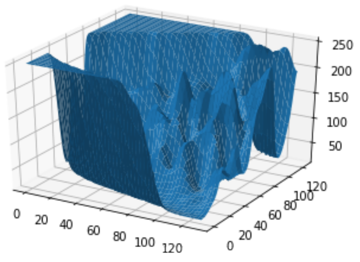

(c) Contour plot

FIGURE 2.14


(a) Applied Gaussian blur


(b) 3D plot of values

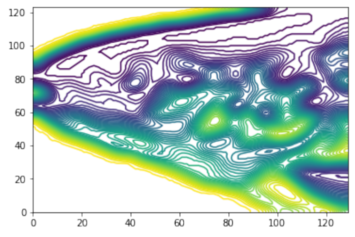

(c) Contour plot

FIGURE 2.15

Grey-Level blobs and Grey-level trees

The ideas of Grey-level blobs is similar to the LoG; that every blob is a region that should be associated with at least one local extrema. In addition, the extension of the blob's area is decided by the gray-level around it. The blob are allowed to grow until merging with another blob[27]. Fig. 2.16 illustrates how these regions are defined.

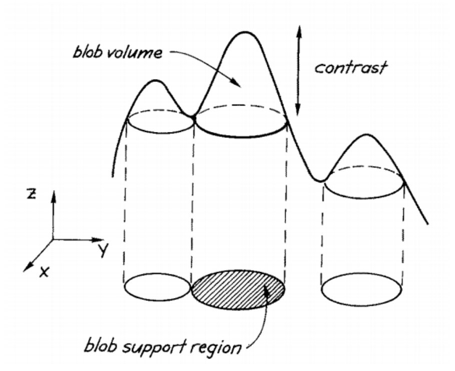


FIGURE 2.16: Grey-level blob detection.
[27]

Grey-level trees are constructed by Grey-level blobs as its "leaves". When the gray-level threshold is reduced or increased, the blobs are fused together, as illustrated in 2.17.[27]

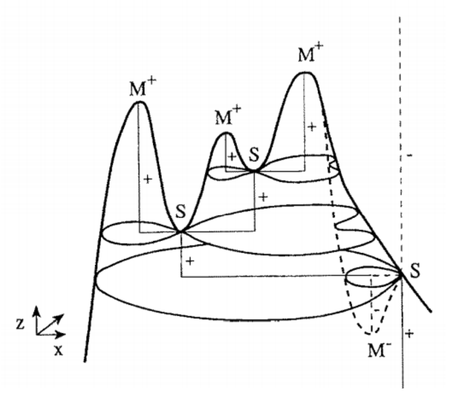


FIGURE 2.17: A Grey-level tree with blobs as "leaves". [27]

2.4.2 Blob features

When blob regions are defined, for example by combining many layers of blobs, the detected blobs in each region can be combined to return suitable blob features, such as an interest point and shape descriptions. By using the shape features, different spots could be filtered or categorized by features such as area, gray value, circularity, inertia and convexity. All these parameters are used when filtering blobs in 3.7.1 Simple Blob Detection Method. They are also thoroughly experimented with in 4.3.1 Spot detection by Simple Blob Detection on data-set 1.

Fig. 2.18 shows an overview that intuitively explains the blob features. The features are further described below.

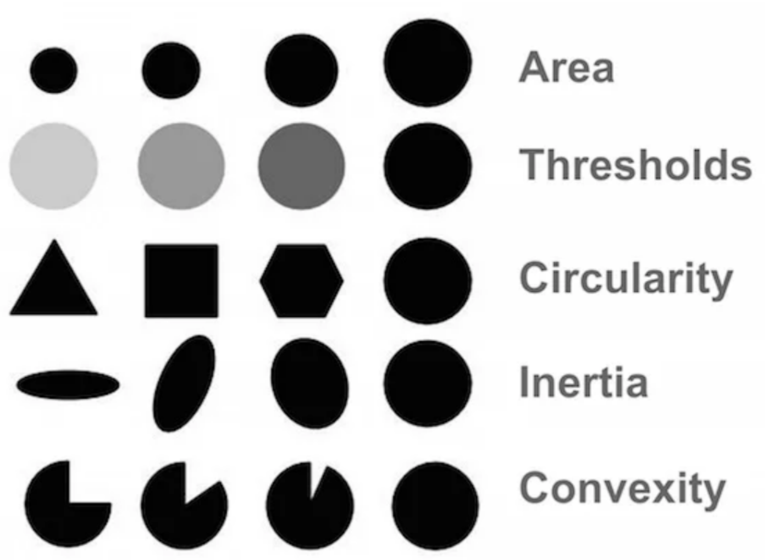


FIGURE 2.18: Blob shape features. [31]

The area could be used to only consider blobs of particular sizes. [31][32]

The threshold parameter can be used to detect blobs of certain gray value.

Circularity measures how close to a circle a blob is, for example a hexagon is closer to a circle than a square. Circularity is defined as eq. 2.4. A perfect circle has value 1, while a square would be 0.785.

$$c = \frac{4\pi A}{(\text{perimeter})^2} \tag{2.4}$$

Inertia ratio measures how elongated a blob is. A circle has the value 1, an ellipse somewhere between 1 and 0, while a line has the value 0. Increasing the inertia ratio will discriminate against blobs that are elliptic. This is illustrated in fig. 2.19.

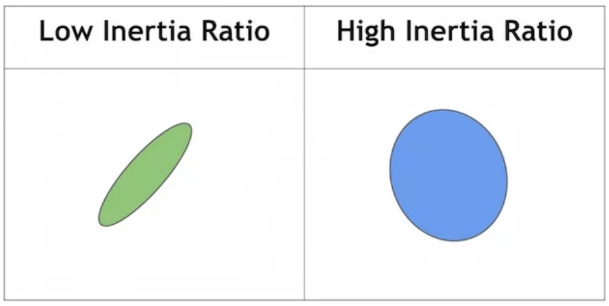


FIGURE 2.19: The inertia measures how elongated an area. [31]

Convexity is defined as shown in eq. 2.5.

$$\frac{\text{Area of blob}}{\text{Area of convex hull}} \tag{2.5}$$

A shape's convex hull is the tightest convex shape that completely encapsulates the blob. This is illustrated in fig. 2.20. A value close to 1 will discriminate against blobs that have concave shapes or concave section in the body of the blob.

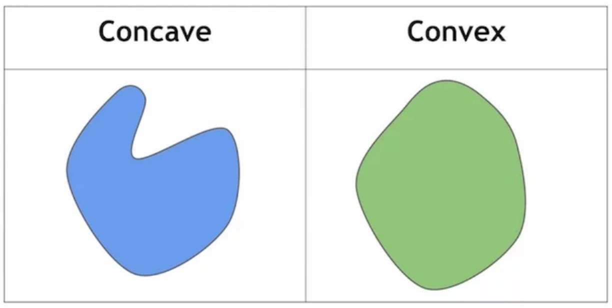


FIGURE 2.20: The difference between a convex and a concave blob. [31]

Chapter 3

Methods

This chapter presents methods for locating the salmon in the image, extracting the head and tail, locating the pectoral fin, finding the gill opening, spot detection and a method for individual recognition. First is a presentation and information about the data, followed by the methods mentioned above.

This is shown in the flowcharts below:

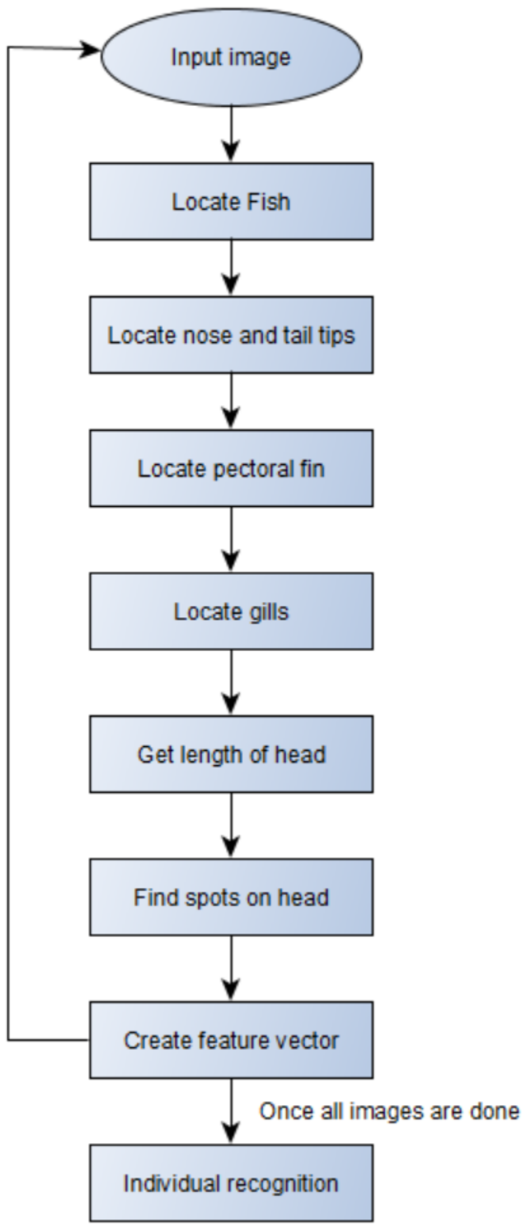


FIGURE 3.1: The process each image is put through to detect features, which are in turn used to create a feature vector for each fish in an image. Once this has been done for all images, the salmon are tried to be recognized from one image to another.

3.1 Presentation of data

Two different data-sets of bred Atlantic Salmon, are used in this project. One was chosen as the main data-set, that was used for developing methods and thorough experimentation. The other set was only used for testing the versatility of the most important methods. The data-sets are further referred to as:

1. Data-set 1: The main data set.

2. Data-set 2: The secondary data set.

On the following pages, the two data-sets are presented and compared.

Main data-set - 16 months old smolts - bred by MOWI

On the 12th of March 2019, 203 MOWI smolts were photographed specifically for this project. Three images were taken for each fish. A Point Grey FL3-U3-13S2C-CS camera was used. The images were captured and saved in TIF format, and the camera color processing algorithm was set to "Weighted Directional Filter". The dataset was cleaned, and 534 images of 195 fish were defined as favourable to use for this project. Appendix A contains list of every image that was not included, and arguments why. Generally, it was because the fish was not immediately surrounded by a blue background, or that the fish was in fast motion (blurry) or in an unfavourable angle, as the preferred angle was somewhat horizontal, with the nose facing to the left.

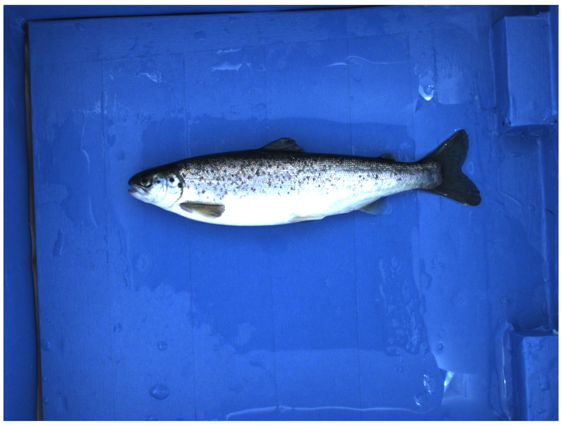


FIGURE 3.2: An example image from the main dataset. These images display an Atlantic Salmon smolt, surrounded by a blue background.

The images of this data-set displays relatively good quality and consistency of the scene. Also, the whole fish was displayed in all of the images. These are the main reasons that this was picked as the main data-set. The approach for capturing these images is described in 3.2 Image Capturing of Atlantic Salmon, page 36. An example image is shown in fig. 3.6.

Secondary data-set - 12 months old smolts - bred by IMR

This dataset was acquired from IMR's biologist Ole Folkedal. The images were originally captured in context with their project on visually recognising salmon by their spots on

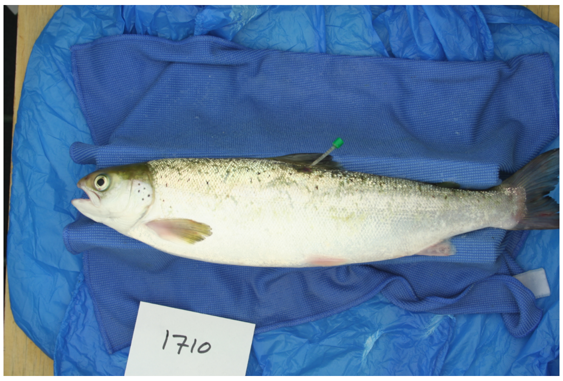


Figure 3.3: An example image from the secondary data-set. These images displays Atlantic Salmon smolts, bred by IMR. The fish are immediately surrounded by a blue cloth-like background, which contains variation due to curves of the fabric.

the gill cover over time [14]. The images were not intended to be used for computer vision, only for manual comparison. Fig. 3.3 shows an example image.

The data-set contains 246 images of 246 fish. The images were taken in similar fashion as fig. 3.3, but the images does often not include the whole fish. The camera used was a Canon EOS 550D. The camera, nor the fish, was not completely static in all images, and thus, the scene varies a bit, and the fish is a bit blurry in many images.

Discussion and Comparison of Data-set 1 and 2

Table 3.1 shows a comparison of important traits of the two data-sets. This serves as an overall overview of the data-sets, and not a description of each individual image. The images vary within the set, but this serves as a general distinction.

Comparison of the data-sets		
Data-set	1	2
Skin color tone	Darker on top	**Overall light**
Separability from background	**Easily**	Harder
Parts displayed	**Whole fish**	Misses tail fin
Static camera position	**Yes**	Nearly
Consistency*	**Good**	OK
Contrast	Not favourable	**Better**
Pixel resolution	1280x960	**3456x2304**
Avg. No. of spots on gill cover	1.90	**2.34**
Avg. size of fish	192+-14mm	**209+-12mm**[14]
No. of fish	203	**246**
Images per fish	**3**	1

Table 3.1: Comparison between the two data-sets. These comparisons are relative to each other, and is done in order to highlight the advantages of each data-set. As can be seen, each data set has it's own advantages. These are highlighted with broad text. Eventually, the advantages of data-set 1 were considered as favourable for developing methods. *Consistency means how persistent the images are in similarity, like fish-placement, focus and same background between images.

As can be seen in table 3.1, both data-sets has their advantages. Data-set 2 has advantages that could cause better results, such as lighter skin color of the fish which then would lead to better extractions of features from spots, eyes and fins. The images are also taken with a better camera with high resolution. However, these advantages were considered as less desirable than the advantages of data-set 1. In practice, one should not expect that the fish has such brightness. Salmon in general look more like the ones in data-set 1. Also, large resolution is good, but often prove unfavourable due to longer processing time. It would be more realistic to challenge the methods by applying them to images of lower resolution. In machine vision, images are often resized to save time. The fact that the images in data-set 1 proved to be consistent and the whole fish is displayed in every image, resulted in the choice of using data-set 1 as the main data-set for developing methods. Since 3 images was taken of each fish, this better enables the opportunity to experiment with recognition/matching between images in later stages. Data-set 2 is used to test the versatility of the methods.

Conclusion

Due to the favorable advantages of data-set 1, it was chosen as the main data-set for developing and testing methods. Data-set 2 has other advantages and was kept as a testing data-set.

3.2 Image Capturing of Atlantic Salmon

Images of Atlantic Salmon were captured when the fish was anesthetized and taken out of water. In addition to standard capturing problems, capturing images of Salmon had a couple of specific difficulties.

Requirements / Problems

1. Capturing needs to be done fast, as the fish should not stay too long out of water. Workers at MOWI suggested as a rule of thumb that 1.5 minutes should be the absolute limit, and that under 30 seconds was preferable. Also, the images needs to be captured before the anesthetic ends, as a conscious fish is quite wriggly.
2. The scales of the fish are very reflecting. Therefore, diffused light is preferable to obtain even lighting and avoid sharp shadows (See 2.1 Optics).
3. The fish is generally very dark on the top, and light on the bottom. This can make it hard for the camera to adjust it's gain value. This may cause a dark image, or blinded regions in the image, where all pixel values are maxed out, and one therefore will miss contrasts that are actually there.
4. Generally, fish factories are tight in space, and there is a lot of water flowing around. The "studio" therefore needed to be small and able to withstand water.
5. All the equipment was transported in a car, a little boat, and then carried inside the factory. Therefore it was designed to be robust and easily portable.

Construction of the "studio"

With limited funds, a photo-box was built by prototyping in order to fulfill the requirements.

A cut of the model is illustrated in Fig. 3.4, while fig. 3.5 shows the inside of the box (before the blue background was installed) and the setup at the factory. The reasons of the implementation is explained in the following points:

1. Normal white printing paper was attached to the insides of the box, in order to achieve diffused light reflection.(See 2.1 Optics))

2. White paper were also placed in front of the LED lists as a medium to spread the direct LED light.

3. LED lists were attached about 17cm above the ground plate, so that the lightning would get some angle towards the ground. They were not placed any closer, because this seemed to cause lower lighting for the upside of the fish.

4. In the end, it was chosen to use a blue background for easier extracting of colorless objects like an Atlantic Salmon. This causes worse diffuse reflection from the ground, and it could be discussed whether white should be used instead. The dark-ish blue background may have made it difficult for the camera to capture all the contrasts, which can be seen in the resulting image in fig. 3.6.

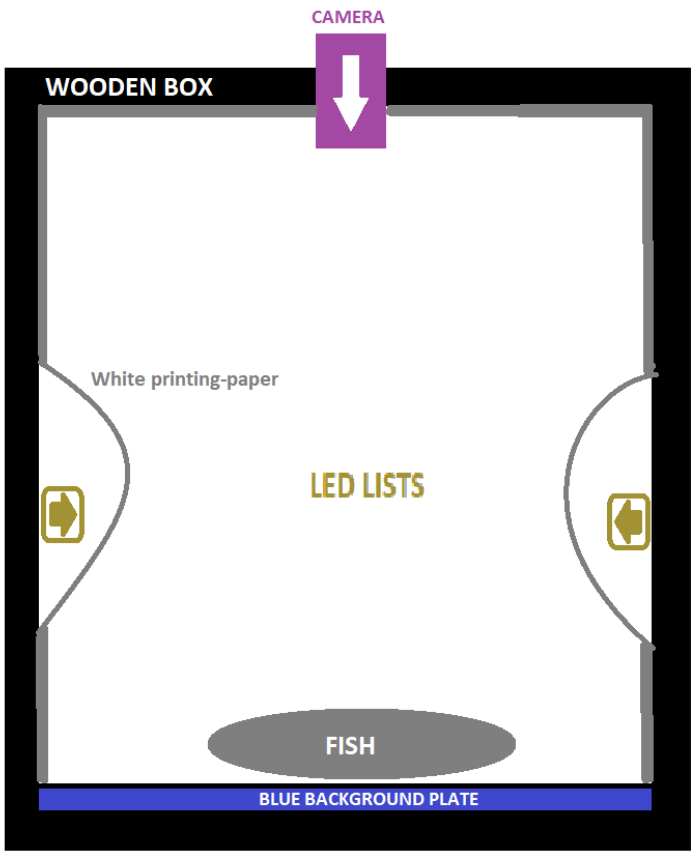


FIGURE 3.4: The design of the fish-photobox prototype. This is a vertical cut. The dark grey color represents the normal white printing-paper. The camera, shown as purple, was attached in a hole from above with Field Of View pointing down. Yellow, represents the LED lists, attached 17cm above the ground plate, pointing inwards. Also, the figure shows how the fish is placed relative to camera, on a blue background.

A result example can be seen in Fig. 3.6. As wanted, there are no sharp shadows around the fish. It also seems relatively even illuminated, and there are little direct reflection from the scales of the fish. A problem is that the camera seems to be unable to detect

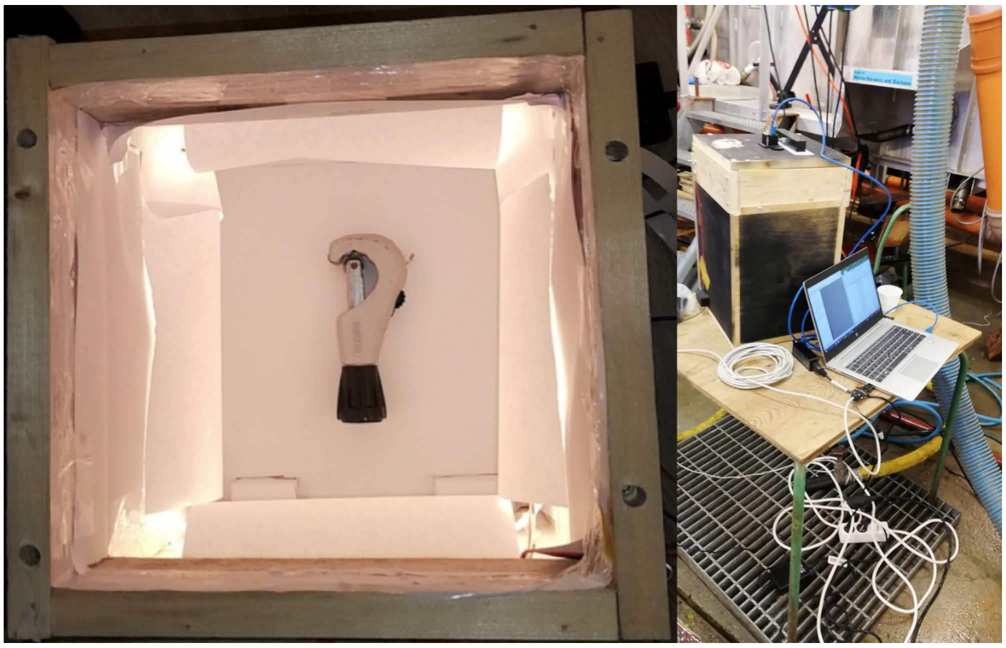


FIGURE 3.5: Left: An above viewing of the photobox, with no lid. The LED-lists are attached around the box, so that there will be light from all angles. The test object at the center receives lighting from every angles, which causes little shadow and even distribution of the light on the object itself. This was before the blue background as shown in fig. 3.4 was inserted. Right: Setup of the image capturing system in the smolt-factory.

the contrasts on the downside? of the fish because of blinding. Simultaneously, it seems to have a hard time in capturing details of the dark upside, as we can barely see the eye. These are opportunities for future work.

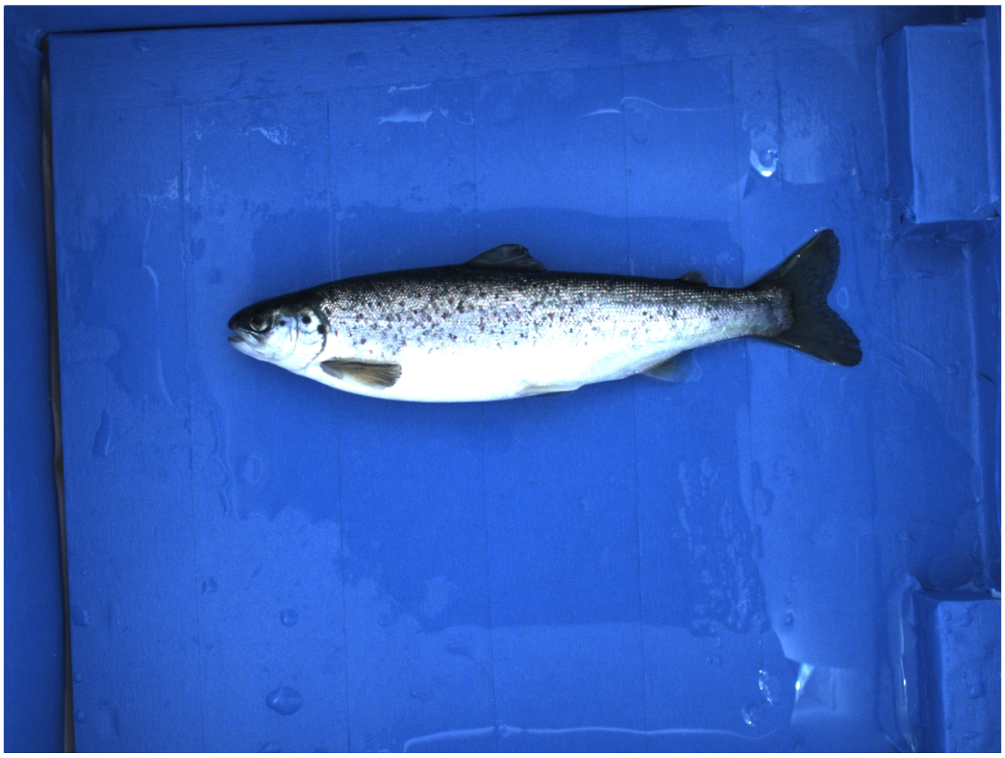


FIGURE 3.6: One of the resulting images from the capturing setup.

Improvents / Future work

The main problem seem to be the lack of contrast details. Three suggestions are listed below as possible solutions to fix this. A combination is probably good.

1. Experiment with a lighter background, for example white. This would prevent the later use of simple color segmenting methods, but there are other ways to segment an image.
2. Steinar Laudal from Mestec suggested to increase the lighting that hits the dark side of the fish, For example by three times the intensity. This was not done in this instance, because it was originally planned to rotate the fish.
3. Use a better camera, which better captures the contrasts.

3.3 Extraction of the fish in the image

This and the next sections shows how the images are processed in order to extract information.

First, the foreground object, or the object of interest in the image, has to be detected and extracted from the image. In this case; the fish. The background should be removed, such that only the salmon is present.

Second, features from the fish itself are extracted. These are locations and areas of nose and back fins, front fin, gill opening and spots. The features should be as exact as possible, since a higher accuracy is needed for potential individual recognition than other tasks, such as differentiating between species.

Step one is to extract the foreground object of the image, the fish. Methods for achieving this depends on the image. Different methods has to be developed for different types of images. The methods can be general and usable on almost any kind of image, or they can be specific, where it is demanded that the images are captured in a certain way. Since the latter is less complex, and because it is assumed the setup of image capturing can be specified also in the future, the method developed was not general, meaning it cannot extract fish from all images. Instead the method was tailored to fit the setup of the images acquired for this report.

The method that extracts the fish in the image, is presented in steps below. The method works for both data-set 1 and data-set 2, with a slight modification. This is explained in point 6. The code for this method can be found in appendix D.1 Extract Fish in image on page 140.

Fig. 3.7 shows the input image.

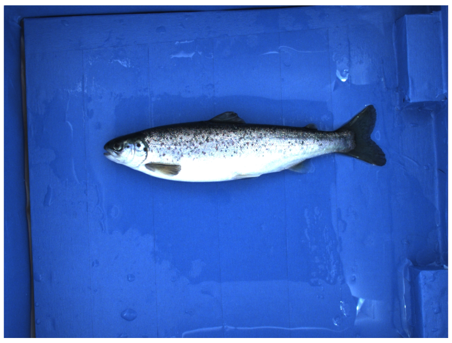


FIGURE 3.7

1. Convert image to LAB color space(See 2.2.2 K-means and clustering by colors, using LAB color space). Fig. 3.9 and fig. 3.10, shows that the color components of the fish is quite different to the background. The background contains high levels of both blue and green, while the fish is more neutral.

2. Create a 2D Array of the A(red-green) and B(blue-yellow) components of the image.

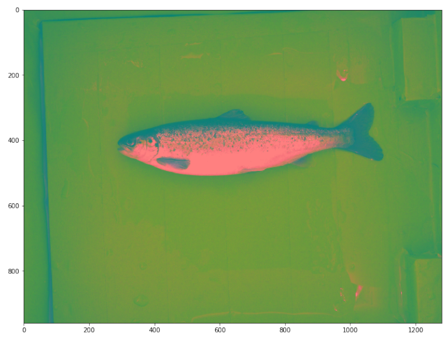


FIGURE 3.8: Display of the image in LAB color space.

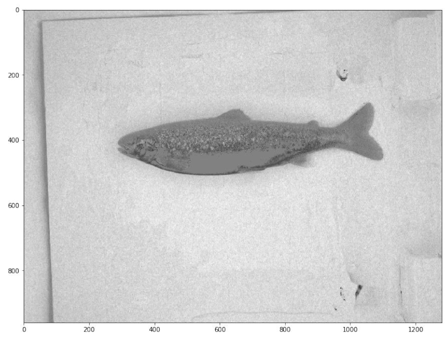


FIGURE 3.9: Display showing only the A, or the red-green component of the image. A white background represents a strong green component.

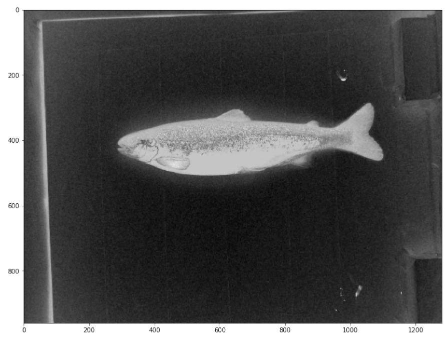


FIGURE 3.10: Display showing only the B, or the blue-yellow component of the image. A black background represents a strong blu component.

3. Create initialization vector for the K-means method. This can and should be done because the locations of the cluster centers are approximately known. This makes the implementation faster and more robust (See 2.2.2 To consider, when using the K-means algorithm). The initializing vector is: [[128, 116], [150, 69]].

4. Run K-means algorithm to find two cluster centers. The method finds two distinct colors.

5. Assign all pixels to their nearest cluster center. Since only two cluster centers are chosen, the image is binarized, as shown in fig. 3.11

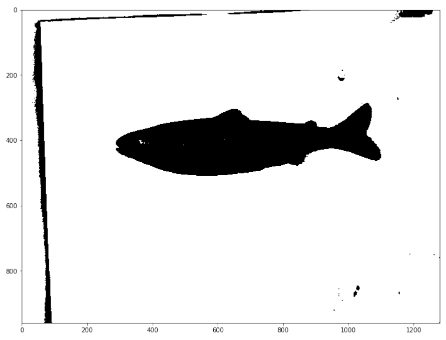


FIGURE 3.11: Binarized image, partitioned into two segments.

6. This point is different for data-set 1 and data-set 2. Data-set 1 is processed as in (a), and data-set 2 as in (b).

   (a) If there are more white pixels (with value 1), invert the image, assuming the background takes up most of the area. This is to ensure that the fish contour is white. See fig. 3.12

   (b) If the centre pixel of the image is black (with value 0), invert the image, assuming the fish contour covers this pixel. This is also to enure that the fish contour is white.

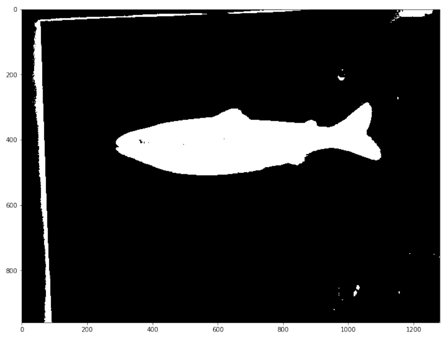


FIGURE 3.12: This image shows the labelling is in this case inverted. It is preferred that the fish gets the label "1".

7. Find all the external contours of the binarized image and keep only the largest, assuming this is the fish. See fig. 3.13

8. Fill holes in the contour. See fig. 3.14

9. Output new image, showing only the fish. See fig. 3.15

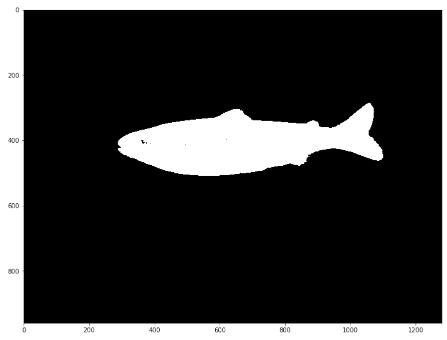


FIGURE 3.13: Only the largest contour, which should be the fish.

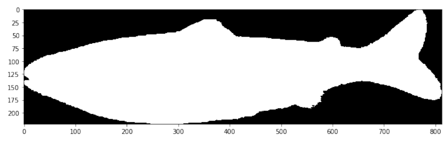


FIGURE 3.14: Only the largest contour, where holes are filled.

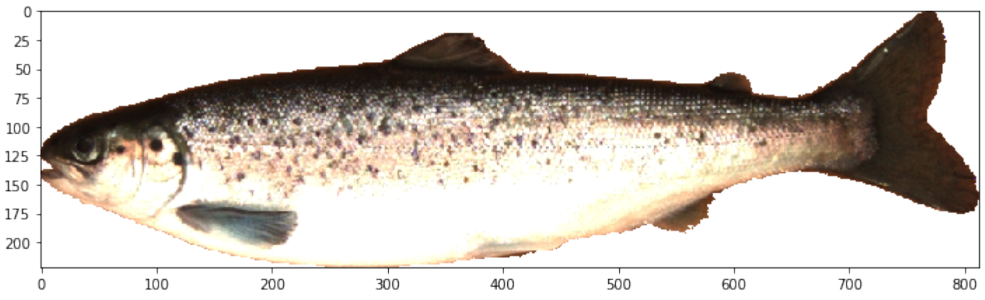


FIGURE 3.15: The fish has successfully been extracted from the image, and this is the image to be used for further experimentation.

Results

The method successfully extracted all 534 fish from data-set 1. This was expected, as the data-set was cleaned for blurry images, and images where the fish was not immediately surrounded by blue background. The methods would not perform as good on such images. "Success" here, means that the fish has been extracted at a subjectively high standard, as the results were measured visually.

To evaluate this method in a more neutral and statistical manner, an algorithm could be used, that compares the actual contour of the fish, with the detected contour. Then, the number of excess pixels, and missing pixels, could be used to evaluate the method and compare it to other methods. It would however be time consuming to manually mark the contour of every fish. This is considered as future work.

3.4 Nose and Tail Tips detection

A method which successfully locates the nose and two tips of the tail fin is desired for the following reasons:

1. Determining the orientation of the fish in the image. Like which way the nose is pointing, and which part of the fish is up or down.
2. The placement of nose tip and back fin tips can be used as reference points for other methods.
3. Estimate the length of the fish.
4. Ability to check how much the fish is bent, which could contribute to better length estimation when the fish is bent.

In this project, the nose point is especially interesting, because it could be used as reference to the extract a ROI around the head of the fish, to further detect the spots on the gill cover of the fish. Also, if it is precise enough, it could be used as a reference point to spots, and the front fin, to extract structural features.

The nose and tip of back fins are located on the outline of the fish. The idea is that these can then be extracted by examining the distance from the center of the salmon, to every point on its outline. The code for this method is found in appendix D.2 Code: Find nose and Tail Tips on page 145. It was implemented as shown in the following points below:

1. Input image is the contour of fish like seen in fig. 3.16

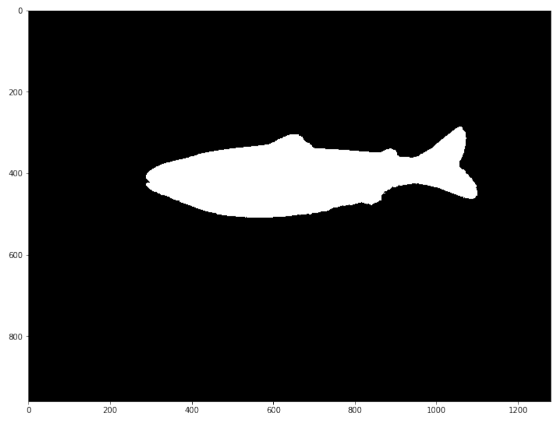


FIGURE 3.16: Contour image of the fish. It is clearly visible and the shape is intact.

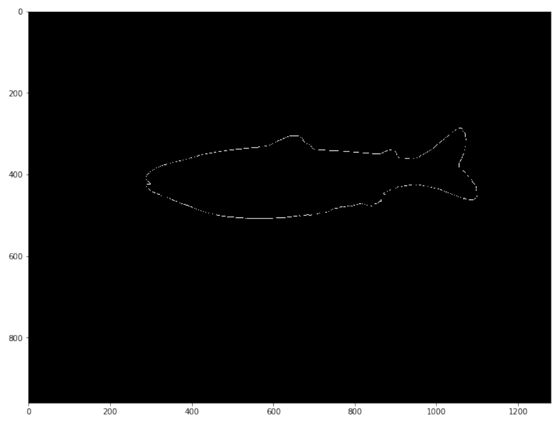


FIGURE 3.17: Same image as before, but now only the contour line is shown.

2. Create outline contour of fish. See fig. 3.17

3. Find all distances from the centroid of the fish to outlined points. A plot of the distances is shown in fig. 3.18 as the leftmost graph. The vector is shifted to the global minimum(rightmost graph).

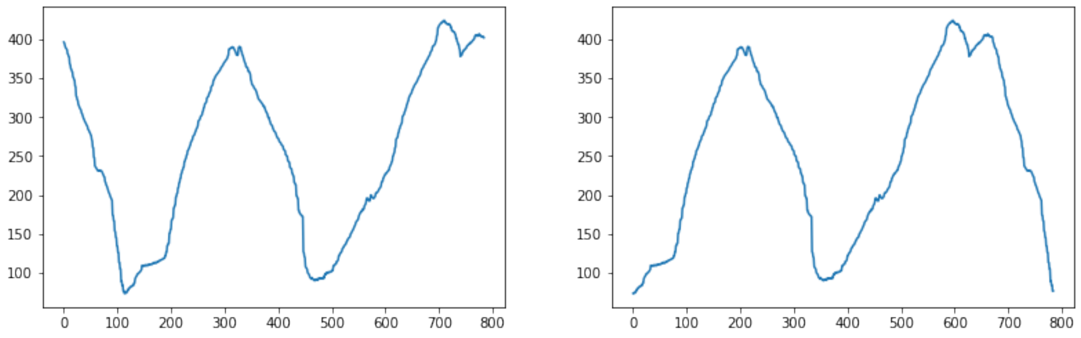


FIGURE 3.18: The leftmost graph computes the distance from the center of the fish to the outline points. The rightmost graph is the same, but shifted to the local minimum.

4. Calculate mean distances over 7 points, to get less noise. Shown in fig. 3.19

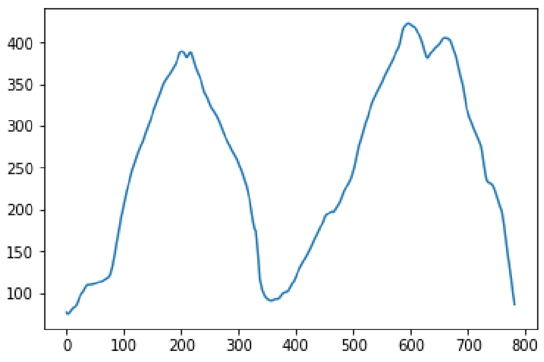


FIGURE 3.19: The mean distance is computed to reduce noise.

5. Find all local maximas in the distance vector.

6. Keep only the three largest local maximas. This should be points close to the nose and two back fin tips, though it is still unknown which points represent what.

7. It is preferred to know what each point represents. First, the original image of the fish is grayscaled and binarized to find the dark parts of the fish, which is on the top side. See fig. 3.20.

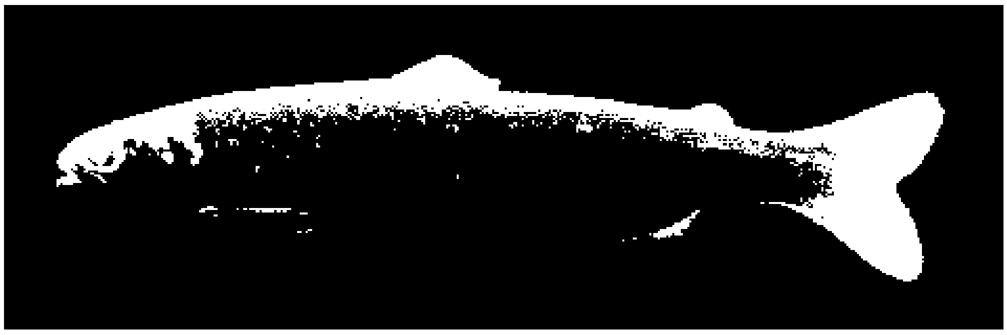


FIGURE 3.20: An binarized image inverted, such that the dark parts of the fish is highlighted.

8. Find the centroid of the dark parts of the fish. Yellow dot in fig 3.21.

9. On the outline of the fish, find the closest point to this centroid. Purple dot in fig. 3.21

10. In the end, the labels are defined by their distance to the outline point(Purple dot), as shown in figure 3.21. A more detailed explanation is given in the image text.

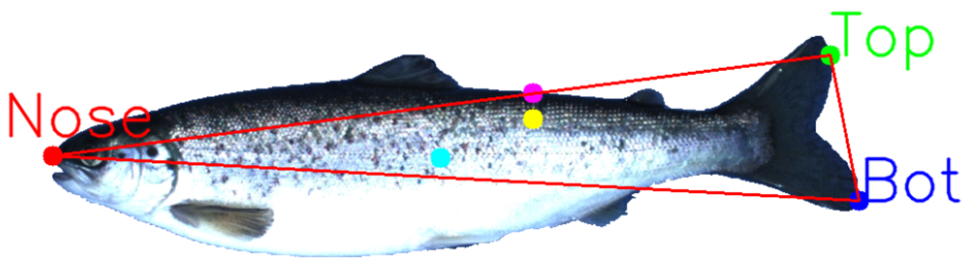


FIGURE 3.21: The nose, top and bottom tail fin tips are localized. The turquoise dot represents the centroid of the whole fish, while the yellow dot is the centroid of only the largest dark area of the fish. The purple dot is the closest point to the yellow dot which is also placed on the outline of the fish contour. It is only by coincidence that it is placed on the red line in this example. The distance from this point is used to define the other points. The top tail fin should be at the nearest point, the bottom tail fin should be at the second nearest point, and the nose should be at the point furthest away.

Results

Considering the 534 remaining in the dataset, the results were 100% accurate for the two tail fins. The nose were in some cases mistaken for the chin. However the orientation of the fish is found. The method could be adjusted to perform better on the nose tip, but as the method has served its main purpose, it is left like this.

Future work would be to better detect the nose instead of the chin. A suggestion could be to test which of the two points have darker neighbouring pixels, as the nose tip is generally darker than the chin.

Reasons of Failure

Even tough these experiments shows 100% accuracy for the image set used, it is not certain that the top and bottom fin would have been successfully identified in cases where the fish is more bent. If the bottom fin tip somehow get closer to the contour point, this would be identified as the top back fin. The method could be developed to be more robust in this area.

3.5 Pectoral Fin Detection

The pectoral fin is the fin located around where the head meets the body. For each specimen, the fin will look different and have different placement on each individual, and can therefore serve as a key component in individual recognition. Fig. 3.22 shows the location of the pectoral fin on a salmon.

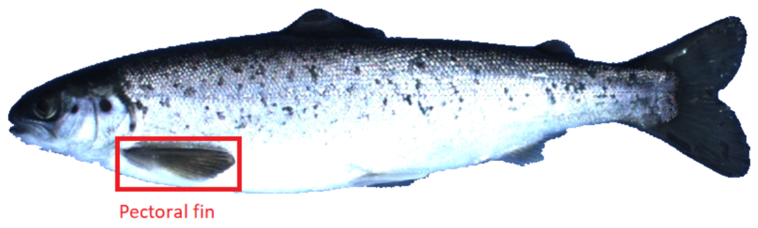


FIGURE 3.22: The location of the pectoral fin on a salmon. The fin is generally in the same spot on each fish.

Before the fin can be used as a tool for recognition, it must first be located automatically on each fish. The method developed is shown in fig. 3.23 and the code can be found in D.3 Code: Fin Detection, page 149. One assumption is made: the background is uniform, no other objects than the fish of varying intensity.

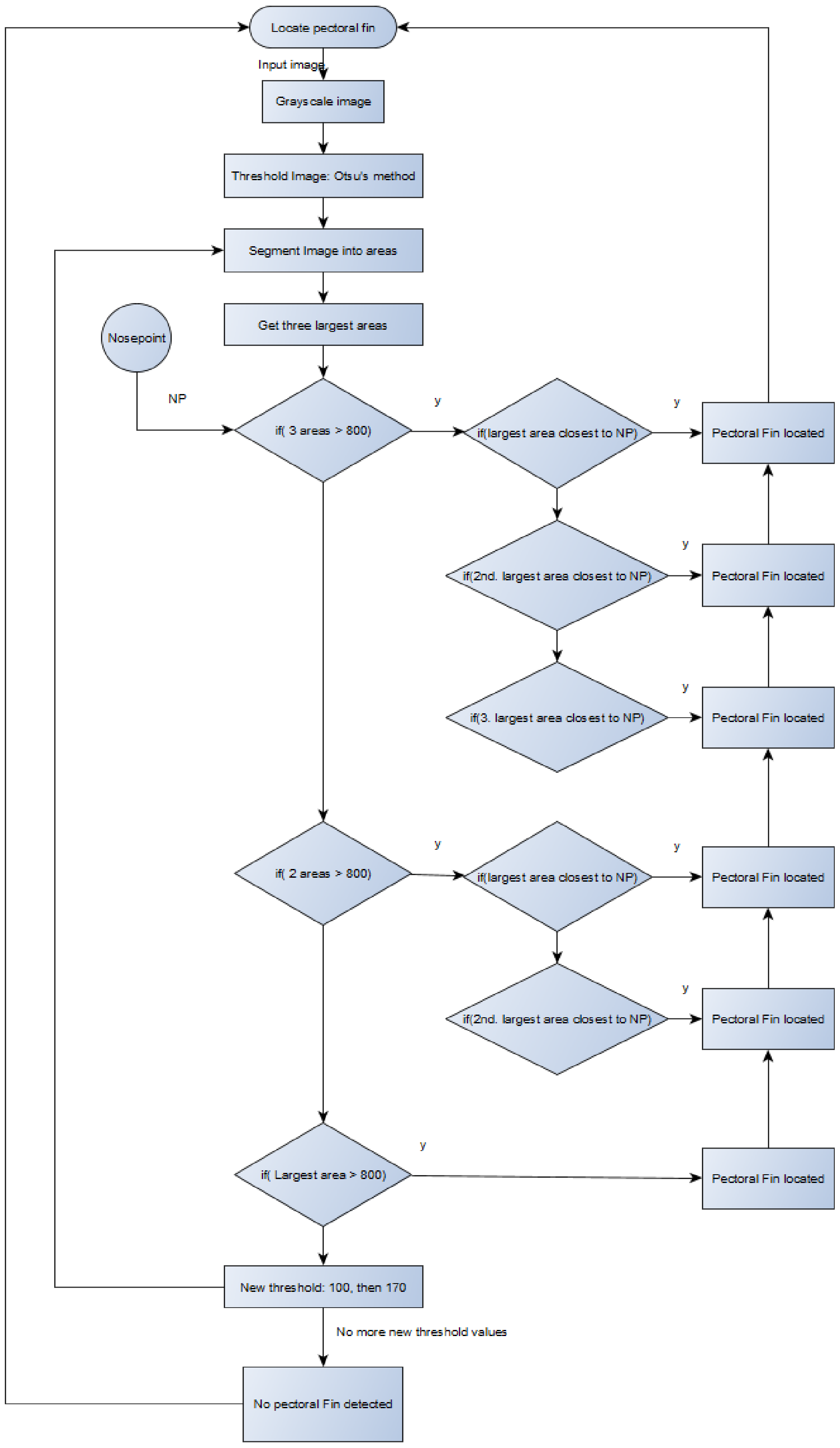


FIGURE 3.23: The flowchart describing the algorithm created to locate the pectoral fin.

The algorithm(named: Fin Detection) uses several measurements to ensure correct location of the pectoral fin. Segmentation by thresholding is used and the method tried to binarize each individual image after it was grayscaled was otsu's method. Otsu's method was primarily used because it would place the threshold between those two peaks, as discussed in Otsu's Threshold page 18. It singled out the pectoral fin in most cases.

Hysteresis thresholding was also considered, but in most cases the pixels around the pectoral fin has very different values than the fin itself. In other words: it is distinct and not generally not fuzzy around the edges.

After binarization, the images were segmented. In most cases, the pectoral fin would be the third largest area. This is visualized in fig. 3.24.

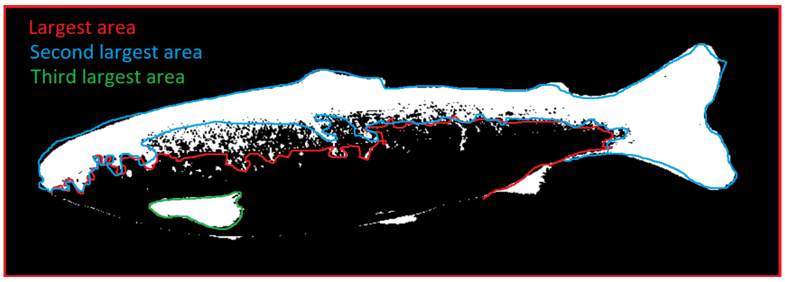


FIGURE 3.24: After segmentation, the idea was that: the background and part of the salmon's stomach will be the largest area, its back and caudal fin the second largest and generally the pectoral fin will be the third largest area

Gathering the third largest area when thresholding with Otsu's method caused misclassifications on images where the fin wasn't singled out. Some of these images are shown in fig. 3.25.

Issues arose because of two different factors: the threshold did not single out the pectoral fin and/or the pectoral fin was not the third largest area. To tackle this, the nose point detected by the method detecting the nose and tail tips was used. The algorithm would gather the third, fourth and fifth largest areas. Using the distance from the nose point it would chose the one of the three areas closest to the nose of the salmon. This eliminated classifying the anal fin as the pectoral fin. In a few cases, small areas(smaller than the pectoral fin) on the gill and head were closer than the pectoral fin and cause misclassifications. This was dealt with with a threshold for the three areas: the correctly classified images were used to make a vector containing the sizes of all correct pectoral fins, to get a measurement of their general size. A threshold of 800 pixels were implemented, discriminating against all located areas below the threshold.

If all three areas consisted of more than 800 pixels, their distances were computed and the one closest to the nose was chosen. Same procedure if two areas were larger than 800 pixels. If only one was, it was chosen as the pectoral fin.

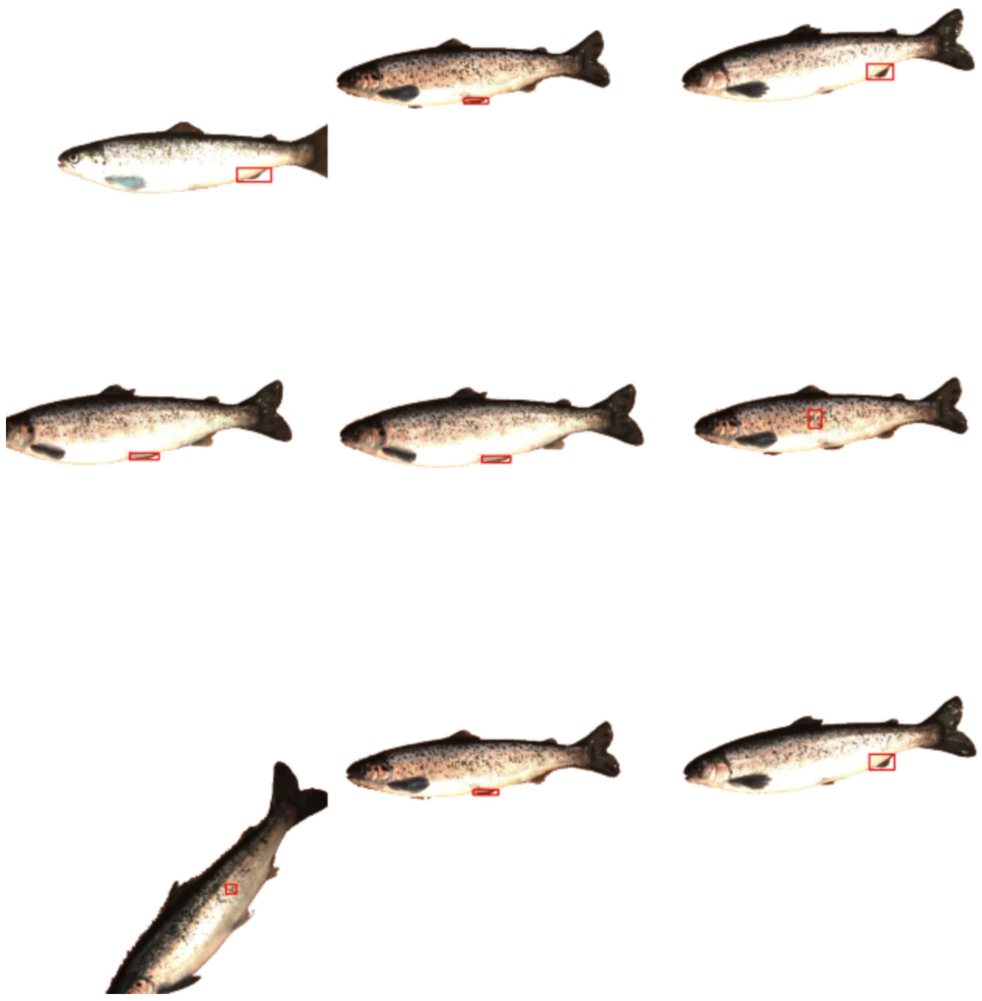


FIGURE 3.25: Misclassified images where the pectoral fin was not detected. Mostly, other fins were detected.

Should no areas meet the 800 pixel threshold, the image would be binarized again, but with a new threshold of 100(instead of otsu's threshold). Intensities at or below 100 would be raised to 255 and those above 100 were reduced to 0. The algorithm then segmented and gathered the three largest areas and ran the same process as before. If still none of the third, fourth or fifth area met the 800 pixel threshold, the image was binarized one last time at 170. The same procedure was performed again. If no pectoral fin could be located, the algorithm would print "No pectoral fin detected" on the image.

3.6 Locating the gills

Locating the gills could be an important step in recognizing fish. It could help with applications like:

1. Finding the orientation of the fish
2. Set Region of Interest(ROI), for example around the "head". More specifically: from the gills to tip of nose. This is useful for further examination of the spots on the gill cover.
3. Draw features from the gills(for example, the length of the "head").

If the gills could be defined visually, then they could be separated from the surrounding area on the fish. A suggestion on how to visually define the gills follows below. Fig. 3.26, shows an image containing a gill opening.

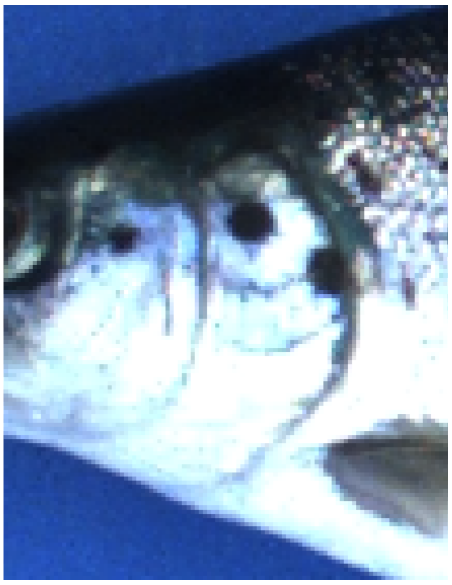


FIGURE 3.26: The gill cover. The image shows how the gills are seen from the side of the fish.

Looking at fig. 3.26, the gill opening is dark, vertical and arc-like. On this particular fish, one of the spots is "connected" with the gill. This often happens, and should be accounted for in the implementation of finding the gills. The dark area of the gill ending, is usually narrow, but sometimes a fish will have a gill opening that is much wider, or almost invisible from the side. The algorithm then has the following features of an object to work with:

1. Is darker then surrounding region
2. Vertical-like shape, compared to fish orientation
3. Narrow, but may vary somewhat it width.

Two different methods were developed for detecting the gill opening. These are from here referred to as Global Gill Detection, and Local Gill Detection. Global Gill Detection tries to detect the gill opening while considering the whole fish, while Local Gill Detection only consideres a ROI, estimated by the nose point and the estimated length of the fish.

Local Gill Detection had the best results, but both methods are presented as they have somewhat different approaches.

3.6.1 Method 1: Global Gill Detection

1. Input image: See fig. 3.27

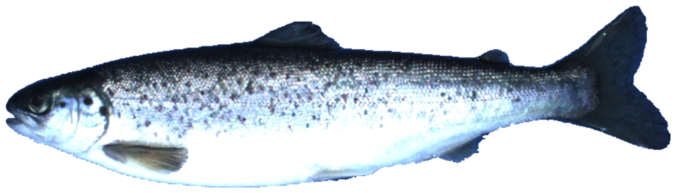


FIGURE 3.27

2. Convert to grayscale. See fig. 3.28

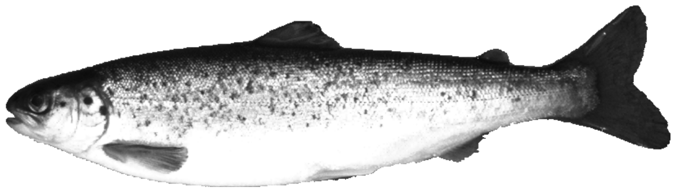


FIGURE 3.28

3. Apply median blur with kernel size 7. See fig. 3.29

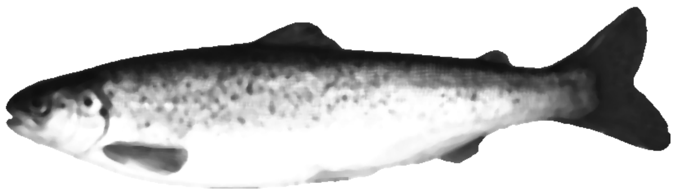


FIGURE 3.29

4. Apply sobel edge detector, considering only vertical edges. See fig. 3.30

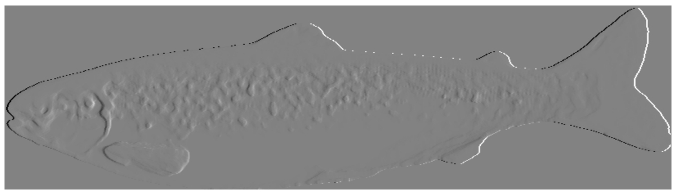


FIGURE 3.30: A sobel edge detector has been run over the image.

5. Remove extreme values (55 and 200)*.Thesearep robablytheoutlineofthefish.Seefig.*??*Createtwo*

6. Label each "object" in the binary images.

7. Keep only the top 10 largest objects in each image. Fig. 3.34

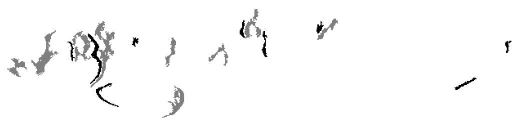


FIGURE 3.34: The 10 largest objects in both images combined into one image.

8. Compare every "dark" edge region, with every "light" edge region. The dark edge should be located to the left of the light edge and the light edge should be not too far away from the dark edge. Also the centroids of the two objects should not be too far apart vertically. Applying these conditions, should leave fewer gill candidates. Shown in fig. 3.35

FIGURE 3.35: The remaining gill candidates.

9. Choose the object that is tallest, and define this as the gill opening. The result is illustrated by fig. 3.36 and 3.42

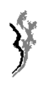


FIGURE 3.36: The remaining candidate for the gill.

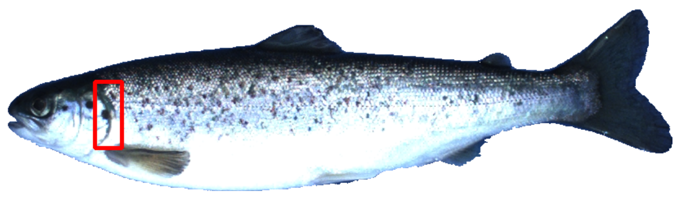


FIGURE 3.37: The location of the gill on the salmon, found with the Global Gill Detection method,

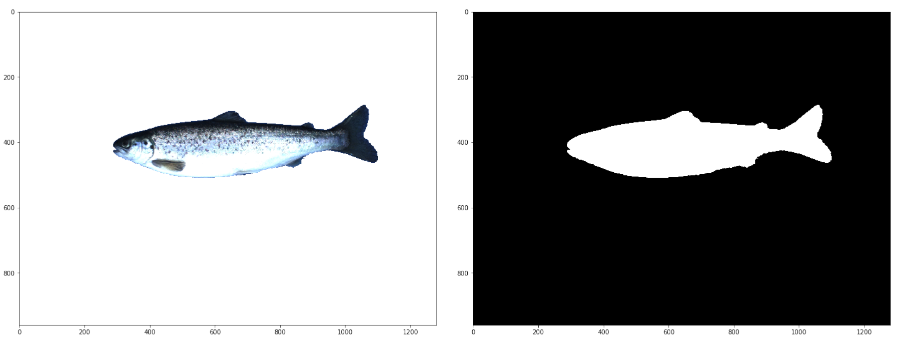
Results and discussion - Global Gill Detection

Off all images, there was a 73.5% hit rate on the gill opening. The failures usually occurs due to issues like vertical spot patterns or marks on the fish, or if the fish's orientation is too skewed. Sometimes, the algorithm detected the eye of the fish instead. Ofen the gill is also nearly invisible. As the results were considered as promising, but not good enough, no further evaluation of the method were done. For future work, it should be considered to allow the method some form for placement guiding.

3.6.2 Method: Local Gill Detection

The code for this method is found in appendix D.4 Code: Get Length and Find Gill Edge on page 158.

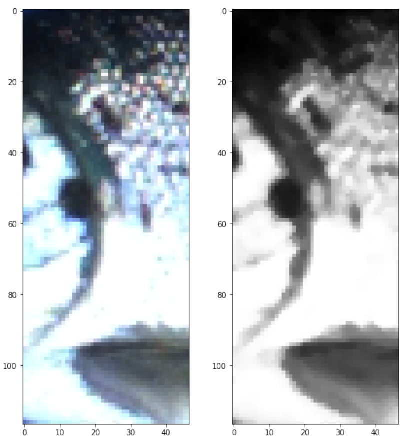
1. Input Data: point of nose and the two back fin tips.
2. Input images: Both the extracted fish and the contour of the extracted fish. fig. 3.38.



(a) Input image 1. Extracted fish. (b) Input image 2. Contour of extracted fish.

FIGURE 3.38

3. Estimate the length of the fish using the euclidean distance from the nose point to the point between top and bottom fin tips.
4. Estimate the ROI by the following steps(Fig. 3.39 (a) shows the process):
   (a) Of the 100% length of the fish, examine the region between 13%-19% of the length from the front. This area has proven to include the gills in all images. Crop out the ROI from this area.
   (b) Convert to grayscale, and apply a median blur with kernel size 3.



(a) The estimated region of interest, concerning the gill opening.
(b) (a), converted to grayscale, and applied a median blur.

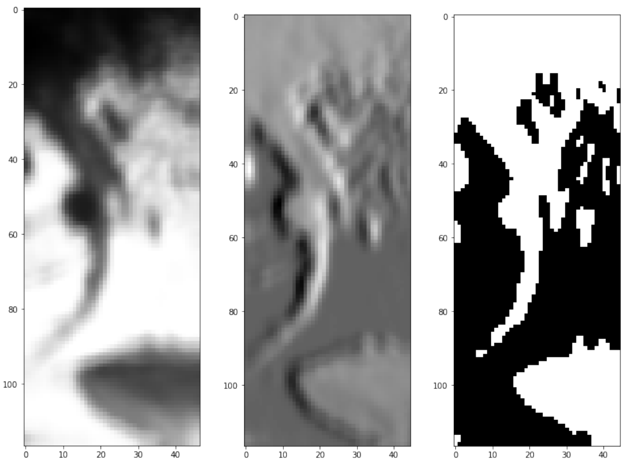
FIGURE 3.39

5. Create a binarized image, representing horizontal edges of the fish, by the following steps: Fig. 3.40

   (a) Apply a vertical blur. Here a 5x3 mask is used, with values: [1, 2, 1] in every row, except in the middle row, which is [0, 0, 0]. This is done in order to accentuate vertical lines. Fig. 3.40 (a)

   (b) Apply a horizontal edge enhancing. Kernel: [1, 0, -1]]. Fig. 3.40 (b)

   (c) Create a threshold for "light" edges. Here, the threshold value was chosen as the median multiplied by 1.03. This was to obtain only the light edges, which are expected to be on the right side of the gill opening.



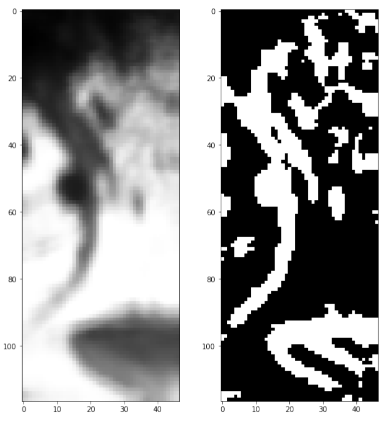
(a) Vertical blur (b) horizontal edge enhancing (c) Binarization

FIGURE 3.40

6. Create another binarized image by adaptive thresholding. Fig. 3.41

   (a) Input the vertical blurred image.

   (b) Apply the openCV function "ADAPTIVE THRESH GAUSSIAN C" to the blurred image, creating an adaptive threshold of the blurred image[22]. Here using neighbourhood size 15x15, and C=4 (See 2.2.1 Adaptive Gaussian Thresholding).



(a) (b)

Figure 3.41

7. Use the two binarized images to find what is most likely the gill opening: Fig. 3.42

   (a) Blend the two images together

   (b) Keep only the brightest pixels

   (c) Label the different objects in the image, and choose the tallest one to be the gill opening.

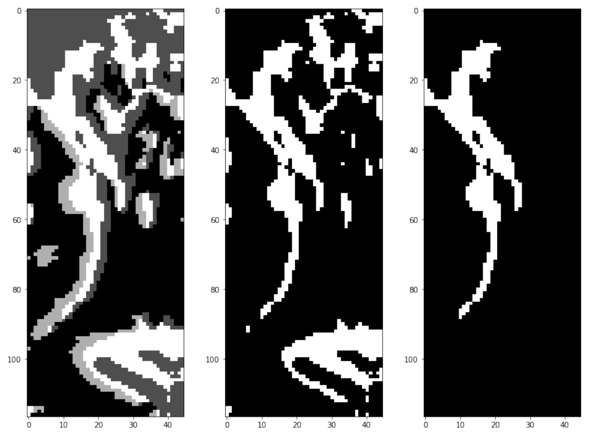
   (d) Illustrated result

8. Find the rightmost pixel that is part of the gill opening. This point will later be used to define the head region of the fish.

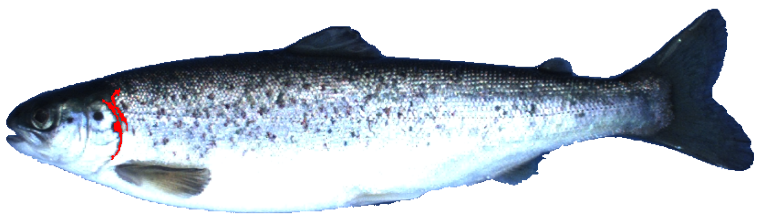
9. Output data: column distance from nose to ending of gill opening.

   - If the nose is at $[x1, y1]$ and gill ending is at $[x2, y2]$, the column distance will simply be $\|x2 - x1\|$, not considering the y-values.

The results of this implementation is presented and discussed in 4.2 Local Gill Detection Results, starting on page 78.



(a) (b) (c)



(d)

FIGURE 3.42

3.7 Detecting Gill Spots

Assuming that the spots on the gill cover of Atlantic Salmon are local regions that are darker than their surroundings, they can be defined as blobs. They should then be considered as important candidates for robust interest points when matching different Salmon (See 2.4 Detection of spot-like structures). The spots are also a favourable choice, considering long term reconition[14].

Two methods were applied and experimented with. The methods will from here on be reffered to as:

1. Simple Blob Detection (Existing method)
2. Dark Point Detection (Custom made for this project)

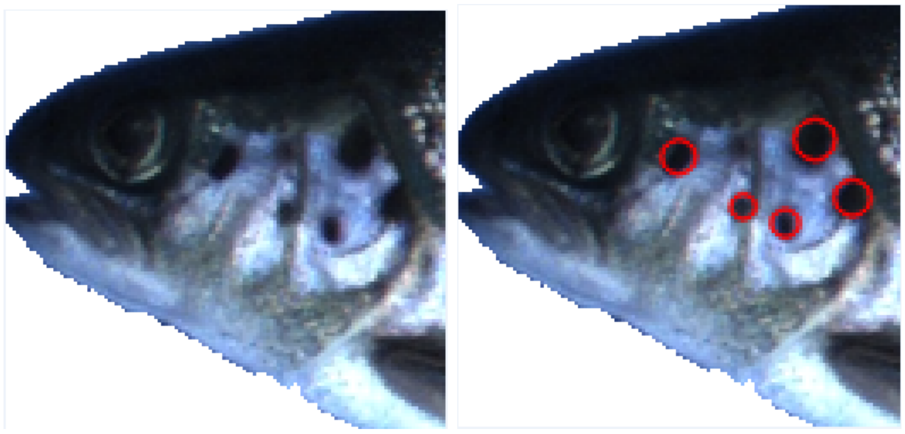
3.7.1 Simple Blob Detection Method

The Simple Blob Detection method is the method "SimpleBlobDetector", in the OpenCV library (`https://docs.opencv.org/`). It has an approach similar to the use of Grey-level trees (See 2.4.1 Grey-Level blobs and Grey-level trees, page 28). The details of the algorithm is exclusive, but detailed experiments are presented in 4.3.1 Spot detection by Simple Blob Detection on data-set 1 to further understand its behaviour.

The method uses the nose point and gill-cover ending from methods Nose and Tail Tips detection and Method: Local Gill Detection to extract the input ROI (See fig. 3.43 (a)).for the methods presented here.

The code for this implementation is found in appendix D.5 Code: Simple Blob Detector Implementation on page D.5. The method works as follows:

1. A blob detector is created with the following parameters:
    - minimum Threshold
    - maximum Threshold
    - Circularity
    - Convexity
    - Inertia Ratio
    - minimum Area
        - This parameter also scales the max area size, which is exclusive to the user. Reducing minimum area size could therefore in some cases result in fewer spot detections.
2. The image is binarized multiple times at increasing threshold values. This results in several images with different binary objects. The incremeent value is fixed and defined inside the function. Therefore it is not possible to adjust.
3. Between the several binary images, nearby blobs are grouped together by their center coordinates, to create the output blobs.
4. Finally, shape features such as area, perimeter and radius, are used to filter out unwanted blob shapes. This is decided by the input parameters. The result is seen in fig. 3.43 (b).



(a) Input image: ROI around the head. Automatically extracted by the nose and gill cover opening.

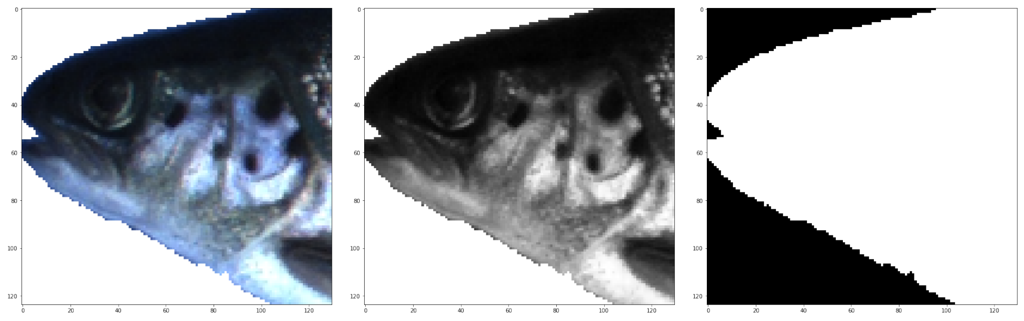
(b) Result of blob detection.

FIGURE 3.43

3.7.2 Dark Spot Detection method

The idea of the Dark Point Detector is similar to the Laplacian of Gaussian blob detector (See chapter 2.4.1 LoG, page 27), but it does not use the Laplacian to find the local extrema. Instead it uses a neighborhood approach. Also, it is combined with some thresholding, so that only dark spots in relatively light areas are detected. Hence, why it is called Dark Spot Detector. The code for this method is found in appendix **??** **??** on page **??**. The method functions as follows:

1. Import input images and convert to grayscale as shown in fig. 3.44.



(a) Input ROI: The head of the fish.Locating the gills

(b) (a) converted to grayscale

(c) The contour of the head of the fish

FIGURE 3.44

2. Define parameter values. The parameters are:

   - Kernel size for gaussian blur: k
   - Sigma for gaussian blur: σ

- Scaling factor for max allowed value of spot center candidate: p

3. Calculate median value of the gray image, which will later be used for thresholding.

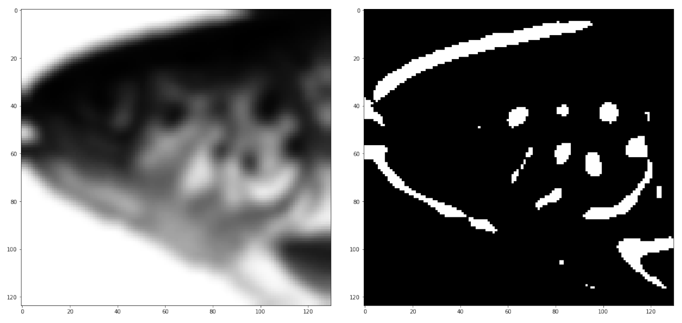
4. To find the possible spot areas, start by creating a binarized image by steps illustrated by fig. 3.45:

   (a) Apply a gaussian blur with kernel size k and σ to the gray image.

   (b) Apply adaptive thresholding. function : cv2.adaptiveThreshold. (See 2.2.1 Adaptive Gaussian Thresholding)

      i. Data-set 1: A 21x21 Neighborhood is considered, and C=14.

      ii. Data-set 2: A 81x81 Neighborhood is considered, and C = 31.



(a) Strong blur. Here, $k = 9$, and $\sigma = 3$. (b) (a) is binarized by adaptive thresholding

FIGURE 3.45

5. To narrow down the possible spot areas, another binary image is created by normal thresholding, as shown in fig. 3.46:
   (a) Apply a weak gaussian blur with kernel size 3 and sigma 3 to the gray image.
   (b) Apply normal/global thresholding. The threshold is set by: threshold = median $* q$.

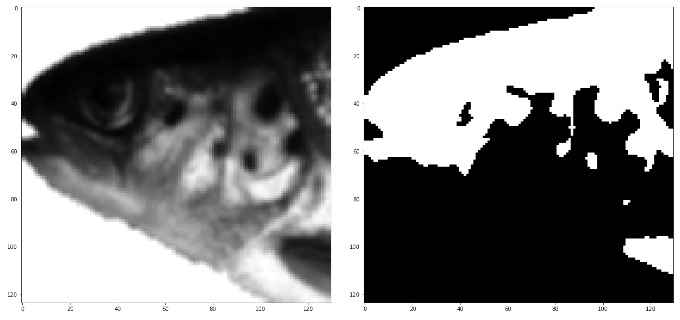

(a) Weak blur by kernel size and σ equal to 3 (b) (a) is binarized by normal/-global threshold. In this example, $q = 1.00$

FIGURE 3.46

6. Find the possible spot areas, as illustrated in fig. 3.47, by steps:
   (a) Blending the binarized images from fig. 3.45 and 3.46.
   (b) Keep only the brightest pixels as the potential spot areas.

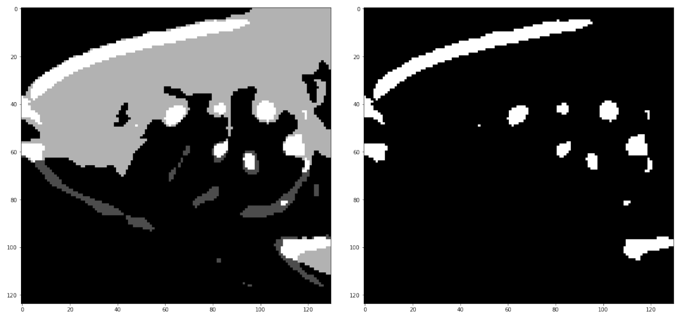

(a) Blended image (b) Only the brightest pixels are kept, as possible spot areas

FIGURE 3.47

7. Remove the contour edges of the head from possible spot areas. This process is explained by points a-d, shown in fig. 3.48:
   (a) Invert the contour image
   (b) Apply adaptive thresholding. function : cv2.adaptiveThreshold. 31x31 Neighborhood is considered, and C = 5, for both data-set 1 and 2.(See 2.2.1 Adaptive Gaussian Thresholding)

(c) Blend with fig. 3.47 (b).

(d) Keep only the brightest pixels.

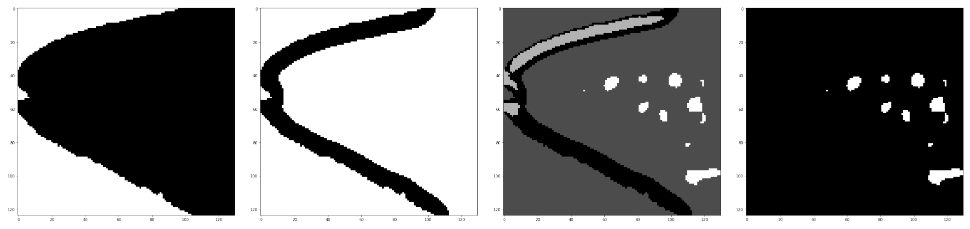

(a) Inverted contour image (b) (a) is applied an adaptive threshold (c) (b) is blended with fig. 3.47 (b) (d) Only the brightest pixels are kept

FIGURE 3.48

8. Find all local minimas in the first blurred image. This is done by iterating through every pixel in the image, and finding every pixel that has lower or equal value than all of the pixels in it's 3x3 neighbourhood. The maximum value of these pixels are described as media $* p$. See fig. 3.49.

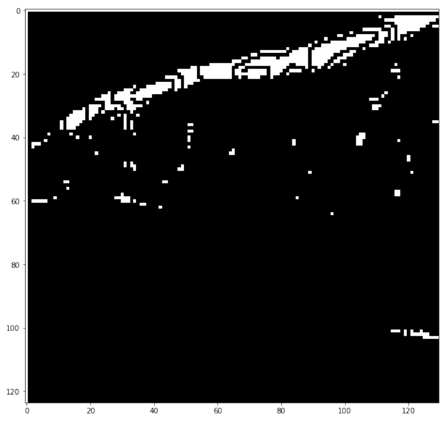


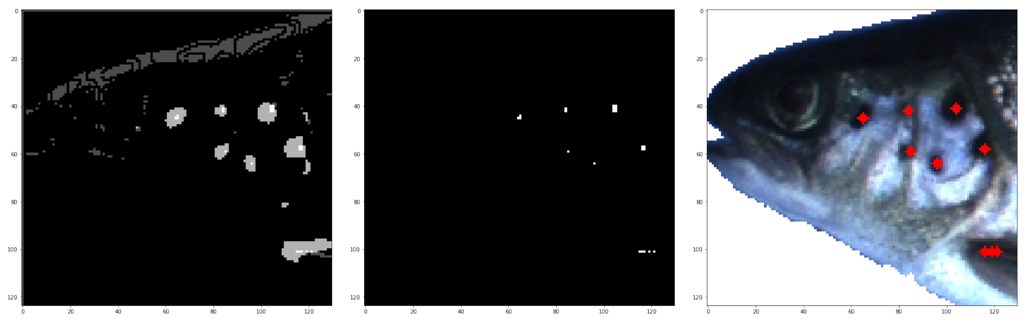
FIGURE 3.49: The pixels in the image that are the local minima in their 3x3 neighborhood. These are possible spot centers. In this example, p = 1.20

9. By combining the spot area image and the local minima images, the spot centers are picked, as shown in fig. 3.50:

   (a) Blend image fig. 3.48 (d) and 3.49.

   (b) Keep only the brightest pixels. The center of each distinct object here, is defined as a spot center. To prevent the eye from being detected, only areas in the image past column number 50 could qualify as spot centers.

   (c) Illustrated result. In this example, all the spots were somewhat detected, but excess spots were detected on the pectoral fin.



(a) Fig. 3.48 (d) and 3.49 are blended.

(b) Only the brightest values are kept

(c) Illustrated result. In this image, the method has detected all spots, though there are some false detection on the pectoral fin.

FIGURE 3.50

Experiments for this method is conducted in 4.3.2 Spot detection by Dark Point Detection - Experiments and Results, starting on page 94. The results are discussed afterwards.

3.8 Individual Recognition

To recognize individual salmon could be achieved with machine learning algorithms, but to do so requires data. The data-set used only has three images of each salmon and is scarce from a machine learning standpoint. As mentioned in Individual Recognition, page 63, the amount of images used to recognize objects are often in the thousands. 1.1.3 Facial Recognition, page 6 mentioned how a structural based method to separate individuals might work better, especially with the lack of enough data. Having three images of each specimen is alike the situation to both [10] and [11], who all used structural based methods.

A method like this doesn't necessarily recognize an individual salmon, but rather discriminate against all salmon who are not similar enough to a set of values. Is it possible to discriminate against all salmon except the correct one? To do so, a feature vector was made for each individual fish. The more similar two feature vectors are, the more likely they are the same salmon. The feature vectors contain 19 features, and the corresponding filename, making it 20 columns long. The entries in a feature vector is shown in table 3.2.

Filename	The filename is used to evaluate correct or incorrect classification
nosetoPec	The distance between the nose and the attachment point of the pectoral fin
Length	The length from the nose to the point between the two tail tips
Head length	The length from the nose slightly behind the gills
pectoTail	Distance from end of pectoral fin to the point between the two tail tips
nose spot(i)	Distance from nose to a spot on the head
angle(i)	The angle corresponding to nose spot(i)
totSpots	The total number of spots on a fish

TABLE 3.2: All entries in the feature vector and a brief explanation of each. For clarification, the head of the fish is from the nose to the end of the gills. The maximum number of spots possible were 8, hence $i = [1 - 8]$.

The spots on the fish were manually extracted, to lower the probability that feature vectors of the same salmon had more of less spots. The spots were also classified as distinct or complex(which is of no importance for recognition; a spot is a spot), with the maximum number of distinct spots equal to three, while the maximum number of complex spots were five. Hence, eight potential spots altogether.

Fig. 3.51 and fig 3.52 shows four distribution plots of the length of the fish, the head-length and the distances from the nose to pectoral fin attachment and from the end of the pectoral fin to the tail.

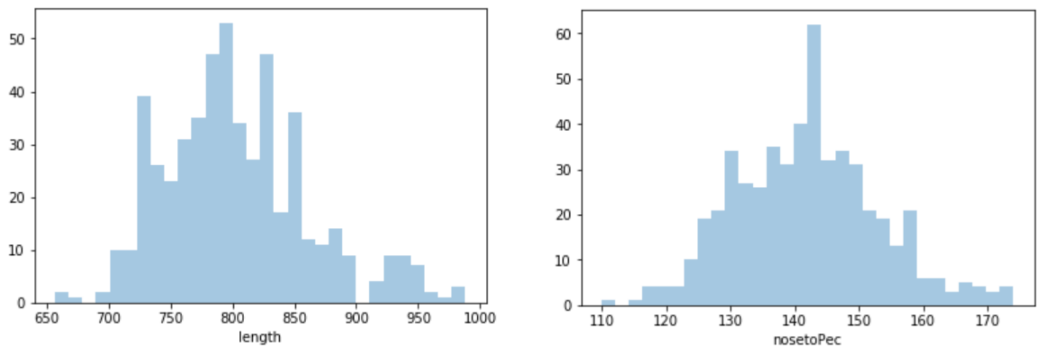


FIGURE 3.51: This is the distribution of lengths of the fish and the distance from the nose to the pectoral fin in the MOWI data-set

Unlike the angle, these features are distance-based. The angle for each spot is calculated with respect to the fish's orientation. A salmon can move from image to image, which would render the angle features useless. Fig. 3.53 illustrates how this is calculated together with equations 3.1

To get the angle between two points, eq. 3.1 was used.

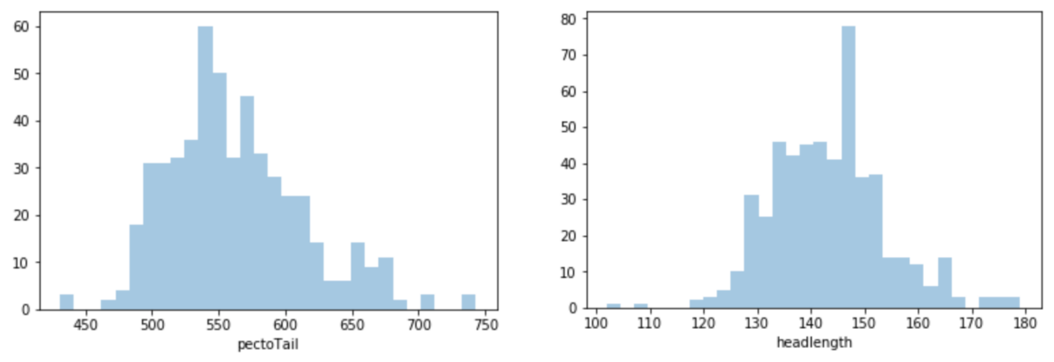


FIGURE 3.52: The distribution plots for the distance from the end of the pectoral fin to the middle between the tail tips and the length of the head of the fish.

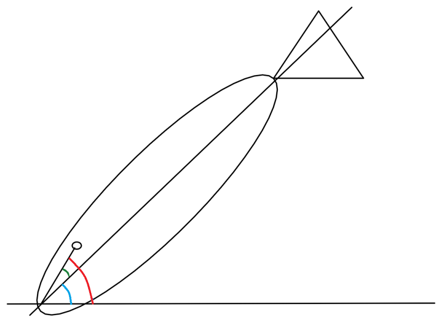


FIGURE 3.53: The angle to a spot, θ_{spot}(marked with green line), is calculated by subtracting θ_{fish}(blue line) from $\theta_{baseline}$(red line).

$$\theta = \arctan(\frac{y_2 - x_2}{y_1 - x_1}) \tag{3.1}$$

Where one point is described by (x_1, y_1) and the other by $(x_2, y_2$. Once the angles are found, the spot's angle is calculated by $\theta_{fish} = \theta_{baseline} - \theta_{fish}$.

Once all features vectors are computed for each specimen, it is put through the algorithm presented in fig. 3.54, fig. 3.55, fig 3.56 and fig. 3.57. The code for how the feature vectors were created can be found in D.9 Code: create Feature vectors page 177. The code corresponding to the flowcharts in this section can be found in D.10 Code: Individual Recognition page 186.

The algorithm goes through two major stages, the first shown in fig. 3.54 and fig. 3.55. Stage 1 creates a text file containing all so-far-accepted matches for all salmons with corresponding score. The format of the file is first the label fish's filename, the matched test fish's filename and the corresponding score those two fish got. The test fish is the fish tested against a label fish to see if it is a possible match.

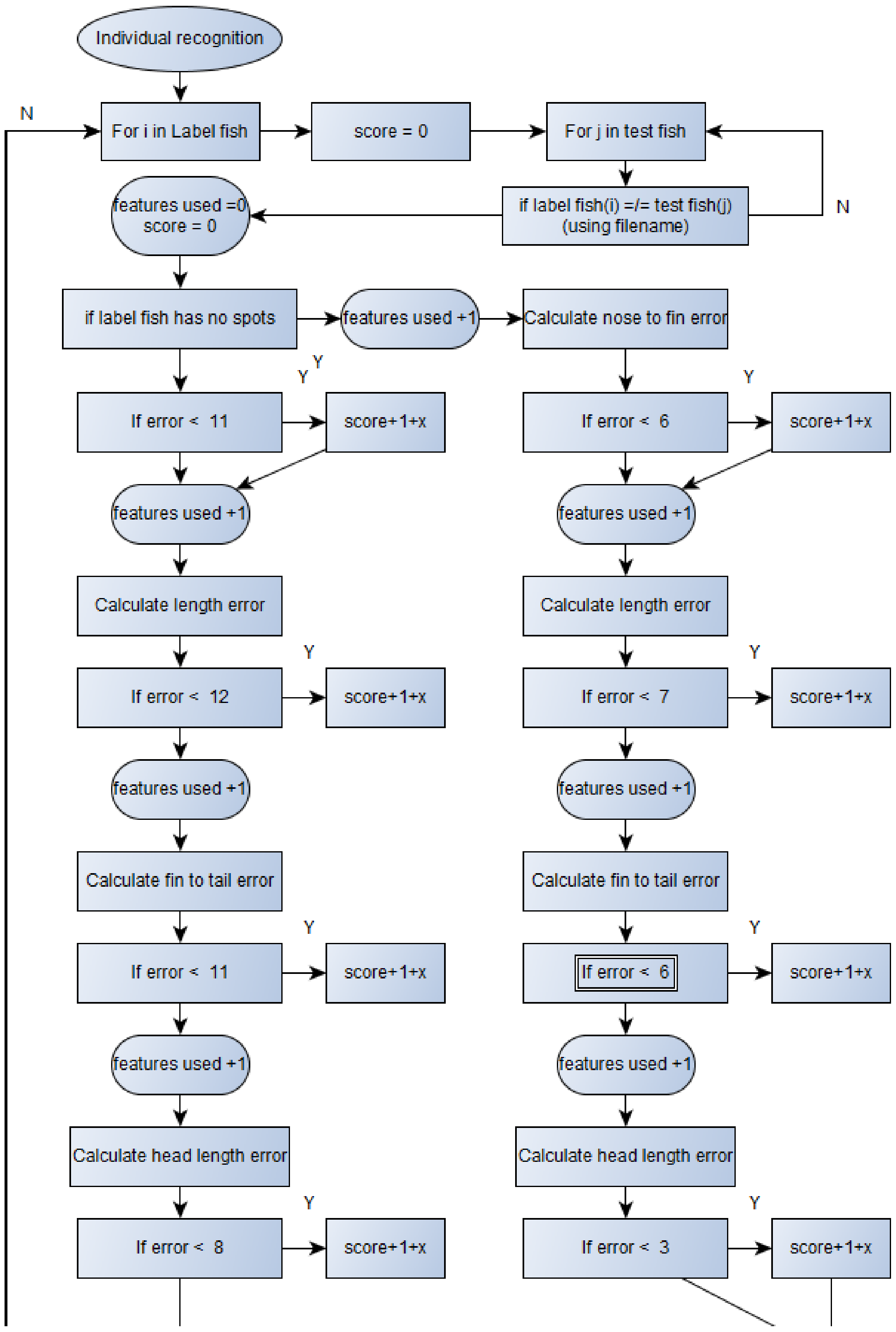


FIGURE 3.54: Stage 1, step 1 of the algorithm goes through every label fish and tests it against every test fish to see if the score gets high enough for a potential match. Using the feature vectors previously created, it find the errors between the label and test fish's features. The score scales with the amount of features used. The x in the score rectangles is a varying parameter that is higher the closer the features of the two fish's are to each other.

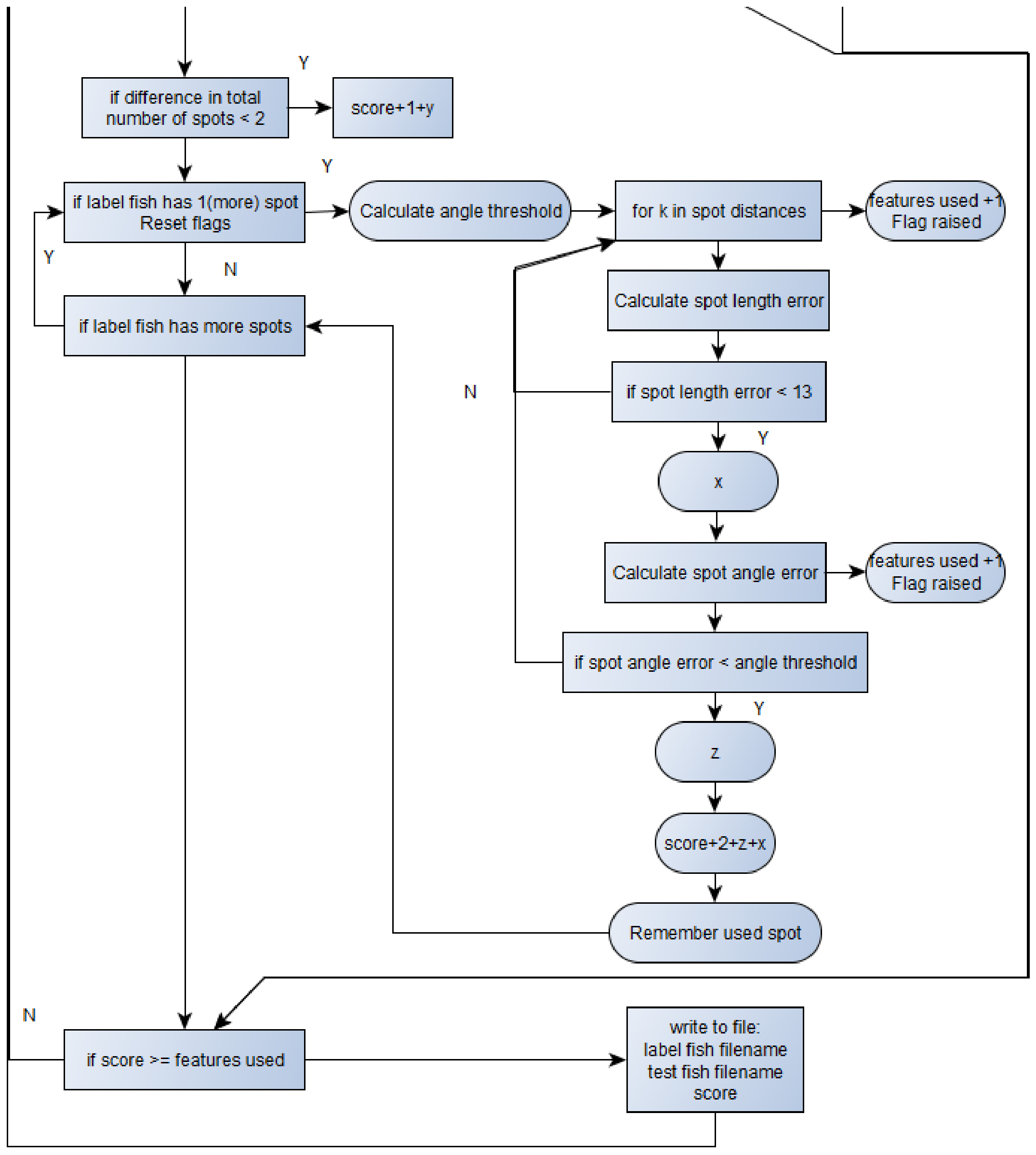

Y
if difference in total number of spots < 2
score+1+y
Y
if label fish has 1(more) spot Reset flags
Calculate angle threshold
for k in spot distances
features used +1 Flag raised
Y
N
if label fish has more spots
Calculate spot length error
N
if spot length error < 13
Y
x
Calculate spot angle error
features used +1 Flag raised
if spot angle error < angle threshold
Y
z
score+2+z+x
Remember used spot
N
if score >= features used
write to file: label fish filename test fish filename score

FIGURE 3.55: Stage 1, step 2 of the algorithm shows almost exclusively the testing of the spots on a fish. If a spot on the test fish is accepted as the same spot on the label fish, it cannot be chosen again. The more similar the spots, the higher the score(shown here as x and z variables. If the final score is the equal to or more than the used features, the filenames of both fish are written to a text file together with their corresponding score.

The algorithm subtracts a fish's feature vector from another fish's feature vector. The lower the result, the more likely they images of the same fish. Since some salmons have more spots than others, the amount of available features for each salmon increases its potential to be correctly classified. Fig. 3.54 shows that if the label fish contains no spots, there are only four features available features to attempt to correctly classify the fish. Because of this, the threshold are about 3-4 times as strict. This is to minimize the amount of misclassifications if there only are four features to use. If the label fish has a spot or more, the threshold for the four first parameters are slacker.

The angle threshold is calculated for each label fish for each spot. It is calculated as shown in eq. 3.2.

$$\theta_{thresh} = \arctan \frac{S_{thresh}}{D_{np}} \tag{3.2}$$

where $\theta_[threh$ is the threshold and angle has to be below, S_{thresh} is the distance threshold for the spot and D_{np} is the distance from the nose to the spot. This means that the further away a spot is from the nose, the lower(stricter) the threshold will be. With a $S_{thresh} = 12$, the spots placement can more around 12 pixels and still be classified as a potential spot. A spot that is closer to the nose will yield a much greater change of angle than spots further away moving the same distance. This ensures that the threshold scales to the distance to the spot, allowing spots closer to the nose to have a larger Angel-Error, but still make it through.

3.8.1 Feature and Score System

If all feature vectors had the same amount of features, then if the score of two fish got to, or over, a set amount for all fish, it could be classified as a potential match. But since the amount of available features vary depending on the label fish, the score system has to be scaled on how many features were used.

For each feature used, the *feature_used*-variable is incremented by 1. The *feature_used*-variable is always incremented to at least 4 for each individual, since all fish have the four first features(nose to fin-, length-, fin to tail- and head length-distance), but not all have spots. If the test fish successfully get through the thresholds, the *score*-variable is incremented by $1 + x$. The x-variable is higher if two specimens are more similar. How this is done is shown the **example below**:

Say the label fish has one spot. The algorithm gathers all spots on whichever test fish it is currently testing. If the distance from the nose to the spot of the label fish is 80, then all distances from the nose to spots on the test fish are subtracted, one by one, from that distance. For simplicity, let's say two spots on the test fish came through the error-threshold with the distances 79 and 68, respectively. The error for each spot is: Spot 1 error $= 80 - 79 = 1$ and: Spot 2 error $= 80 - 68 = 12$. Spot 1 on the test fish is closer to the same placement of the spot on the label fish than spot 2.

Since there was a spot to test against, the *feature_used*-variable is incremented by 1, but no score is added to the *score*-variable yet. This is to ensure that the fish is discriminated against should the angle difference for the spots be too great.

Both spot errors are divided by 100, reducing them to: Spot 1 error = 0.01 and Spot 1 error = 0.12. If the difference in angles of both spots are below the angle-threshold, the value added to the *score*-variable is shown in eq. 3.3.

$$\text{score} = \text{score} + 2 + (0.12 - \text{spot}x\text{error}) \tag{3.3}$$

where x is 1 and 2(for the two spots). If the existing score was 4, the new score for both spots would be: score1 $= 4+2+(0.12-0.01) = 6.11$ and score2 $= 4+2+(0.12-0.12) = 6$. The score for the closest spot will have a higher score than the one further away, but both are eligible as correct matches. The same method is used to yield a higher score to angles that are closer too, as well as all distance based features.

If a spot is accepted, the score is always incremented by at least 2. This is because two features were used: the distance and the angle. Two features are still used if the distance-error is lower than the threshold and the algorithm has to test the angles. Should no angle suffice, a feature was still used and the *feature_used*-variable is incremented by 1, but no score is awarded, making sure that specimen is out of the question. The same spot cannot be used again if the label fish had another spot that also was close to the already used spot on the test fish.

If a test fish has many spots, but the label fish only has one, there is a chance that at least one of the many spots will be around the same position as the one spot on the label fish. To combat this, the first rectangle fig. 3.55 will calculate the difference in spots. If the difference is either 1 or 0, the *score*-variable is incremented by $1 + y$. The difference is divided by 10 and $y = 0.3 - (\frac{\text{Spot Difference}}{10})$. This gives fish with either an equal amount or a one spot difference a significantly higher score, making it less likely that a salmon with many spots gets a lucky hit on a salmon with only one spot. If the label salmon has no spots, it doesn't matter, because then the algorithm skips that whole segment.

Fig. 3.55 has several rectangles where a ”flag” is mentioned. Either they are reset, which means the variable is set to 0, or they are raised; set to 1. This is to make such that if many spots on a test salmon qualifies as a potential spot, the *feature_used*-variable is not incremented too many times, but can only be incremented once for each spot checked on the label fish.

Stage 2 of the algorithm handles the text file created at the bottom of fig. 3.55 and uses the scores to determine which potential so-far-classified specimen is which, and is illustrated in the flowchart in fig. 3.56 and fig. 3.57.

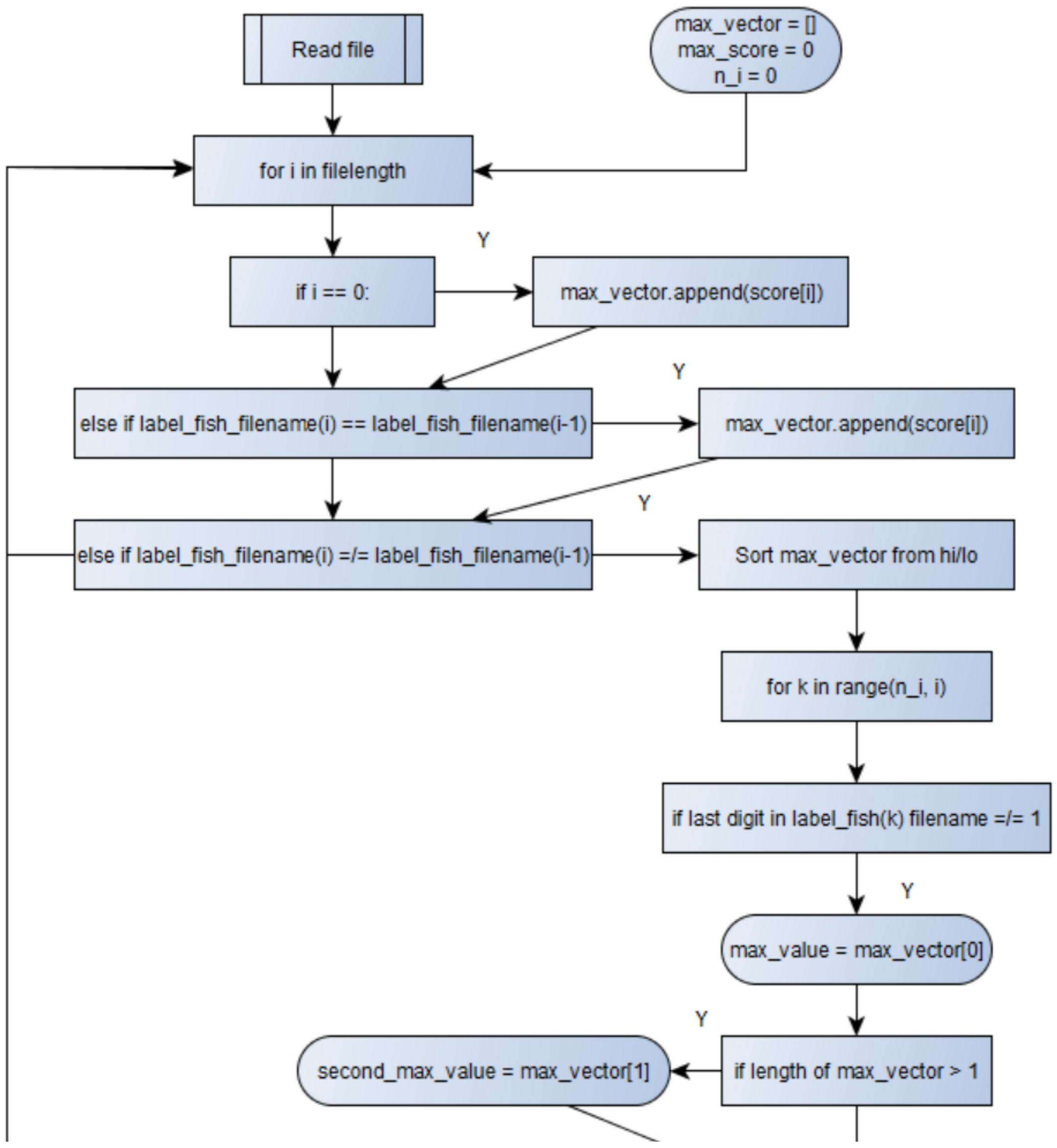

Read file
max_vector = []
max_score = 0
n_i = 0
for i in filelength
if i == 0:
Y
max_vector.append(score[i])
else if label_fish_filename(i) == label_fish_filename(i-1)
Y
max_vector.append(score[i])
else if label_fish_filename(i) =/= label_fish_filename(i-1)
Y
Sort max_vector from hi/lo
for k in range(n_i, i)
if last digit in label_fish(k) filename =/= 1
Y
max_value = max_vector[0]
if length of max_vector > 1
Y
second_max_value = max_vector[1]

FIGURE 3.56: The text file containing filenames for the label and test fish and the corresponding score is read line by line. Once the algorithm reaches the last entry for one label fish, the two largest scores in the $score_vectorareextracted$.

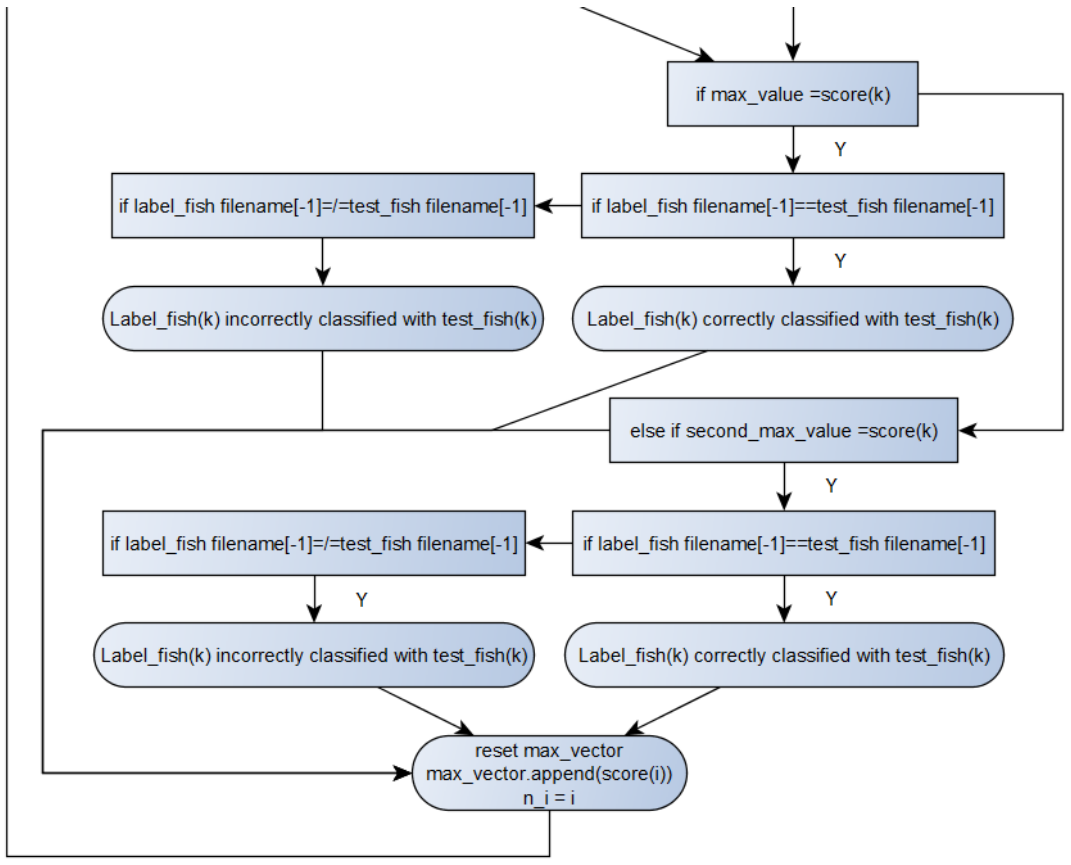

if max_value =score(k)
Y
if label_fish filename[-1]==test_fish filename[-1]
if label_fish filename[-1]=/=test_fish filename[-1]
Y
Label_fish(k) correctly classified with test_fish(k)
Label_fish(k) incorrectly classified with test_fish(k)
else if second_max_value =score(k)
Y
if label_fish filename[-1]==test_fish filename[-1]
if label_fish filename[-1]=/=test_fish filename[-1]
Y
Y
Label_fish(k) correctly classified with test_fish(k)
Label_fish(k) incorrectly classified with test_fish(k)
reset max_vector
max_vector.append(score(i))
n_i = i

FIGURE 3.57: The entries for the current label fish are run through once more to locate which test fish held the highest scores. There are three images of each fish(almost), hence the two largest scores are extracted to find the two images of the same salmon. When the max score is equal to the score between the label fish and a test fish, they are classified as the same specimen. If the three first digits of the filename is the same, it was a correct classification, and incorrect if not. Same procedure for the second largest score. Finally, the $score_vectorisresetandthenextscoreinlineappended.n\_iissettothecurrenti.$

The text file contains all possible matches for a given label fish, each with the corresponding score. An example is: ”0022”,”0023”,”14.23”, where the first number is the label image, the second the test image and lastly; the score. The algorithm will go trough every line in the file and increment i respectively. When the algorithm has gone through all entries for one label fish(for example ”0022”), it needs to stop and extract the two largest scores in the *max_vector*, which is a vector containing all the scores for a particular label fish. This is done by checking if the filename of the label fish at entry i(for example ”0031”) is the same as in entry $i - 1$(”0022”). It it is not, it stops going through the document and instead works with the scores in the *max_vector* so far. The reason the two largest scores are extracted is because there are two correct matches for each label fish, except for 16 of the label fish, which only got one correct match.

The algorithm runs through the scores for all test fish for the particular label fish and locates which one held the same score as either the largest score or second largest score. The test fish belonging to those two scores are set as the match for the label fish. Then the algorithm checks if it made a correct or incorrect match. If the three first digits in the filename are the same, it is the same specimen in both images. This is done in fig 3.57, in the rectangles with the line *if label_fish file[-1]...*, where the last characters in the filename is removed.

After a test fish has been either correctly or incorrectly classified, the *max_vector* is reset. Since i already holds the value for the next label fish, the corresponding score is appended into the vector. The *n_i*-variable is set to i. This is done so that when all test fish's scores for a certain label fish has been added to the *max_vector* and the largest and second largest scores have bee found, the algorithm will only search through the scores of the test images for *that* label fish, in case another test fish has the same score anywhere else in the file.

Chapter 4

Experiments and Results

This chapter presents experiments and results of Fin Detection, Local Gill Detection, Simple Blob Detection, Dark Spot Detection and Individual Recognition. It should be noted that the experiments on the Simple Blob Detector and Dark Spot Detector are comprehensive and extensive compared to the other experiments and results.

The results of the Fish extraction method and Global Gill Detection were only evaluated subjectively, and is not presented here.

4.1 Fin Detection Results

The fin detection algorithm was tested on the two data-sets and the results are presented in table 4.1.

Data-set	1(MOWI)	2(IMR)
Images in data-set	537	246
Success rate	99.2%	93.5%
Classified with threshold: Otsu	519	232
Threshold: 100	11	8
Threshold: 170	4	1
No pectoral fin detected	2	5
Misclassifications	1	8
Sum of faults	3	13

TABLE 4.1: The results for testing the fin detection algorithm on both data-sets. The algorithm performs better on dataset 1, but this was the set the algorithm was developed on. It performs well on the IMR data-set as well.

It is clear for table 4.1 that segmenting the images with thresholding using otsu's method was a good choice. It distinguished the pectoral fin in over 90%(close to 95%) of the cases. The hard thresholding of 100 and 170 picked up some of the few images otsu's method couldn't solve and bettered the success rate in both instances.

The method was developed to detect the pectoral fin on data-set 1, but was tested on data-set 2 after finalization. Data-set 2 has a more crude set of images, where the background fluctuates with patterns and other aforementioned disadvantages in table 3.1.

It should be noted that table. 4.1 is not as truthful as it may appear. The results in the "Threshold"-rows are not completely accurate. The misclassification row are images where the algorithm thought it found the pectoral fin, but other parts of the fish were actually detected. Whether the algorithm classified them wrongly with the otsu-threshold or any of the hard ones, is unknown, but since the otsu-threshold classified many times more images than the hard thresholds, it can be assumed that the misclassifications belong there. Truer values may therefore be: 518 and 224 in row 4.

A detailed explanation of the results is found below for both data-sets.

4.1.1 Results on Data-set 1, MOWI

This algorithm achieved a success rate of 99.2% on 537 images, either misclassifying or not detecting a pectoral fin on three images. In two images, no pectoral fin could be located and one was wrongly classified. The three images are presented in fig. 4.1.

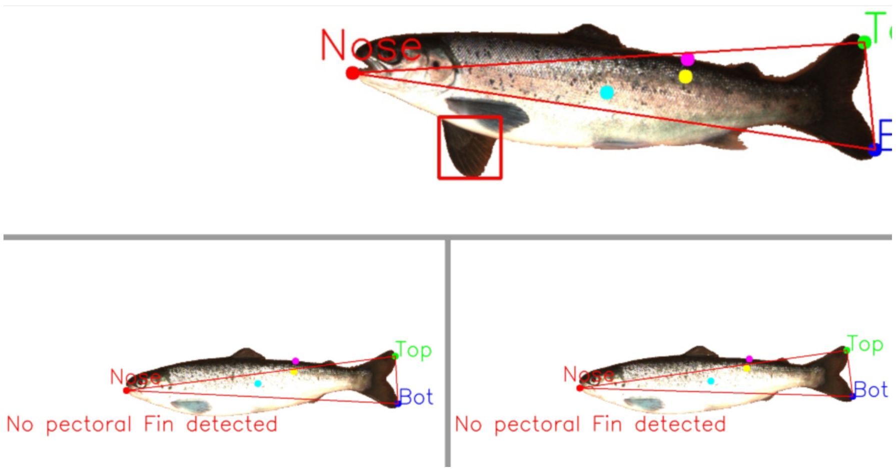


FIGURE 4.1: The three images who were either misclassified or no pectoral fin could be detected.

Fig. 4.2 shows nine images who were correctly classified with both their nose, tail tips and pectoral fin.

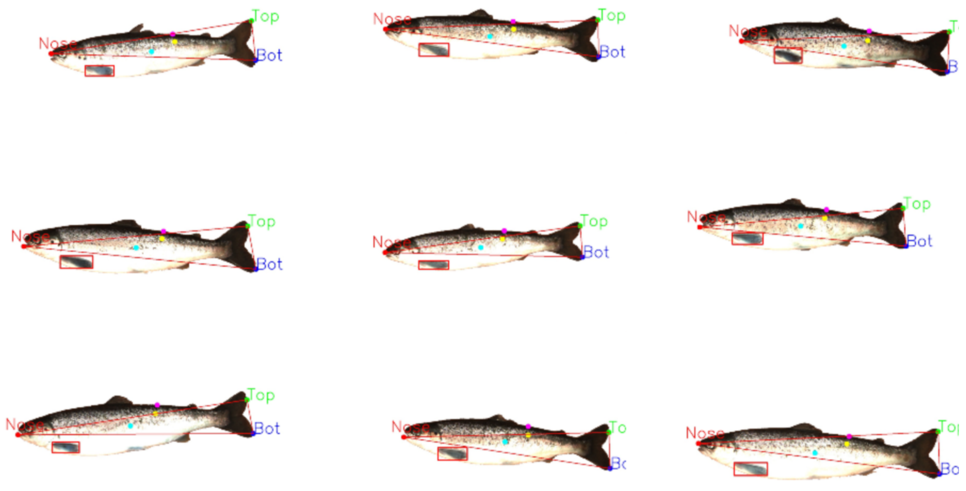


FIGURE 4.2: Nine random correctly classified images of 537

4.1.2 Results on Data-set 2, IMR

It should be experimented how the developed method worked on a data-set it wasn't specifically built for. Since it is tailored to data-set 1 from MOWI, its robustness should be tested on images with poorer quality for this project, as mentioned in table. 3.1 with data-set 2 from IMR.

The method was modified slightly to fit these images. These images were larger than the data-set 1 images, and thresholds had to be adjusted. In data-set 1 the size-threshold for the pectoral fin was 800, but in data-set 2 most pectoral fins occupied over 20 000 pixels. It led to areas much smaller than the fins being detected, such as the iris of the eye and random areas closer than the fin to the nose.

The method also works together with the previous algorithm which locates the nose and tail tips. For this data-set however, the tail was not in the image. In most cases, it was simply left out and in a few images a human hand held the salmon still. Instead of modifying the method, the nose point was located manually on each image and fed into the fin detection method. Since locating the nose and tail tips wasn't the objective, this was a good substitute.

Even on cruder images, the method performed with a 93.5% success rate. There was a total sum of 13 faults, either misclassifications or no fin was detected, in the data-set.

Fig. 4.3 shows four instances where the algorithm either misclassified images or detected no pectoral fin at all.

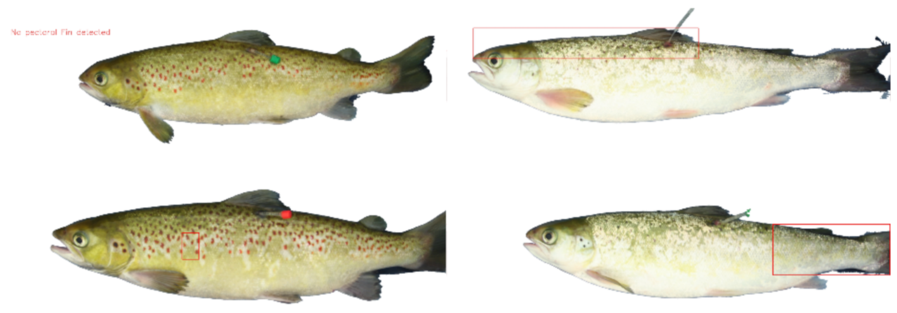


FIGURE 4.3: Four images out of thirteen faulty images. The other wrongly classified images were similar instances like these

The images in fig. 4.3 often have clear possible reasons for why they were misclassified. The fish in the leftmost images are greener compared to most other salmons. The pectoral fin is a darker green, but still green. Compared to the fish in data-set 1, where the pectoral fin is distinct(dark to light skin), the fin is harder to separate when it has a green hue similar to its skin. The rightmost images' fish has either a small pectoral fin or one of normal size, but they are both very light in intensity, much like the skin of the rest of the fish. This makes it harder to separate it from the body.

4.1.2.1 Potential Improvements

To make the method more efficient on images it wasn't constructed to work on, instead of only having a lower threshold for the fin size, and upper could be added. Then large areas like the rightmost images in fig. 4.3 would be eliminated. With an accurate bracket for the pectoral fin, an adaptive threshold method could be added. The method could try out different thresholds(with increments of, for example, 10) until a suitable area is found close enough to the nose.

4.2 Local Gill Detection Results

The results are evaluated by the criteria that it has somewhat managed to localize the gill opening of the fish. In 522/534 images (97.8%), the gill opening was roughly detected, meaning significant parts were included in the result, but often not a 100%. In many cases, other dark patterns close to the gill opening was included in the result. Hence, the result of 97.8% is discretionary, and this evaluation is based on whether it is assumed that the result is favourable or not.

To examine the "successful" results further, a custom program was made. The user is asked to manually detect the gill cover ending by a mouse click. This manually collected data was then used to evaluate the algorithm for finding the gill cover ending. The python code for this program is found in Appendix D.7 Code: Manual Spot Detection, Manual Gill Detection.

To get an understanding of how good or bad these results are, they are compared against a statistical guess of the placement of the gill opening. By the manual detection, it was found that the gill cover ends at approximately 17,26% of the full length of the fish. This was used to calculate the estimated gill cover end of every fish. The Local Gill Detection method could then be compared to a statistical guess. This comparing is shown in table 4.2.

Local Gill Detection VS Statistical guess		
Type	Local Gill Detection	Statistical guess
Absolute avg. distance	5.32/1.27mm	5.50p/1.32mm
Std. variance	5.34p/1.28mm	6.86p/1.64mm

TABLE 4.2: This table shows the comparison of Local Gill Detection VS a statistical guess. The Local Gill Detector seems to perform better than a statistical guess.

What is not concerned in the table, however, is that a statistical guess would not take into account the need to be conservative in terms of rather placing the gill opening further away from the nose, than closer. This is desired because when creating a ROI to detect the spots on the gill cover, as a too large ROI is much preferred over a too small ROI. Also, the variance is smaller using the algorithm, meaning more precise. This is important if the feature would be used for individual recognition. It would not matter that the detection was a bit too much to the right, as this would be the case for most images.

The results of the automatically detected gill cover end, are shown in the histogram of fig. 4.4.

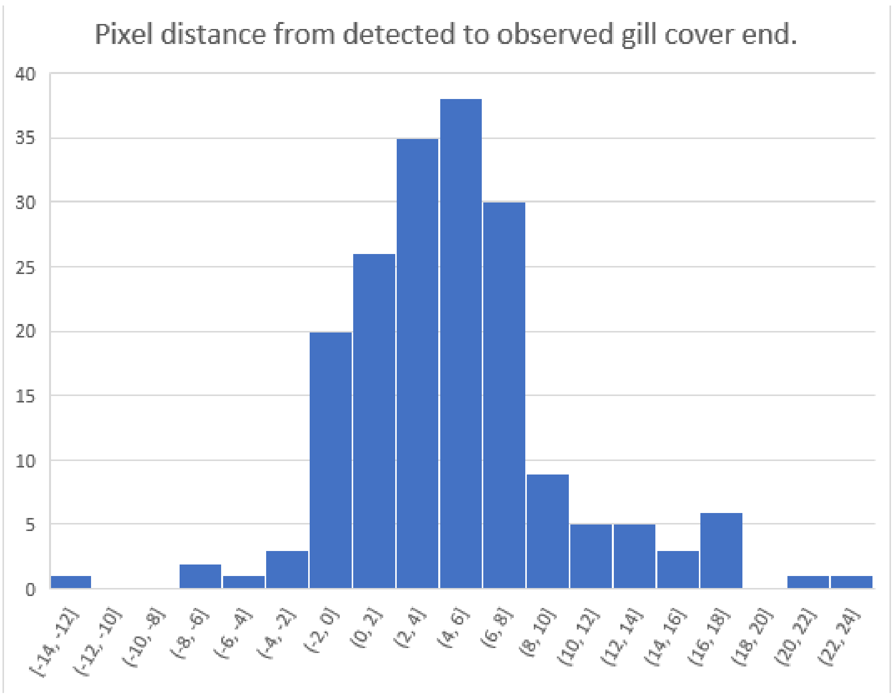


FIGURE 4.4: Histogram of the distances between automatically detected gill cover ending, to the manually picked out. The distances are only measured in columns, as it is assumed the fish is horizontally positioned. It can be seen that in most cases, the gill cover end is detected slightly to the right of where it actually is. This was intended, and is suitable for this project.

Based on these results and discussions, the method could be defined as a success. It performs significantly better than a statistical guess.

4.3 Spot detection - Experiments and results

In order to experiment and obtain results for spot detection methods, all spot centers in all of the images were manually detected. The code that was used to register the spots, is found in appendix D.7 Code: Manual Spot Detection, Manual Gill Detection on page 165. The code that was used to compare the automatically detected spots to the manually detected spots is found in appendix D.8 Spot Coordinates Comparison on page 167.

The spots were assigned into three different categories shown in table 4.3 along with the number of manual detections for each. Further description of the spots is given in the bullet points below the table. Fig. 4.5 shows an example image where spots are partitioned into the three different spot categories.

Spot Categories		
Distinct spots	Should be easy to detect	474 cases
Complex spots	Could be harder to detect	542 cases
Uncertain	Does not need to be detected	314 cases

TABLE 4.3: The three spot categories and the total number of cases in all images used in data-set 1.

1. Distinct spots: Large, dark, distinct and easily separable from its surroundings. These should be easy to detect.
2. Complex spots: Small/blurry/light and/or hardly separable from its surroundings.
3. Uncertain spots: Very small, non distinct spots that may not even be a spot. They could also be a lighter tone in the skin, or simply noise. These are included as they should be distinguished from False detections.

A spot that is detected close to either a Distinct or Complex spot, is counted as a **True** detection. By adding together the amount of Distinct and Complex spots(see table 4.3), there are a potential of **1016** True detections. When the spot detection percentage later is presented, it is in comparison to this number. If a detection occurs too far away from any manually detected spot, it will be counted as a **False** detection. Detections in the Uncertain category is counted as neither True nor False.

Experiments were conducted for both Simple Blob Detection (existing method; See 3.7.1 Simple Blob Detection Method on page 57) and Dark Point Detection (Custom made for this project; See 3.7.2 Dark Spot Detection method on page 59).

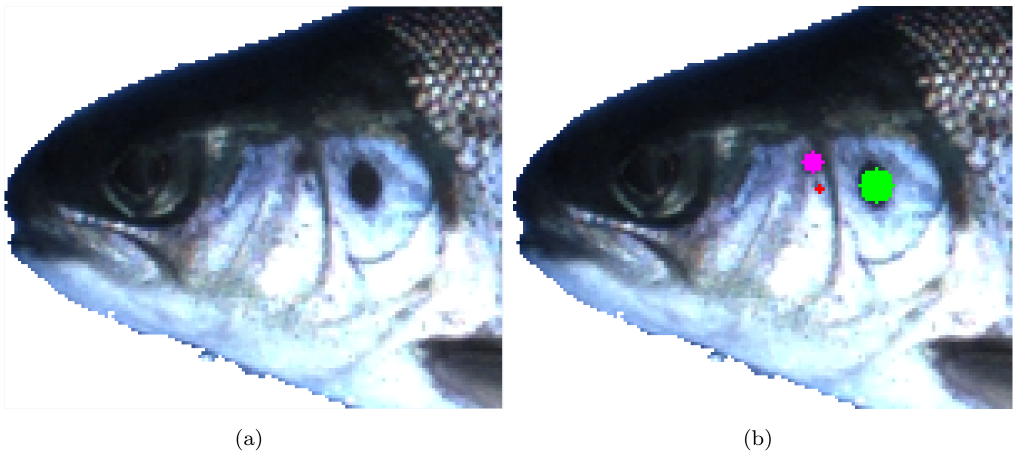


FIGURE 4.5: Manual detection of spots. The green dot means that the spot has been labelled as Distinct. This spot is large, dark and separable from its surroundings. The purple dot represents a Complex spot. This spot is small and slightly blurry. It is also connected to the vertical line to its right, which makes it harder to separate from the surroundings. The spot just below the complex spot, marked red, represents a Uncertain spot. If this is a spot, it is only the size of a few pixels. This even makes the form change between different images of the fish.

A highlight of the results is shown in fig. 4.33 and fig. 4.34.

Highlighted Results on data-set 1					
Result type	Strict	Balanced	Tolerant	Tolerant+	Of Total
Distinct spots	367	419	451	474	474
Complex spots	131	293	437	534	542
Spot Detection%	49.02%	70.08%	87.40%	99.21%	1016
False Detections%	0.60%	4.24%	52.01%	91.32%	

TABLE 4.4: A variety of results from spot detection on data-set 1. The first three columns are results from combinations of Simple Blob and Dark Spot Detection. The last column, Tolerant+, is the result of only Dark Spot Detection.

Highlighted Results on data-set 2					
Result type	Strict	Balanced	Tolerant	Tolerant+	Of Total
Distinct spots	66	69	71	71	71
Complex spots	198	458	493	501	507
Spot Detection%	45.67%	91.18%	97.58%	98.96%	578
False Detections%	4.32%	20.62%	50.00%	80.39%	

TABLE 4.5: A variety of results from spot detection on data-set 1. The first three columns are results from combinations of Simple Blob and Dark Spot Detection. The last column, Tolerant+, is the result of only Simple Blob Detection.

Detailed experiments were conducted by both Simple Blob Detection and Dark Point Detection data-set 1, to explore the behavior of the methods, but also the features of the spots. Afterwards, the two methods were combined to return better results. For comparison, some similar experiments on data-set 2 were also performed. The results are commented along the way, and a summarizing discussion can be found in 4.3.5 Summary and Discussion of blob detection results on page 108.

4.3.1 Spot detection by Simple Blob Detection on data-set 1

The experiments in this section explores gill spot detection by Simple Blob Detection, with varying parameters. The following list describes the experiments that are presented:

1. Varying and determining the maximum distance deviation allowed for a spot to classify as a True detection
2. Varying minimum and maximum threshold value
3. Varying Circularity ratio
4. Varying Convexity ratio
5. Varying Inertia ratio
6. Varying minimum Area size
7. Varying several parameters, based on the previous results.

The standard values of the parameters are shown in table 4.6. These parameters are changed in turn, while all other parameters remain constant. In experiment 7 however, several parameters are varied in each attempt.

Standard Blob Detector parameters	
minArea	25
maxThreshold	150
minThreshold	10
minCircularity	0.5
minConvexity	0.5
minIntertiaRatio	0.01

Table 4.6: These are the standard parameters set for the blob detection algorithm. While one parameter changes for experimentation, the remaining ones are locked to these values.

Experiment/Result 1 - Euclidean distance to define a True detection

To define what a True detection is, experiments were conducted where the maximum allowed distance was varied as shown in table 4.7.

This experiment is not for evaluating the blob detection algorithm, but rather for estimating where the threshold for ”success” should be put. An eventual False detection can also occur because of inaccurate picking of the manual detected spot centroids. Therefore, a certain tolerance is necessary.

Results when varying maximum allowed pixel distance						
Max pixel distance	1	2	3	5	8	Of total
Distinct spots	274	425	446	450	450	474
Complex spots	186	332	376	382	386	542
Uncertain	22	42	50	52	52	314
False	826	509	436	424	420	
Spots detected(%)	45.28%	74.51%	80.91%	81.89%	82.28%	1016
False(%)	63.15%	38.91%	33.33%	32.42%	32.11%	

Table 4.7: Varying maximum allowed euclidean pixel distance from observed to detected spot centroid. It can be seen here that increasing the allowed distance lead to more spots being defined as detected. It is clear that 1 pixel would be too little, while increasing could cause misclassifications. Also, an inaccurate detected spot would cause worse performance in actual individual recognition. By these alternatives, it would therefore be reasonable to use 3 or 5 as the preferred distance.

The results in table 4.7 shows that an allowed pixel distance of 1 or 2 is such strict that it has significant influence on the results. and it is evident that the amount of spot detections increases considerably from max pixel distance 1 to 3. From pixel distance 3 to 8 however, there are little change in the amount of spots detected. This means that blob detector usually detects the spot centroid at a distance less or equal to 3 pixels from the manually detected spot centroid. Therefore, 3 is used as the maximum allowed pixel distance, in the following experiments. In these images, 1 pixel equals approximately 0.24 mm.

When the distance allowed is raised as high as 8, it is possible that some of the ”True” detections actually are detections near a spot, and not in the spot itself. Besides, larger deviation is unfavourable considering the quality of spots used as interest points for object matching (See 2.4 Detection of spot-like structures on page 26). The fact that few more spots are detected when the pixel distance is raised high, indicates that there seldom are False detections close to actual spots.

Experiment/Result 2 - Varying threshold parameters

This experiment was conducted in order to determine which thresholds to set. This also explores in which intensity value intervals most spots are included. The results are shown in table 4.8.

Result when varying minimum and maximum threshold								
Threshold min-max	10-50	10-100	10-150	0-150	5-150	10-200	10-255	Of total
Distinct spots	194	443	446	446	445	448	448	474
Complex spots	27	283	376	376	366	399	402	542
Uncertain	1	23	50	50	47	64	76	314
False	186	330	436	436	442	585	803	1330
Spots detected(%)	21.75%	71.46%	80.91%	80.91%	79.82%	83.37%	83.66%	1016
False(%)	45.59%	30.58%	33.33%	33.33%	34.00%	39.10%	46.44%	

TABLE 4.8: Results of blob detection when varying only the threshold parameters. It is evident that there are a significant amount of spots that have brightness between 10-100. When the interval is extended, the number of both True and False detections increase.

The results in table 4.8 shows that a large proportion of the spots has an intensity value below 100, especially the Distinct ones. However, an additional 83 Complex spots are detected when increasing to 150, and another 33 when increasing to 200. After this, further extension mainly leads to more False detections. The results indicate that a threshold between 10-150 or 10-200 could be favourable.

What is interesting is that the results from using 5-150 is worse than using 10-150, which gets the same results as 0-150. This could be explained by the detector's incrementing behaviour (See 3.7.1 Simple Blob Detection Method on page 57). If each increment is for example 10, 0-150 and 10-150 could give the same result, as they do, while 5-150 could get different results.

Experiment/Result 3 - Varying Circularity parameter

Table 4.9 shows the results when the circularity parameter is increased from 0.3 to 0.9.

Result when varying Circularity parameter					
Circularity	0.3	0.5	0.7	0.9	Of total
Distinct spots	453	446	420	37	474
Complex spots	381	376	300	11	542
Uncertain	53	50	35	1	
False	872	436	143	0	
Spots detected(%)	82.09%	80.91%	70.87%	4.72%	1016
False(%)	54.61%	33.33%	15.92%	0%	

Table 4.9: Stricter conditions regarding the circularity of the spots will make less spots be detected. Values like 0.5 and 0.7 allows for a decent amount of detections, while still limiting the number of False detections. The results shows that the spots in these images are generally somewhat circular, but not completely, as most spots fails to be detected when the parameter is 0.9.

Table 4.9 shows that increasing the circularity parameter from 0.3 to 0.5 results in almost the same amount of True detections, while decreasing the number of False detections considerably. Further increasing to 0.7 leads to even fewer False detections, but also has an impact on the amount of True detections. Parameter value 0.9 leads to no False detections at all, but also very few True detections. It is obvious that the Circularity parameter has an important role when filtering out excess detections.

Experiment/Result 4 - Varying Convexity parameter

Table 4.10 shows the results for experiments where the Convexity parameter is increased from 0.3 to 0.9.

Result when varying Convexity parameter					
Convexity	0.3	0.5	0.7	0.9	Of total
Distinct spots	446	446	446	434	474
Complex spots	376	376	376	335	542
Uncertain	50	50	50	41	314
False	436	436	436	211	
Spots detected(%)	80.91%	80.91%	80.91%	75.7%	
False(%)	33.33%	33.33%	33.33%	20.66%	

Table 4.10: The results are very similar to the above until 0.9 is reached. This shows that the spots of an Atlantic salmon are mostly convex to some degree.

The results changes nothing at all, when the Convexity parameter is increased from 0.3 to 0.7. Changes occur first When the ratio is set as high as 0.9. The number of False detections are more than halved, but it also has an impact on the detection of both Distinct and Complex spots. From this experiment, it is evident that the spots of Atlantic Salmon generally are convex.

Experiment/Result 5 - Varying Inertia parameter

Table 4.11 shows the results for experiments where the inertia ratio parameter is increased from 0.01 to 0.3.

Result when varying Inertia parameter					
Inertia ratio	0.01	0.10	0.20	0.30	Of total
Distinct spots	446	446	446	440	474
Complex spots	376	376	370	336	542
Uncertain	50	50	47	43	314
False	436	425	352	281	
Spots detected(%)	80.9%	80.9%	80.31	76.4%	
False(%)	33.33%	32.77%	28.97%	25.55%	

Table 4.11: Results while varying minimum inertia. The results are similar from 0.01-0.20, meaning that most spots are not similar to lines, but instead resemble circular shapes.

When the inertia ratio is increased from 0.01 to 0.1, no True detections are lost, but 9 False detections. Further increasing to 0.2 leads to the rejection of another 73 False detections in exchange with the loss of 6 Complex spot detections. When the parameter is set as 0.3, another 71 False detections are lost, but also 40 True detections. Thus, somewhere around 0.20 seems like a reasonable inertia ratio for these images.

Experiment/Result 6 - Varying Area parameter

Table 4.12 shows the results for the experiments where the minimum area size parameter was increased from 3 to 50.

Results when varying minimum area size						
min Area	3	10	15	25	50	Of total
Distinct spots	411	448	449	436	430	474
Complex spots	393	429	423	376	205	542
Uncertain	144	105	79	50	12	314
False	7684	2188	1089	436	183	
Spots detected(%)	79.13%	86.32%	85.83%	79.92%	62.50%	1016
False(%)	89.02%	69.02%	53.38%	33.33%	22.05%	

TABLE 4.12: The most spots are successfully detected at a threshold of 10, although, there are many false detections. The failure percentage decreases as the area parameter is increased, because then smaller possible spots or spot-like patterns are discriminated against. However, this has a negative impact on the detection of complex spots, as they are trickier to detect and often smaller than the distinct ones.

When the minimum allowed area is raised from 3 to 10, the number of True detection actually increases. This must be because the blob detector uses increments? Except from this, the number of True detections decreases when the minimum area allowed is increased. There are most True detections when the minimum area is set to 10. But again, it can be seen that 1 more Distinct spot is detected when the parameter is raised to 15. When the parameter is raised further, it especially has a negative effect on Complex spots, as these in general are smaller. Generally, it can be seen that increasing the minimum area parameter reduces the number of False detections significantly.

From min Area size 3 to 10, it can be seen seen that actually more spots are detected when the min area size is raised. This is because the minimum area parameter also will scale the max area parameter (See 3.7.1 Simple Blob Detection Method).

Result/Experiment7 - Combining parameters

In this last experiment, some different combinations were tested in order to examine the behaviour of the detector further, for example to test if it is possible to detect all the spots, or if it is possible to detect a decent amount of spots while having none False detections. The combinations are represented in table 4.13, and the results are shown in table 4.14.

Various combinations						
Combination no.	1	2	3	4	5	6
Circularity	0.7	0.7	0.3	0.1	0.8	0.7
Convexity	0.9	0.9	0.5	0.1	0.9	0.9
Area	25	10	10	10	10	15
Threshold	10-150	10-200	10-200	10-250	10-130	10-170
Min inertia	0.2	0.2	0.01	0.01	0.2	0.2
Comment	Strict	Balanced	Tolerant	Very tolerant	Balanced	Balanced

TABLE 4.13: This table shows the parameter values for 6 different combinations that were tried. The bottom row gives a short comment on the function's purpose, where "strict" means that the purpose is to have few false detections, and "tolerant" means that it wants to detect as many spots as possible.

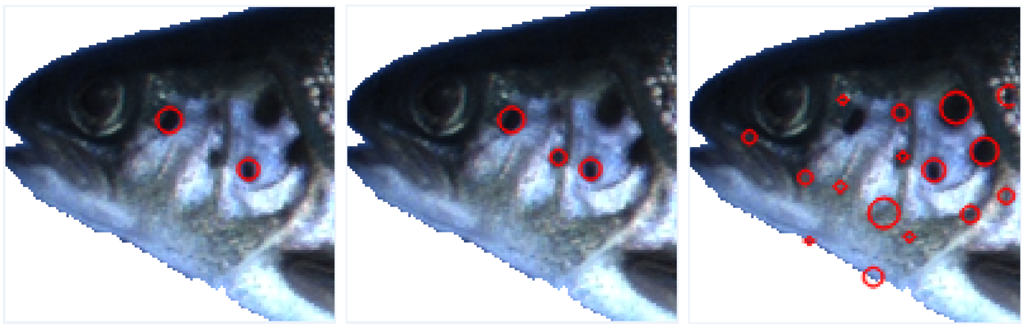
Results of various combinations							
Combination no.	1	2	3	4	5	6	Of total
Distinct spots	420	425	452	453	376	423	474
Complex spots	296	385	446	436	228	363	542
Uncertain	34	111	152	182	26	73	314
False	138	1375	4407	7027	319	336	
Spots detected%	70.47	79.72	88.39	87.50	59.49	77.36	1016
False%	15.54	59.89	80.76	86.77	33.61	28.12	

TABLE 4.14: Results of the combinations shown in table 4.13. Combination 1 has the fewest false detections, but not relatively few True detections compared to the other combinations. Combination 3 has the best detection rate, but many False detections. Combination 6 performs almost as good as 2, but has way less false detections. It is evident that combination 4 and 5 are outperformed compared to the others, especially since Uncertain spots are not considered.

The experiments shows that the blob detector fails to detect all spots in data-set 1 in one attempt, even when all the parameters are set to tolerant values. This does however not confirm that all the spots could not be detected with multiple attempts with varying parameters. This is especially regarding the minimum area parameter, where large spots might be left out when the parameter is low.

The results are generally decent concerning detection of the distinct spots. It is the complex spots that are harder to detect without many False detections.

From table 4.14, it could be suggested that combination 1, 3 and 6 could be favourable for different types of results. For the rest of this section, combination 1 will be regarded as Strict detection, combination 6 as Balanced detection and combination 3 as Tolerant detection. Fig. 4.6 and 4.7 shows two typical examples for the results using these combinations.



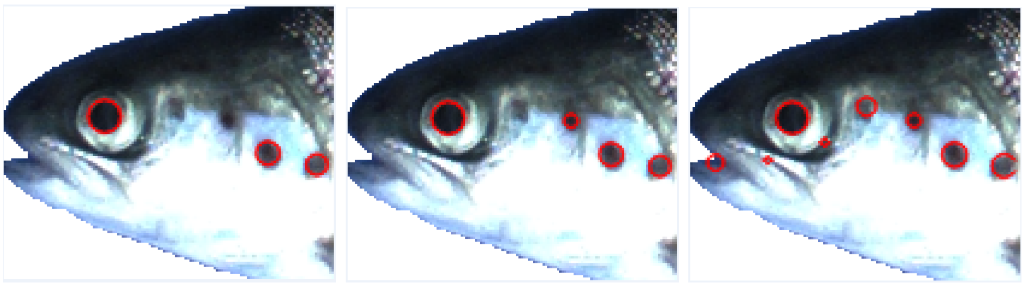
(a) Combination 1: Strict detection. (b) Combination 6: Balanced detection. (c) Combination 3: Tolerant detection.

FIGURE 4.6

The detector seems to have a harder time detecting correct spots in darker images, which can be seen in fig. 4.6. Especially when the spot is in contact with darker parts of the fish like the gill opening or upper part of the fish. This is shown in both combination 1 and 6.

What is curious is that while combination 3 is supposed to be more tolerant than the other 2, it fails to detect the large spot just right from the eye, where the stricter combinations succeed. The combinations here had minimum area size respectively 25, 15 and 10. This is an example of how the reduction of minimum area could result in worse results for larger spots.

Fig. 4.7 is significantly brighter than fig. 4.6. This causes different challenges. Large and dark spots are more distinct in these brighter images. For example, the eye is usually detected in these types of images. Regarding the strict combinations which had only 138 False detections, many of these are caused by the eye. Even though the image is bright and with good contrasts, the strict detection fails to detect the more complex spot in the middle of the image. Combination 6 seems to function well on these types of images, as all spots are detected, with no more false detections than the Strict combination.



(a) Combination 1: Strict detection. (b) Combination 6: Balanced detection. (c) Combination 3: Tolerant detection.

Figure 4.7

These examples show that the combinations may have different advantages on different types of images and spots. Brighter images leads to better results.

None of the blob detector combinations could be used alone, but combining them in a larger algorithm, might give a more complete result. Also, other restrictions could be made. The eye can easily be disregarded as a spot because of its position, and this would make the stricter algorithm cause fewer false detections.

4.3.2 Spot detection by Dark Point Detection - Experiments and Results

As with the experiments with Blob Detection, some parameter values are chosen to be varied, as listed below.

- Kernel size
- σ
- p-factor

The use of these parameters are explained in the pseudocode found in 3.7.2. Briefly summarized, the kernel size is the size of the kernel used when applying gaussian blur, and σ determines its distribution. The p-factor is a threshold factor that is multiplied with the median of the image, to set a threshold value that decides if a local extrema(pixel) is dark enough to be considered as part of a spot.

In these experiments, it would make little sense to vary one parameter at the time, as they strongly influence each others functionalities. For example, a larger kernel size would require a larger p-factor, as the pixels displaying the spots would become brighter. It would neither make sense to vary σ on a small sized kernel.

An overview of the experiments is shown below:

1. Varying and determining the maximum euclidean distance deviation allowed for a spot to classify as a True detection. This is the same experiment that was conducted in 4.3.1 Experiment/Result 1 - Euclidean distance to define a True detection.
2. Various combinations with a small kernel size of 3 to 5
3. Various combinations with a medium kernel size of 9
4. Various combinations with a large kernel size of 15

Experiment/Result 1 - Euclidean distance to define True detection

Similarly to what was done regarding blob detection, experiments are conducted using different values for maximum euclidean distance to define a successful spot detection. This was done both in order to test the accuracy of the method and to decide which euclidean distance to use for further experiments.

The parameter values that are used while varying the max allowed euclidean distance, is shown in table 4.15, and the results in table 4.16.

Parameter values in Experiment 1	
Kernel Size	5
σ	3
p-factor	1.2

Table 4.15: The parameter values used in experiment 1

Results when varying maximum allowed pixel distance						
Pixel distance	1	2	3	5	8	Of total
Distinct spots	150	394	462	473	473	474
Complex spots	86	388	506	532	532	542
Uncertain	26	103	140	146	177	314
False	9504	8881	8658	8605	8584	
Spots detected%	23.23%	76.97%	95.28	98.92%	98.92%	1016
False(%)	97.32%	90.94%	88.65%	88.11%	87.90%	

Table 4.16: At an allowed pixel distance of 1, few spots are defined as detected. It increases significantly when raised to 2 and 3, and stabilizes at 5. Increasing from 5 to 8 has no effect on the number of True detections.

Based on the results in table 4.16, the spot center detections of the Dark Point Detector is less accurate than the ones of the Blob Detector. It is shown that the results are significantly affected when the allowed distance is as low as 1. Allowed distance 2 and 3 results more True detections, but it also increases significantly from 3 to 5.

Thus, 5 will be used as the maximum allowed pixel distance to define True detection using Dark Point Detection.

Experiment/Result 2 - Small kernel size

Table 4.17 shows the input parameters for the attempts in this experiment, which focuses on small kernels, while the the p-factor is varied.

Input combinations for Experiment 2 - Small kernel							
Combination no.	1	2	3	4	5	6	7
Kernel Size	3	5	5	5	5	5	5
p factor	1.0	0.2	0.4	0.6	0.8	1.0	1.2
σ	3	3	3	3	3	3	3

Table 4.17: Combination 1 is a single attempt with a kernel size as small as 3. In experiments 2-7, the threshold factor, p, was incrementally increased from strict to tolerant.

Results for Experiment 2 - Small kernel								
Combination	1	2	3	4	5	6	7	Of Total
Distinct spots	473	190	455	471	473	473	473	474
Complex spots	534	51	271	460	510	529	532	542
Uncertain	166	6	28	68	101	141	146	314
False	15534	816	3616	6503	7831	8454	8605	
Spots detected(%)	99.11%	23.72%	71.46%	91.63%	96.75%	98.62%	98.92%	1016
False(%)	92.98%	76.76%	82.75%	86.68%	87.84%	88.09%	88.11%	

Table 4.18: Combination 1, with kernel size 3, fails to detect only 9 spots. However, many False detections also occur. Combination 7 scores almost as good as combination 1, but has nearly half the amount of False detections. Combinations 2 - 7 shows how gradually more spots are detected when the threshold factor, p, is increased.

The results shows that a lower p-factor increases the number of both True and False detections, and that the share of false detections is high also for very low p-value. This indicates that many excess spots are detected in dark areas, and that a too strict threshold is not favourable as it cuts too many True detections.

Combination 7 performs almost as good as combination 1 in number of True detections, but has almost half the number of False detections. This indicates that increasing the kernel size is effective for such small kernels, when cutting False detections.

Experiment 3 - Medium kernel size

Table 4.19 and 4.20 shows input parameters and results for experiments with a medium kernel size. Here, the focus is mainly on varying the σ.

Input combinations for Experiment 3 - Medium kernel						
Combination no.	1	2	3	4	5	6
Kernel Size	9	9	9	9	9	9
p factor	0.8	0.8	0.8	0.8	1.4	1.8
σ	3	2	1	0.5	3	3

Table 4.19: In combinations 1 to 4, σ is decreased while other parameters are held constant. Combinations 5 and 6 tests for more tolerant p-values without decreasing σ.

Results for Experiment 3 - Medium kernel							
Combination	1	2	3	4	5	6	Of Total
Distinct spots	467	471	474	474	470	470	474
Complex spots	430	485	522	532	479	479	542
Uncertain	55	78	138	158	101	101	314
False	2986	4372	11420	25232	3375	3379	
Spots detected(%)	88.29%	94.09%	98.03%	99.02%	93.41%	93.41%	1016
False(%)	82.07%	80.87%	90.97%	95.59%	76.27%	76.29%	

Table 4.20: In combinations 1-4, σ is decreased. This leads to many True detections, but even more False detections. Combinations 5 and 6 shows that there is little change when the p-value is increased to very tolerant values.

The results show that reducing σ leads to more detections, both true and false. Combination 5 and 6 also shows how a large kernel size compared with a not so small σ will cause less spots to be detected, and that increasing the threshold factor p is useless in this case.

Experiment 4 - Large kernel size

Table 4.21 and 4.22 shows input parameters and results for experiments with a large kernel size. The size was set to 15, as this is a typical size of the larger spots in the data set. Here, the σ value is further experimented with.

Combinations for Experiment 4 - Large kernel						
Combination	1	2	3	4	5	6
Kernel Size	15	15	15	15	15	15
p factor	1.0	1.0	1.0	1.0	1.0	1.0
σ	1	1.5	2	3	5	9

TABLE 4.21: The σ value is increased in every attempt, while the other inputs are held constant.

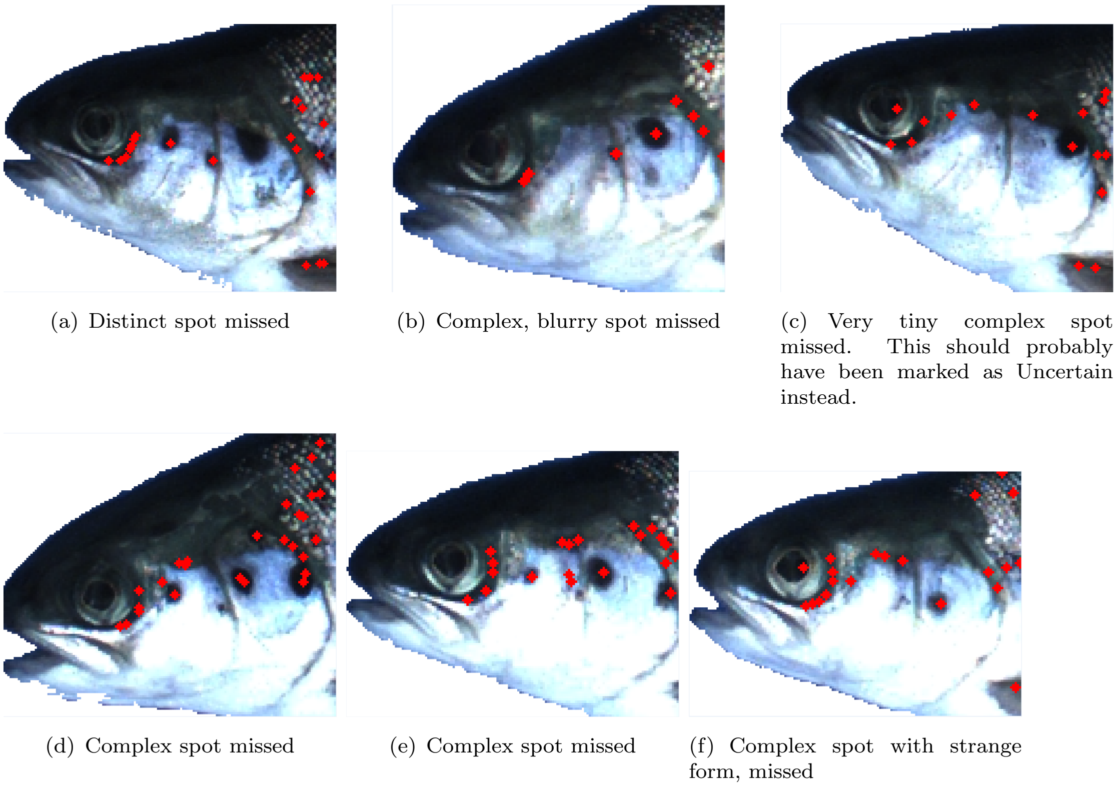
Results for Experiment 4 - Large kernel							
Combination	1	2	3	4	5	6	Of Total
Distinct spots	474	470	469	458	403	325	474
Complex spots	534	526	505	365	175	91	542
Uncertain	167	137	95	43	4	2	314
False	12359	7049	4137	1913	819	511	
Spots detected(%)	99.21%	98.03%	95.87%	81.00%	56.89%	40.94%	1016
False(%)	91.32%	86.15%	79.47%	68.84%	58.46%	55.01%	

TABLE 4.22: Combination 1 has the highest number of True detections of all the experiments by Dark Point Detection on data-set 1. When σ is gradually increased, the number of both True and False detections decrease.

The results implies that varying the σ has similar effects as to varying the kernel size, while holding other factors constant. It is therefore possible to use a large kernel to detect both many, and few spots, by varying the σ value.

Missed Spots

In these experiments, it is shown that Dark Spot Detector is able detect close to 100% of the spots. For example, Combination 7 of table 4.18 managed to detect 98.92% of the spots, while still having a false detection share of 88.11%. Only 11 spots were not detected. Fig. 4.8 shows and explains the case of each missed spot. The figure only shows 6 images, as the rest of the failures was caused by the same spot, in the other images of that specific fish. It should also be noted that some of these failures did not occur on the other images of the same fish, such as the case of the distinct spot in (a).


(a) Distinct spot missed

(b) Complex, blurry spot missed

(c) Very tiny complex spot missed. This should probably have been marked as Uncertain instead.

(d) Complex spot missed

(e) Complex spot missed

(f) Complex spot with strange form, missed

FIGURE 4.8: All the cases where some spots were not detected by Tolerant Dark Point Detection.

Example Results of Dark Point detection on data-set 1

Result wise, the Dark Point Detector is able to detect as almost every spot with tolerant parameters. The Simple Blob Detector on the other hand, is more accurate and has fewer False detections.

In similarity to what was done with Simple Blob Detector in 4.3.1 Result/Experiment7 - Combining parameters, three different types of combinations are chosen to present the variety of results for the Dark Spot Detector. Combination 7 in Experiment/Result 2 - Small kernel size is chosen as the "Tolerant" example, Combination 5 in Experiment 3 - Medium kernel size is chosen as "Balanced", and Combination 5 in Experiment 4 - Large kernel size is chosen as "Strict".

Fig.4.9 and 4.10 shows examples of how the method performs with some varying parameters, on the same images that were used as examples for Simple Blob Detection.

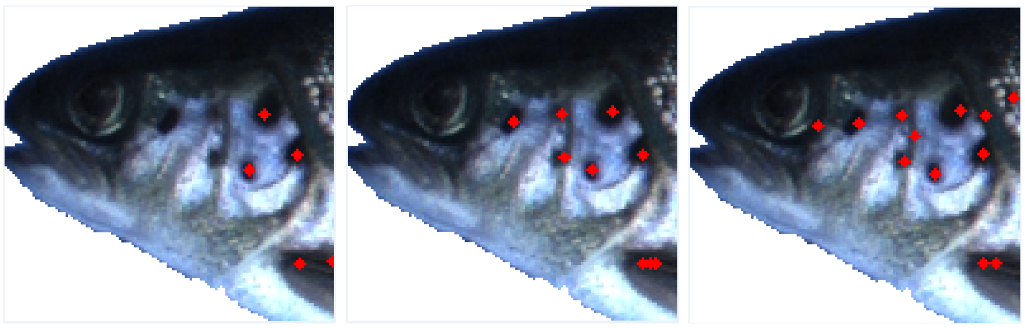

(a) Strict detection (b) Balanced detection (c) Tolerant detection

FIGURE 4.9

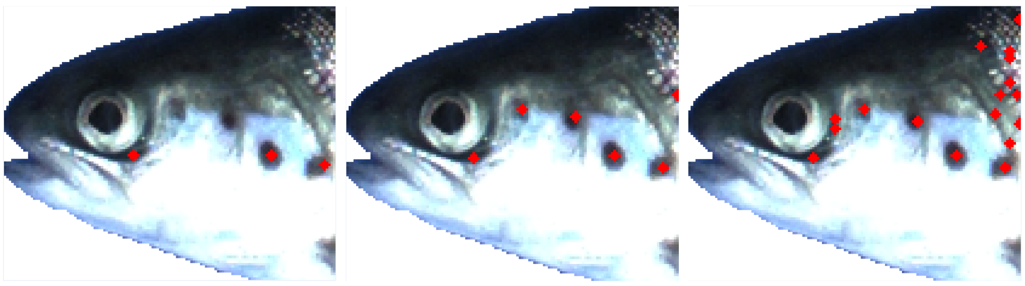

(a) Strict detection (b) Balanced detection (c) Tolerant detection

FIGURE 4.10

The strict detections, (a), in both figures illustrates that the method has difficulties in detecting smaller/blurry spots, which are close to other dark areas. It also has difficulties in disqualifying non existent spots, such as the pectoral fin in fig. 4.9 (a), and the corner of the eye in fig. 4.10 (a). However it can be seen that when the parameters are stricter, many points are correctly disqualified, especially in areas of high frequency, such as the scales in the top right corner.

4.3.3 Combining Blob Detection and Dark Point Detection

The results of Blob Detection and Dark Point Detection were combined together to test for the possibility to improve results. It was done in the following way: For every spot detected in blob detection, check if custom detection has detected at least one spot nearby. Nearby is here defined as 9x9 neighbourhood. The code for this, is found in D.8 Spot Coordinates Comparison on page 167.

The results from Simple Blob Detection was chosen to be the final result, because these were the most accurate per spot detected.

These results describe how the two methods may be used together to disqualify False detections, and to strengthen the credibility of spots that are detected by both algorithms.

In this section, it is referred to Strict, Balanced, and Tolerant - Simple Blob Detection and Dark Point Detection. These are the same example combinations that were presented in Result/Experiment7 - Combining parameters, page 91, and Example Results of Dark Point detection on data-set 1, page 99.

The results are presented and compared to the previous outcome of using only Simple Blob Detection. The goal was to keep the number of True detections, while eliminating as many False detections as possible.

Table 4.23 shows that the majority of the False detections by Strict Blob Detection can be removed by combining with Dark Spot detection, while loosing few True detections. As earlier suspected, a large proportion of the False detections regarding Strict Simple Blob Detecition, is caused by the eye of the fish. The Dark Spot Detector tries to eliminate the eye as a spot candidate. This may be one of the reasons that the combination of these seem to have favourable results.

Combining **Strict** Blob Detection with Dark Point Detection				
Dark Point Detection	Tolerant	Balanced	Strict	Before
Distinct	419	420	367	420
Complex	293	283	131	296
Uncertain	34	30	1	34
False	33	27	3	138
Spot Detection%	70.08%	69.19%	49.02%	70.47%
False Detections%	4.24%	3.55%	0.60%	15.54%

TABLE 4.23: Few True detections are lost when applying Strict/Tolerant, or Strict/Balanced combinations. When the combination is Strict/Strict a significant amount of True detections are lost, but it also result in only 3 False detections.

Table 4.24 shows that again, combining causes less False detections. Still, some True detections are also cut, especially in the Balanced/Balanced and Balanced/Strict combinations.

Combining **Balanced** Blob Detection with Dark Point Detection				
Dark Point Detection	Tolerant	Balanced	Strict	Before
Distinct	422	423	367	423
Complex	353	329	134	363
Uncertain	67	52	2	73
False	126	89	16	336
Spot Detection%	76.28%	74.02%	49.31%	77.36%
False Detections%	13.02%	9.97%	3.08%	28.12%

Table 4.24: Few True detections are lost using a Balanced/Tolerant combination. Balanced/Balanced causes some loss, while Balanced/Strict loses a significant amount. Meanwhile, many False detections are removed.

Table 4.25 shows that a Tolerant/Tolerant combination is able to keep a high amount of True detections, while eliminating many False detections. Still, about half of the total detections are False detections, but the improvement is significant. The two other combinations also show that many False detections are cut, but also some True detections.

Combining **Tolerant** Blob Detection with Dark Point Detection				
Dark Point Detection	Tolerant	Balanced	Strict	Before
Distinct	451	450	387	452
Complex	437	400	173	446
Uncertain	103	74	3	152
False	1074	550	94	4407
Spot Detection%	87.40%	83.66%	55.12%	88.39%
False Detections%	52.01%	37.31%	14.31%	80.76%

Table 4.25: The Tolerant/Tolerant combination causes few True detections losses, while eliminating many False Detections. The Tolerant/Balanced and Tolerant/Strict combinations also gets rid of many False detections, but the results are not peculiar compared to the other combinations.

4.3.4 Testing on data-set 2

Some experiments was also conducted on data-set 2. These were performed in order to evaluate whether the methods are applicable on different kinds of images. The fish in these were brighter, a bit more developed, and the images had larger resolution. Therefore a few scale changes had to be applied.

4.3.4.1 Testing Simple Blob Detection on data-set 2

Six similar combinations to the ones that was used on data-set 1 (See 4.3.1 Result/Experiment7 - Combining parameters, 91), were also used here, only the area parameter was changed to four times the size. The input variables for the six combinations are shown in 4.26, and the results are shown in 4.27.

Input combinations for Simple Blob Detection on data-set 2						
Combination no.	1	2	3	4	5	6
Circularity	0.7	0.7	0.3	0.1	0.8	0.7
Convexity	0.9	0.9	0.5	0.1	0.9	0.9
Area	100	40	40	40	40	60
Threshold	10-150	10-200	10-200	10-250	10-130	10-170
Min inertia	0.2	0.2	0.01	0.01	0.2	0.2
Comment	Strict	Balanced	Tolerant	Very tolerant	Balanced	Balanced

TABLE 4.26: This table shows the parameter values for 6 different combinations that were tried. The bottom row gives a short comment on the function's purpose, where "strict" means that the purpose is to have few false detections, and "tolerant" means that it wants to detect as many spots as possible.

Results for Simple Blob Detection on data-set 2							
Combination	1	2	3	4	5	6	Of Total
Distinct	65	69	71	71	54	68	71
Complex	216	461	485	501	140	349	507
Uncertain	5	42	64	146	2	12	191
False	300	430	1751	2943	233	352	
Spot Detection%	48.62%	91.70%	96.19%	98.96%	33.56%	72.15%	
False Detections%	51.19%	42.91%	73.85%	80.39%	54.31%	45.07%	

Table 4.27: The poor results in combinations 1, 5 and 6 indicate that many spots in data-set 2 are somewhat bright, and therefore sensitive to the strict maximum threshold values of these combinations. With more tolerant thresholds, as in 2, 3 and 4, almost all spots are detected. Combination 2 even has the lowest False rate, while still having a high percentage of detected spots.

The results shows that almost all spots can be detected in data-set 2. By comparing combination 2 to combinations 1, 5 and 6, it is evident that here, it is favourable to conduct tolerant thresholding, while having strict values for Circularity, Convexity and Inertia.

4.3.4.2 Testing Custom detection on data-set 2

Then input combinations for the Dark Spot Detections performed on data-set 2, are shown in fig. 4.28 and the results are shown in 4.29.

Inputs for Dark Spot Detection on data-set 2			
Combination	1 Tolerant	2 Balanced	3 Strict
Kernel Size	11	59	59
p factor	1.0	1.0	1.0
σ	3	5	7

Table 4.28: Input combinations with different kernel sizes and σ

Results for custom detection on data-set 2				
Combination	1	2	3	Of Total
Distinct	71	71	68	71
Complex	499	386	207	507
Uncertain	73	13	3	191
False	4619	724	294	
Spot Detection%	98.62%	79.07%	47.58%	
False Detections%	87.78%	60.64%	51.40%	

TABLE 4.29: Results for Dark Spot Detection on data-set 2.

The results show that Dark Spot Detection is able to detect almost all spots also on data-set 2, when the parameters are tolerant. Still, many False Detections ocurr.

4.3.4.3 Combining blob detection and custom detection on data-set 2

As was done also with data-set 1, Simple Blob Detection and Dark Spot Detection were combined. Also here, 9 combinations og Tolerant, Balanced and Strict were used. The results are shown in tables 4.30, 4.31 and 4.32.

Combining **Strict** blob detection with custom detection				
Custom detection	Tolerant	Balanced	Strict	Before
Distinct	65	65	63	65
Complex	216	213	165	216
Uncertain	5	3	1	5
False	62	36	9	300
Spot Detection%	48.62%	48.10%	39.45%	48.62%
False Detections%	17.82%	11.36%	3.78%	51.19%

TABLE 4.30: Strict blob detection, combined with custom detection.

Combining **Balanced** blob detection with custom detection				
Custom detection	Tolerant	Balanced	Strict	Before
Distinct spots	69	69	66	69
Complex spots	458	369	198	461
Uncertain	35	10	2	42
False	146	77	12	430
Spot Detection%	91.18%	75.78%	45.67%	91.70%
False Detections%	20.62%	14.67%	4.32%	42.91%

TABLE 4.31: Balanced blob detection, combined with custom detection.

Combining **Tolerant** Simple Blob Detection with Dark Spot Detection				
Custom detection	Tolerant	Balanced	Strict	Before
Distinct	71	71	68	71
Complex	493	386	207	501
Uncertain	73	13	3	146
False	637	297	89	2943
Spot Detection%	97.58%	79.07%	47.58%	98.96%
False Detections%	50.00%	38.72%	24.25%	80.39%

TABLE 4.32: Tolerant blob detection, combined with custom detection.

The results seem slightly better for data-set 2, than dataset 1. Especially when considering the Balanced—Tolerant combination in 4.31, where 91.18% of the spots were detected, with only 20.62% False detections. This is partly because of the worse performance of Simple Blob Detection on data-set 1, as it was unable to detect a significant amount of the spots. The Dark Spot Detector performs more eqaully on both data-sets, and manages to detect almost all spots in both, with tolerant parameters.

4.3.5 Summary and Discussion of blob detection results

To summarize, table 4.33 and 4.34 shows some highlighted results of blob detection on the two data-sets. The first three result columns display results where combinations of Simple Blob Detector and Dark Spot Detector have been combined to produce favourable results. The "Tolerant" column shows the absolute highest result, when only prioritizing the amount spots detected.

Highlighted Results on data-set 1					
Result type	Strict	Balanced	Tolerant	Tolerant+	Of Total
Distinct spots	367	419	451	474	474
Complex spots	131	293	437	534	542
Spot Detection%	49.02%	70.08%	87.40%	99.21%	1016
False Detections%	0.60%	4.24%	52.01%	91.32%	

Table 4.33: A variety of results from spot detection on data-set 1. The first three columns are results from combinations of Simple Blob and Dark Spot Detection. The last column, Tolerant+, is the result of only Dark Spot Detection.

Highlighted Results on data-set 2					
Result type	Strict	Balanced	Tolerant	Tolerant+	Of Total
Distinct spots	66	69	71	71	71
Complex spots	198	458	493	501	507
Spot Detection%	45.67%	91.18%	97.58%	98.96%	578
False Detections%	4.32%	20.62%	50.00%	80.39%	

Table 4.34: A variety of results from spot detection on data-set 1. The first three columns are results from combinations of Simple Blob and Dark Spot Detection. The last column, Tolerant+, is the result of only Simple Blob Detection.

4.3.5.1 Simple Blob Detection Method

The experiments indicate that it is favourable to input strict values for circularity, convexity and inertia, while holding tolerant values for area and thresholding. This enables most spots to be detected, as they vary in size and grey-level intensity.

The experiments on the blob detector does mainly focus on single parameter modification. Therefore, the results displayed may be somewhat correlated. For example, larger blobs are more likely to have a larger circularity ratio, especially in data-set 1 where the pixel resolution is low. A strict circularity ratio would therefore probably filter out many small objects, and is therefore correlated with the area parameter. More experiments while varying several combinations could be conducted to explore this further.

It was shown that different types of blobs could be detected when varying the threshold parameters and minimum area parameters. This could be a reason why the method is unable to detect all the spots in data-set 1 in a single attempt. Especially considering the area parameters, it might be that the sizes of the spots in this data-set vary such that the method is unable to detect all of them without multiple changes to this parameter.

4.3.5.2 Dark Spot Detection Method

In Dark Spot Detection, increasing the kernel size and/or decreasing the σ-value, causes both fewer True and False detections. Therefore, reducing or increasing one of the two, have similar effects.

Increasing the p-factor makes the method more tolerant, while decreasing makes it stricter. Especially from the experiments with small kernel size, it is confirmed that the distinct spots are darkest, and that the threshold has to be quite strict in order to disqualify them as spots.

It seems that a lot of the spots classified as "distinct", are robust to the size and σ of the kernel. This is because they are generally larger. The resuts show that the method detects very many false spots, compared to the Simple Blob Detector. This is however somewhat "unfair", as this algorithm may detect several local extrema within a single spot. Then only one of these extremas will be regarded as a True detections, while the others will be regarded as False. This makes the result look worse. Improving the "result-extracting" method would present the actual performance better. This could be done by merging nearby spot centers together, which is a common approach in blob detection (see 2.4.1 Grey-Level blobs and Grey-level trees) and is also done by Simple Blob Detection.

With tolerant parameters, Dark Spot Detector is able to detect nearly all spots. This makes it a favourable initial method to use when choosing candidate coordinates that could be spot centers, before combining with other methods. However, it was shown that the method is less accurate than blob detection, regarding placement of the spot centroids.

4.3.5.3 Future work

In these experiments, it is not explored whether **different** spots are detected when the parameters are changed in each attempt. Here, only numbers have been considered. For example if two different experiments has detected exactly 50 spots, this does not mean that these are the same 50 spots. In the most extreme case, the two experiments combined could have detected a total of 100 different spots. Therefore, it can not be alleged that the Simple Blob Detector is not able to detect all the spots in data-set 1. Combining several different input parameter combinations and/or a multi-scale approach as recommended by Lindeberg [27](See 2.4 Detection of spot-like structures) for blobs of varying size should be considered as important future work.

4.4 Individual Recognition Results

The algorithm was tested on 178 images of different fish; the label fish. Except for 16 of these salmon, there were two images of the same specimen, giving 340 possible correct classifications. Some images were removed due to the image not being desirable for image processing, or in some cases other methods couldn't find the desired feature(like the Fin detection-method not finding the pectorla fin), leading to 16 of the label fish only having one match.

The algorithm correctly classified 124 salmon and incorrectly classified 98 salmon, totaling 222 classifications. 118 salmon could not be classified as any other salmon in the set. Table 4.35 shows how many specimens were classified with how many features.

	Correctly classified	Incorrectly classified
4 features used	4	8
5 features used	2	3
6 features used	0	0
7 features used	45	45
8 features used	0	3
9 features used	47	12
10 features used	3	12
11 features used	16	9
12 features used	0	3
13 features used	4	2
14 features used	1	0
15 features used	0	0
16 features used	2	1

TABLE 4.35: The result after running the individual recognition algorithm on the salmon, showing how many salmon were classified, either correctly or incorrectly, with how many features.

The method correctly classified 36.47% of the salmon and incorrectly classify 28.8%, leaving 34.7% unclassified. The error-parameters were tested at stricter and more tolerable levels, but it led to the number correct classification being reduced faster than the incorrect classifications(stricter) or incorrect classifications growing at a greater speed than the correct classifications(more tolerable). In other words; trying to get more salmon classified by looser error-thresholds, led to more incorrect classifications.

If a fish is classified with four features, whether it be correct or incorrect, the fish had no spots. Even if those four error-thresholds were stricter, it proves difficult to correctly classify fish with this amount. The number of incorrect classifications is twice as high as the correct classifications.

The amount of features six, eight, 10, 12 and 14 are meant to help the method discriminate against images of different salmon. If the salmon had seven features, it means it had the five first features accepted: length, head-length, nose to fin, fin to tail and spot-difference. Two features are used for each spot, the length and the angle, meaning that if the method used seven features, the salmon had one spot. Nine features-two spots, and so on. If a fish used eight features, it means the length threshold was accepted, but not the angle. The reason there are still classifications are best explained by a short example: image number 2021 was correctly classified with 2022 with five features. The salmon was marked with one complex spot in image 2021, but not in 2022(both images are of the same fish). Then the first five features are used(the distance features all salmon had) and the method searches for the one spot from image 2021 in image 2022. Because all spot values in 2022 are 0(it doesn't have a spot), no more features are used and the *variable_used* is not incremented. It would be pointless to search for a spot that clearly isn't there, and then punish the test image if the fish in that image has no spot. That is how five features were used. It is the same scenario with image correctly or incorrectly classified with six, eight, 10, 12, 14 or 16 features, and those rows are therefore not that interesting. The interesting numbers are those where seven, nine, 11 and 13, because those are cases where the salmon had one or more spots on both or all three images.

Table 4.36 shows the correct and incorrect classifications when using the first five features, and with one, two, three or four spots. There were so few salmon with five spots or more that they are not mentioned.

	Correctly classified	Incorrectly classified	Percent correct and incorrect
No spots(4 features)	4	8	33%/67%
1 spot(7 features)	45	45	50%/50%
2 spot(9 features)	47	12	80%/20%
3 spot(11 features)	16	9	64%/36%
4 spot(13 features)	4	2	67%/33%

TABLE 4.36: It is clear that with more features come more accuracy. From having less correctly classified salmon to either 50/50 or more is a clear improvement.

Table 4.36 shows how much better the accuracy gets once more features are added. It is best at nine features, or 2 spots. The reason one spot gives such a high incorrectly classified number is because the number of salmon with either one or two spots on the head outnumber the number of salmon with three or more. If there is only one spot in difference between two images, the match gets through the spot-difference-threshold. When trying to match one spot with another, the chance of finding a match increases if there are two spots to chose from, instead of just one. Also, if there is at least one spot on the label fish, the error-thresholds for the five previous features are slacked(length, head-length, nose to fin, fin to tail and spot-difference), letting through almost any salmon slightly resembling the label salmon, many of which has two spots.

The score is best with nine features, using two spots. The percentage drops at using 11 features(three spots). A reason for this can be that the head of the salmon is not large, and the space where the spots emerge is even smaller. There is not much room and once the number of spots is above two, the space where the spots can position themselves do not vary much in angle or distance.

4.4.1 Potential Improvements

The slacker parameters when the salmon has one spot is a reason for the misclassification. Too many potential matches are made possible and the chance that other salmon with one or two spots gets a greater score rises. The features of the salmon could be counted on beforehand and the more features available, the stricter the thresholds. Now, the method has the same slack parameters whether the salmon has 7 features(1 spot) or 13 features(4 spots).

More features could also have been added to make the individual recognition even better, like the circumference of the fish, the size of the spots, range to anal fin, size of the eye, location of the eye relative to other locations. Since the features of the fish are not as distinct as wanted, but rather have a resemblance to a Gaussian distribution, it is debatable how much these added features would help, but they could surely better the improvement.

Another reason for some incorrect classifications lie in the method for finding the nose and tail tips. On some occasions, though rarely, the nose point is moved from one image to another from the nose of the fish to its chin, despite being the same salmon. This affects both the distance to the spot and the angle, making it unrecognizable. This method was originally made to find the orientation of the fish, whereas it wasn't crucial to hit the nose point exact. A solution to this could be to implement a pattern method if the salmon has three of more spots, then use the pattern made from the spots for recognition, avoiding the nose point entirely.

Regarding the spots, the spot in itself could be used for individual recognition. Their sizes, convexity, circularity and other parameters could be added to the feature vectors. Then not only the distance and angles to the spots mattered, but the spot's appearance too.

If this method were to be utilized over time, the parameters would have to be accustomed to growth over time. An addition to the distance feature would have to be added, to deal with this uncertain change for each fish. They grow at uncertain rates, lengthening the overall length, circumference, head-length and distance to fins, most likely lowering the success-rate of the method. How much the addition to each feature in the feature vector should be, should be calculated with statistical measures.

Chapter 5

Economy

This is a stand alone chapter which addresses driving factors in the Norwegian and world-wide aquaculture industry. like nourishment demand and GHG emission in section 5.1 Global issues and aquaculture and economy in section 5.2 Potential of realizing IFarm. It is not necessary to read this chapter in order to understand this report, although it is recommended. Assumptions are made, and simulations are conducted in order to determine the actual economical impact individual fish recognition could have for the Norwegian aquaculture industry.

5.1 Global issues and aquaculture

Growing food demand

The population is increasing and with a larger population comes a larger need for food. This is not only caused by the population growth itself, but people are also consuming more as poverty decreases. UN's agriculture production index for the entire world, was 100 in 2005, and then 127 in 2018. This means an increase of 27% over 13 years. Over the same time period, the population grew from 6,542 billion to 7,633 Billion, which is just less than 17%[33]. This indicates larger consuming per individual. The food demand per person increases most in developing countries. IPPC states that the energy/kcal consumption per capita in Asia has increased by 32% from 1970 to 2010,[34](page 822) which has been an area of development in the past decades. Experts estimate the food production has to double within 2050, and aquaculture probably will play an important role[35].

Not only is the demand for food increasing, but also the demand for nutritious food, especially from health conscious consumers is on the rise. This is also a contributor to the growth of aquaculture[36].

GHG emission from agriculture compared to aquaculture

Increasing food demand raises another critical question in context with today's society; What about the environment? It is known that commercial agriculture stands responsible for a considerable amount of the world's emission of Greenhouse Gases**(GHG)**. 24% of the total annual GHG emission comes from agriculture, forestry and other Land Use**(AFOLU)**. IPCC does this grouping because there is a deep connection between them. For example increased agriculture is a key contributor to deforestation, which again causes GHG-emission.

Another number comes from *United States Environmental Protection Agency***(EPA)**. They claim that 9% of total **GHG** emissions in the US in 2016, came from Agriculture. Almost half of this comes directly from enteric fermentation(30%) and manure management(15%), and the rest comes mainly from soil management(50%). These numbers are only for the US, and should not be applied globally.[37] For example does developing countries usually have a much higher percentage of **GHG** emission from AFOLU.

According to MOWI, the carbon footprint is 5.9 for pigs, 30 for bovines and only 2.9 for fish. Fig. 5.1 shows emission development from agriculture. As we can see, cattle meat has been and still is responsible for a significant share of the emission. MOWI suggests that aquaculture can be a solution to this problem and implies that it is wrong that only 2% of the worlds food comes from the ocean, which covers most of the world. Their CEO, Alf-Helge Aarskog, looks to the future and predicts "The Blue Revolution".[38]

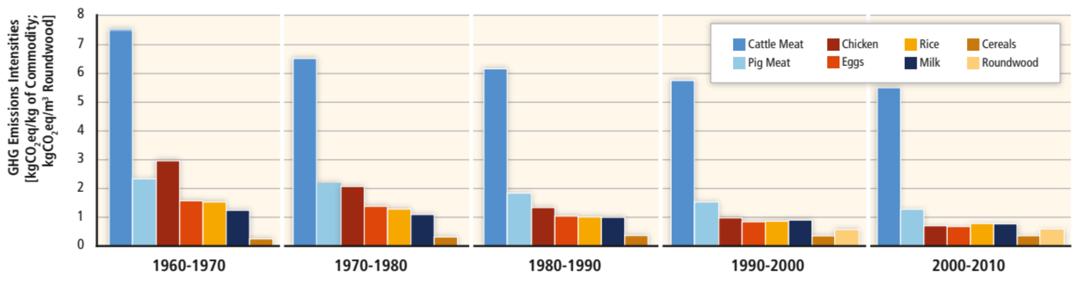


FIGURE 5.1: The figure shows the development of emission from a couple of sources. We can see that the top three contributors are from meat sources, and that cattle meat causes most emission. Especially as we approach today's date further to the right on the graph. The descending trends are probably due to technical development. The figure is fetched from IPCC's "AR5 Climate Change 2014: Mitigation of Climate Change". https://www.ipcc.ch/report/ar5/wg3/

On the other hand, especially considering Norwegian aquaculture, large amounts of emission is caused through export transportation by flight. Exporting salmon from Norway to Japan would triple the total GHG emission of the product. This could be solved by increasing local land-based fish farming, which would pose a threat to production in Norway. Another way is to transport the products in more sustainable ways.[36]

5.2 Aquaculture and the potential of tracking fish individually

Aquaculture can be defined as the farming of seafood and water plants. It is an old practise, but has recently seen fast growth, as high as 8.3% on average every year since the mid 1970's to 2008. It was assumed that 47% of all fish consumed in 2009 came from aquaculture, and this amount is expected to grow.[34] By 2030, the World Bank estimates that this percentage will have increased to 62%.[35]

Compared to traditional wildlife fishing, aquaculture makes it easier to avoid depleting the ocean life-stocks by over-fishing. Recent history has seen several fish-species almost been fished to extinction, but stricter policies have caused significant improvement. Traditional fishing is also a source of pollution, mainly by old fishing nets, which are regarded as one of the worst sources of ocean pollution.[35].

This is avoided when turning to aquaculture. But aquaculture is not immaculate itself. Not all fish species cope good with small spaces, and this can easily lead to the spreading of disease and sea lice in the sea cage[35][1]. Faulty offshore cages combined with stormy seas, may cause infected fish to flee, which may result in them infecting wildlife population.[35] This has received attention in Norway the last couple of years, but the problem seems to have decreased considerably[39].

In the past, Norwegian politicians have proposed regulations in the Norwegian Aquaculture industry. As late as 9th of April 2019, "Miljøpartiet De Grønne", suggested several actions, including tracking escaped fish. This was voted down, because it is expected that such regulations would slow down the industry growth[40].

In Norway, over 20% of the bred salmon in the sea cages died in 2017, and the worst companies report more than 40% casualties. Harald Takle, head researcher of Cermaq, states that using individual health records for each fish will be a revolution in the industry. Geir Stang Hauge, founder of Biosort, estimates that the mortality could be cut by 50-75%.

Machine Vision is seen as a part of a possible solution for both these problems. For example identifying escaped salmon to find who is responsible, and to keep a health journal for every fish in a sea cage to detect early signs of disease and lice spread. This last concept is currently under development, and is called "IFarm". (See 1.1.1 Biosort's IFarm - Tracking fish by machine vision, 4.

Tracking fish by RFID technology

Today, tracking of large fish populations is mainly done by RFID glass-tags, or so-called pit tags. Every pit tag has a unique code. The system is 100% accurate as long as the reader system functions, and the pit tag is not lost. The fish is anaesthetized before tagging, and a significant amount of fish dies during the process. It is also assumed that the process is stressful for the fish, but the impact is unknown. Only in Norway, hundreds of thousands of salmon are tagged every year. The tagging process is time consuming and expensive. Should political demands for tagging be enacted, the industry would have to spend an increased amount of money. Such demands have been proposed several times in the past, but has been shut down for the benefit of industrial growth. Today, Pit tags are mainly used for scientific work on "smaller" populations of fish. It is suggested that recognition by computer vision could replace the use for pit tags and apply tracking on a larger scale. The implementation of this technology could not only help save capital and contribute to more humane treatment of fish, but could accelerate possibilities in the industry even further.

Potential of realizing IFarm

The Norwegian aquaculture industry has grown in recent years. Especially since 2015, both the price of salmon and volume has increased notably.[36]

To get an understanding of how individual tracking could affect the economy of Norwegian salmon breeding, this section presents possible outcomes and important factors. Most numbers are estimated roughly but reasonable, because the outcomes have significant uncertainties, and finding exact values would be superfluous. The IFarm concept is the background for these estimations. This is because the IFarm is the current most relevant concept that plans on using individual recognition on a large scale.

First, the potential income of realizing IFarm is estimated. Secondly, the potential annual costs of realizing IFarm with RFID technology is estimated. At last, a simulation is carried out and the results are discussed.

Potential Income

The goal here is to make an estimate of how much income could be achieved by realizing IFarm. Costs are not considered here. The points below shows the approach for choosing reasonable inputs for the simulations of this chapter:

- The price of salmon was about 40 NOK from 2014-2016. Then it went up to about 60 NOK. Because of continually growing demand, it is expected that the demand will continue to grow, along with the production. Thus, it is expected that the price will stay the same, or rather rise slightly than fall[36]. Thus, simulations will use a triangular distribution, where 50 NOK is the lowest, 60 NOK is the most likely, and 70 NOK is the highest possible sales price.
- It is further assumed that the average weight at accidental death is 4,2kg. This information was extracted from articles[39]. The number is also reasonable because it fits with the assertion that bred salmon are between 3-6kg in the cages. Still, there is uncertainty related to the actual average weight at death, because different time of the season may cause different average weight. Simulations will be performed with a triangular distribution varying from 3.9-4.2-4.5kg.
- It is assumed that a cage holds 200 000 bred salmon. Further it is assumed that currently, 20% of these die on average.[1].
- 50%-75% of fish deaths in cages can be expected to be prevented.[2] These estimates are however not be fully trusted, as they were stated in a interview. Therefore, there will be used a uniform distribution, ranging from as low as 5% up to 75%. The 5% possibility is used to demonstrate what the results may be, in case the scientists have misjudged the situation, or their intentions are biased. This is the most unpredictable input factor associated with the income estimation.

Simulations were made in Excel with the inputs as shown in the bullet points below.

- Price: 40-60-80 NOK/kg
- Average weight at death: 3.9-4.2-4.5kg
- Current average cage death: 40 000 (20% of 200 000)
- Percentage of fish that could be saved: 5%-75%

The income function per cage is:

$$Price * AverageWeight * CurrentDeath * SavePercentage$$

Scenarios and simulations are presented in the following sections.

Result: Income approximations

With the given inputs, worst, most likely and best case scenarios are calculated:

- Worst case scenario: 400 000 NOK / fish cage
- Most Likely scenario: 5 040 000 NOK / fish cage
- Best case scenario: 8 820 000 NOK / fish cage

These are the possible extra income for one single cage containing 200 000 salmon, after implementation of a concept like IFarm.

Potential costs using RFID to tag every fish in a cage

The potential costs of realizing the IFarm concept with an RFID system, starts with tagging all the fish. The cost of this is estimated to get a rough idea how expensive it would be, in order to understand the potential of tracking by machine vision. If mass tagging would actually be initiated, the process would be developed more efficiently. Therefore it is not possible to make an exact estimation of the probable cost using today's prices for tagging. For the estimations, the following assumptions are made, loosely based on established routines:

1. The fish spend 12-18 months in the sea cage. Assuming 200 000 fish in a cage, an average of spending 15 months in the cage, would require implanting of 160 000 pit tags every year.
2. At one of MOWI's hatcheries in 2018, about 75 000 smolts were tagged in one week, by about 10 workers. The work included anaesthetizing, tagging, fin clipping and logistics. Here, it is assumed that 6 workers would be required only for tagging, as three people do the tagging, while the other three does anaesthetizing and logistics.
3. assuming 37.5 work hours per worker for one week. This leads to 10.8 seconds of work per fish tagged, which seems reasonable.

4. The costs of pit tag would probably decrease as production increases, and becomes more efficient. At the same time the demand for pit tags would increase significantly. Today, ordering large (hundreds of thousands) quantities of pit tags results in a price between 5-6 NOK each. It is hard to determine what would happen if the demand increased to 100 million. The question is then how much further it is possible to make the production of pit tags cheaper. 5 NOK is used as the highest and most likely price, which is unlikely to be any higher. 4 NOK is used as a minimum price.
5. More specific costs will not be assumed. Instead this will be called an unknown cost, which will be varied in order to experiment with how much money could be used to develop and run the system.

Here, it will be focused mainly on the cost associated with tagging the fish, as these are seen as the most vital. The factors that are especially important to consider are listed below:

- Pit tag price
- Quantity of fish tagged per work hour
- Salary

Result: implementation by RFID cost approximation

- Worst: -768 400 NOK
- Most likely: 4 038 400 NOK
- Best: 8 228 000 NOK

To get a grasp of understanding of how the outcome result distribution could be, 10 000 simulations were done with the following variations on the input values:

- Prevented Salmon Mortality, %: 5-75% (uniform distribution)
- Price per kg: 50-60-70 NOK. (Triangular distribution with 60 NOK as most likely)
- Average weight at death: 3.9-4.2-4.5 kg. (Triangular distribution with 4.2 kg as most likely)
- Workers needed: 5-6-8. (Triangular distribution with 6 workers as most likely)

- pit tag cost: 5-3 NOK. (Linear distribution from 5 to 3).

Palisades "@RISK" version 7.6 was used to do the 10 000 simulations. The results are presented in fig. 5.2 and 5.3. As can be seen from fig. 5.2, the results are mainly positive.

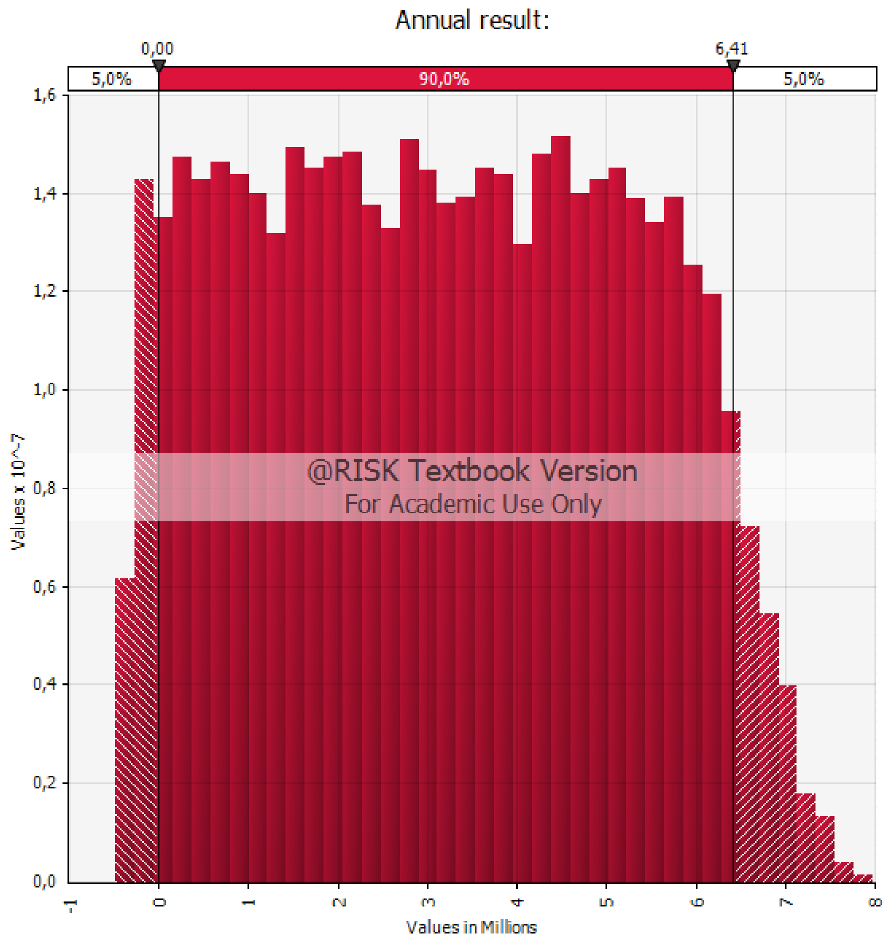


FIGURE 5.2

What is more interesting is the tornado graph in fig. 5.3. This graph displays the impact each input variable has on the output value. It is clear that the fish death prevention percentage is the main variable which determines the income. This basically means that the other input variables are not that important.

To look more closely at this variable, the other input values are held at their most likely value, while the fish death prevention percentage is varied.

Impact of fish death prevention

Only the prevented mortality rate is used as a variable, in order to check out its impact on the ending result.

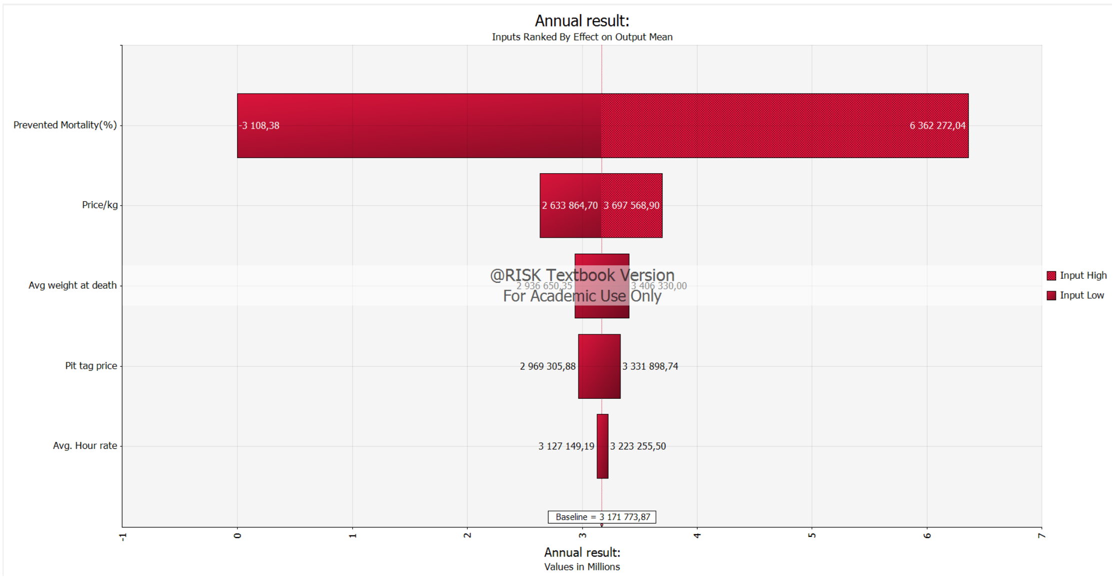


FIGURE 5.3

First, Microsoft excel's solver is used to find the fish death prevention at the breakeven point (0 NOK annual result). The result shows that 9.60% death prevention, would cancel out the estimated costs. A 1% increase, the fish death prevention would result in 0,1M higher incomes. If the death prevention ratio would actually be between 50-75%, as was alleged, the annual result would be between 4,072,000 and 6,592,000 NOK per sea cage with 200 000 salmons and a former death ratio of 20%. This corresponds to a overwhelming gross margin of 421 - 681%.

Discussion and further work

In these estimations, only the possible annual incomes and annual costs of tracking fish by RFID are presented. In reality, if IFarm were to be realized, there would also be initializing costs related to construction and systems. An RFID system is needed to read the pit tags, and a machine vision system to detect sea lice and illnesses. It may seem that an implementation solely by machine vision would have many similar costs as when implementing with RFID technology, except for the annual tagging expenses. Thus, if possible, it is understandable that the IFarm developers are looking to Machine Vision for identification instead of RFID Technology. Anyhow, these systems would need regular service, and there would still be possibilities for errors. To make a decision whether the investment would be worth it or not, the initializing costs and life time estimates needs to be included in a life cycle analysis. Finally, the specific company's desired interest rate will decide if the investment is favourable or not. It would also be up to the companies themselves, to determine factors like wages, number of workers, logistics and so on.

Except for the death prevention ratio in this chapter, the numbers used are somewhat reliable. The uncertainty lies mainly in how prices would be affected by the implementation of such a system. For example, larger orders of pit tags would probably make the price cheaper. It is quite clear that the death prevention potential is both the most decisive and unpredictable factor. This key factor needs be investigated further, before jumping to any conclusions.

Some rough startup expense estimations are given in Appendix C.

Conlusion

There are uncertainties on the estimation of the net economic benefit of the technology, because we do not have exact knowledge of all the income and cost drivers. However, it is clear that there is a large potential to save money, using a concept like IFarm. The decisive factor is the actual death prevention ratio after realisation. This needs to be researched further. If it is as high as 50-75%, prospects seems positive.

Chapter 6

Discussion

In this report discussion has been ongoing through chapters. This chapter therefore contains both summarizing and new discussion.

6.1 Data-sets and image capturing

The data-sets used in this project, were usable for their purpose. Data-set 1 was successfully used to develop working feature extraction methods, while data-set 2 could confirm that some of these methods also worked on an alternative data-set. However, the Nose and Tail Tips detection and Method: Local Gill Detection methods could not be tested against data-set 2, as the back fins were not placed in the image, which was required for both methods. Acquisition of other data-sets could therefore be considered, in order to further test these methods.

Image quality

Even though data-set 1 proved usable, it was evident that the contrasts displayed in the images, were not optimal. This may be one of the reasons that Simple Blob Detection Method was unable to detect a significant proportion of the spots on the gill for this data-set. Even though many of the images in data-set 2 were somewhat blurry, it performed better, because of good contrasts.

In general, it is evident that machine vision methods can not be fairly and completely evaluated by a single data-set, as shown in the example above. It is therefore obvious that further work on the optics and image capturing procedure could be favourable for better results.

In future attempts, it should be considered to increase the lighting on the dark side of the fish(See 3.2 Image Capturing of Atlantic Salmon). A better camera should also be considered.

The state of the used fish

The fish in the data-sets are in the late smoltification stage (See appendix B Biology of Atlantic Salmon). This means that they will likely not have evolved many spots on the gill cover yet(See 1.2 Spots in the skin). Both spot detection methods proved well on distinct spots. Since the spots of a salmon becomes more distinct over time, it is reasonable that the methods could perform even better on more adult salmon. Adult salmon also have more spots, which could be favourable for individual recognition. The fish in the fish cages are are all adult, which would be the case for IFarm (See 1.1.1 Biosort's IFarm - Tracking fish by machine vision). The study [14] by IMR, suggested that the growth of the fish were unfavourable when testing for long term recognition, as they had not necessarily developed enough spots. Future work should include testing the methods on older fish.

6.2 Feature extraction methods

Contour extraction method

The Extraction of the fish in the image method fits its purpose when the fish is immediately surrounded by a blue background. Since this was one of the demands when data-set 1 was cleaned (See appendix A), it performed well on all of the remaining images. It was important to be able to extract the fish accurately to get an accurate contour of the fish. Since this contour indirectly affected the results of the other methods, it would be unfair to the other methods if their results was heavily influenced by the extraction method. Thus, for future work, it will also be important to develop extraction methods that are accurate. It would probably require a more complex method if the images were acquired from a live camera in a river, due to varying background and illumination.

Nose and back fin tips detection

The Nose and Tail Tips detection method partly fulfilled its purpose, which was to determine the orientation of the fish and find the nose point. Sometimes, the method misinterprets the cheek as the nose, which is particularly unfavourable for later recognition stages that uses the nose point. It was however not of importance to the ROI extraction of the head, and the length estimation. The nose point could be better extracted by considering the intensity values in the area, which is normally darker on the nose.

The length estimation is inaccurate when the fish is bent, as it is measured from the nose point, to the point between the back fin tips. A method that follows the through the center of the fish to measure it, should be developed for accurate measures.

Gill cover end detection

The Method: Local Gill Detection proves usable in extracting a ROI around the head of the fish, that is later used for spot detection on the gill cover. It is however a bit imprecise, with a standard variance of 5.50 pixels / 1.32 mm. Better preciseness here, could result in better individual recognition possibilities. More experiments could be conducted using different parameters, to get more optimal results.

Pectoral fin detection

It was wise to use otsu's method to find a threshold for segmenting the pectoral fin. The method did its task well on both data-sets. How well it would do in other environments is debatable. If a camera was submerged into a river with varying illumination, otsu's method would probably not do as well as it did for these data-sets(2.2.1 Otsu's Threshold page 18) and the method would have to be modified for this.

Gill Cover spot detection

The two methods used for detecting the spots on the gill cover are promising, but not optimal, as many False detections ocurred when it was attempted to detect all the spots. The relatively best attempt was perhaps the one that resulted in 91.18% of spots detected, with only 20.62% percent of total detections being false. It is evident that the methods should be developed further and/or be implemented in a more thorough way; it should be conducted experiments using several input combinations and/or multi-scale approaches, as recommended by T. Lindeberg in [27] (See 2.4 Detection of spot-like structures). Especially regarding the Dark Spot Detector, the use of multiple variations of Gaussian kernels, could also help detect various spot shapes and sizes.

As for blob detection, there are several alternative approaches to the ones used in this project, as mentioned in 2.4 Detection of spot-like structures on page 26. These should also be implemented for gill spot detection, and eventually combined with the methods developed here.

6.3 Individual recognition

The method for individual recognition correctly classified 36.47%of the salmon. This include whether it recognized the same specimen for both images, but it recognized roughly 65 out of 178 salmon. If this method was instead tried on a fish tank with more than 200 000 individuals, the correct-classification rate would surely drop even further. The method would have to be stricter and scale the thresholds with the amount of features used.

Whether it comes down to the quality of the images or the robustness of the methods is hard to determine, and there are room for improvements in both fields. The methods' possible improvements are discussed earlier and would lead to better results, especially when combined with better imagery.

Chapter 7

Conclusion

The methods for extracting the fish, locating the nose, tail tips, pectoral fin and gills work well on images of salmon as long as the input image displays the fish, immediately surrounded by a blue background. Under different circumstances the methods might need to be adapted to achieve the wanted result. For these data-sets, the methods performed acceptably. The detection of the nose point should should however be improved, as the cheek sometimes is detected instead.

The Dark Spot Detection method is able to detect almost every spot in both data-sets, although it causes many excess detections. The Simple Blob detector performed best on data-set 2, while struggling to detect spots in data-set 1. The methods should be implemented with several parameters and multi-scale approach for improved results.

Regarding individual recognition, the method does not perform well enough to be a suitable alternative to recognize individual salmon compared to using pit tags, with only recognizing 65 of 178 salmon(36.47% success rate). The uncertainty is too great. With more features, better optics and a good way of solving a salmon's growth, it could perhaps become a suitable solution to track an individual fish over a period of time. For now, the individual recognition method does not serve well as a standalone method for recognizing individual salmon.

Appendix A

Dataset Information

General:

- starts at fish no. 1,
- ends at fish no. 203.
- images of fish no. '*mnk*' is labelled as *mnk*1, *mnk*2 and 4 *mnk*3. As an example, the 3 images of fish no. 14 is labelled as 0141, 0142 and 0143.

Fish that have other than 3 no. of images:

- 12. only has 2 images
- 85. has 4 images
- 93. has 4 images
- 101. has 0 images

A complete dataset with handpicked images were gathered from the main dataset(from MOWI). This filtered dataset was then the set of images used for salmon recognition. Images with either of the unwanted cases below were removed from the selection. The images removed because it fulfilled one of the criteria are noted with their filename:

1. The fish in not immediately surrounded by the blue background, in other words: a part of the fish is touching another object that is not blue.
    - 0073
    - 0122

- 0131
- 0132
- 0133
- 0151
- 0152
- 0153
- 0132
- 0133
- 0453
- 0632
- 0633
- 0673

2. The fish is oriented more vertically, than horizontal

   - 0011
   - 0012
   - 0013
   - 0021
   - 0702
   - 0703
   - 0792
   - 0793
   - 0992
   - 0993
   - 1181
   - 1182
   - 1183

3. The fish is or has blurry regions

   - 0712
   - 0821
   - 0953
   - 1211

- 1253
- 0532
- 0832

4. The fish is placed far from the image center.

- 0701
- 0713
- 0853
- 0852
- 0842
- 0843
- 0851
- 0854
- 0893
- 0923
- 0991
- 1002
- 1003
- 1021
- 1022
- 1023
- 1051
- 1052
- 1053
- 1081
- 1082
- 1083
- 1092
- 1093
- 1122
- 1123
- 1131
- 1132

- 1133
- 1151
- 1152
- 1173
- 1191
- 1192
- 1193
- 1201

The dataset contains 534 images of acceptable fish that was used.

Appendix B

Biology of Atlantic Salmon

There are several types of salmon. The type that lives in Norway and the Atlantic Ocean, is called Atlantic Salmon, or in latin, Salmo Salar. The males reach a maximum size of around 1.50 meters and 40kg, and they typically live from 2-8 years. Wild Atlantic salmons are born in the rivers connected to the Atlantic Ocean, all the way from Spain to Russia, to Canada, to southern US. They spawn in the rivers, and do not turn to the ocean before they become adults.[41]

The appearance of Atlantic Salmon changes several times through its lifeline, and is affected by both genes and environment[14][13]. Considering use of Machine Vision, it is important to understand how the appearance of Atlantic Salmon may vary. Therefore, this chapter includes a brief presentation of:

1. The life cycle of Atlantic Salmon
2. The impact of genes and environment on the appearance of Atlantic Salmon.

B.0.1 Life Cycle of Atlantic Salmon

The salmon has a unique life cycle, which phases are roughly described by fig. B.1. These phases are explained in more detail in the following sections.

Hatching - Alevin phase

Atlantic Salmon eggs spawn between October and January. During spawning, the female chooses a fitting area with good current and coarse gravel to cover the eggs. The eggs lie in the river gravel until they hatch in the spring. The first 5-6 weeks of their lives,

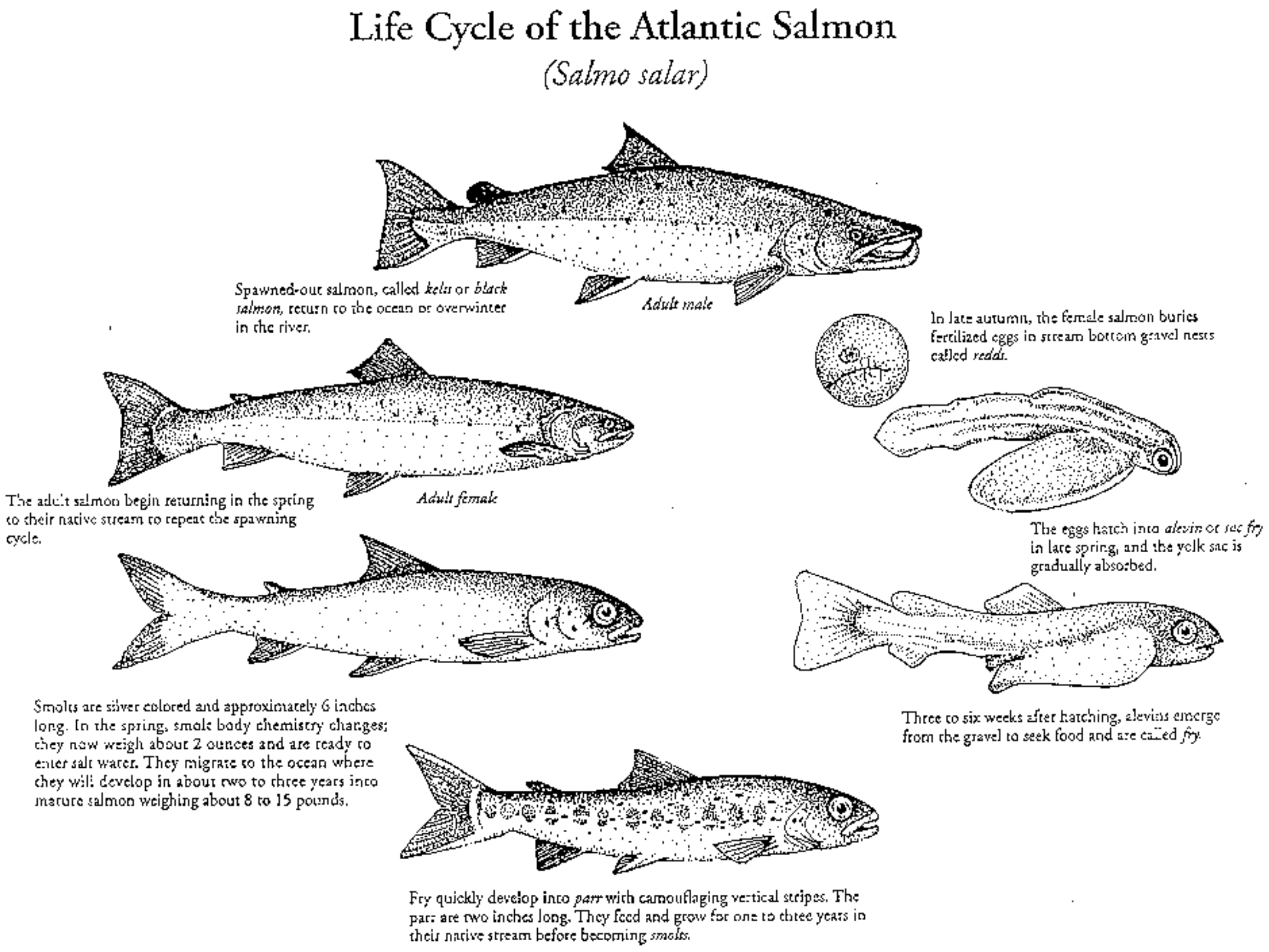

Life Cycle of the Atlantic Salmon
(Salmo salar)
Spawned-out salmon, called kelts or black salmon, return to the ocean or overwinter in the river.
Adult male
In late autumn, the female salmon buries fertilized eggs in stream bottom gravel nests called redds.
The adult salmon begin returning in the spring to their native stream to repeat the spawning cycle.
Adult female
The eggs hatch into alevin or sac fry in late spring, and the yolk sac is gradually absorbed.
Smolts are silver colored and approximately 6 inches long. In the spring, smolt body chemistry changes; they now weigh about 2 ounces and are ready to enter salt water. They migrate to the ocean where they will develop in about two to three years into mature salmon weighing about 8 to 15 pounds.
Three to six weeks after hatching, alevins emerge from the gravel to seek food and are called fry.
Fry quickly develop into parr with camouflaging vertical stripes. The parr are two inches long. They feed and grow for one to three years in their native stream before becoming smolts.

FIGURE B.1: The Life Cycle of an Atlantic Salmon. Image from: Wikimedia Commons.

the salmon is called an alevin. This can be describes as a larvae, or baby fish. In this period, it gets its nourishment from a yolc sac attached to it.[41] A newly hatched Alevin is shown in fig. B.2.

Fry phase

In the next phase, they are called Fry. As Fry, they emerge from the gravel and are now able to navigate around in the river(with their eight fins). They now feed on microscopic animals. Because most Fry are eaten by predators, they gradually develop camouflage. Fig. B.3 shows a couple of frys.

Parr phase

After further evolving, it is then called a Parr. Camouflage like stripes and spots are developed, like shown in fig. B.4. They now feed on insects. As a Parr, they will live

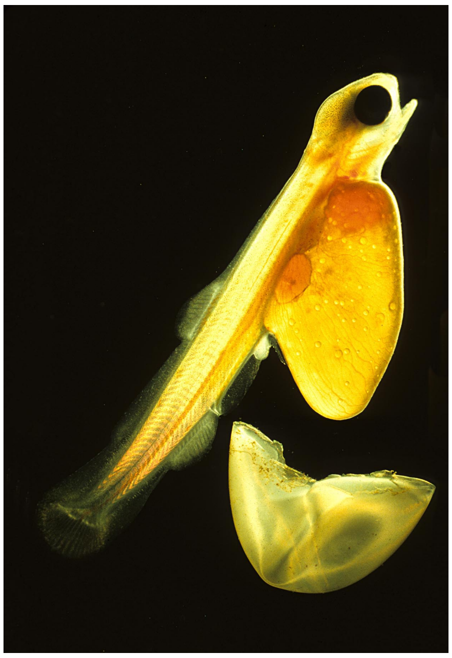


FIGURE B.2: A newly hatched Salmon Alevin. Image from: Wikimedia Commons.

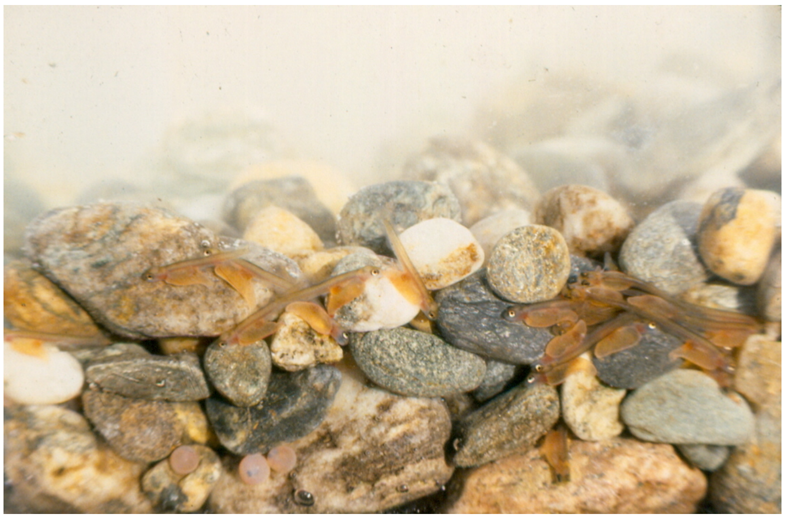


FIGURE B.3: Several Atlantic Salmon Fry, that have emerged from the gravel. Image from: Wikimedia Commons.

in freshwater rivers for at least one year, but this time period varies with circumstances like temperature and feeding quality. For example, the Parr in the cold glacier rivers of Sogn og Fjordane in Norway, can stay in the rivers for up to five years.[41]

Smoltification

The next phase happens when the salmon is preparing to enter seawater. This is called the smoltification process and involves a change of appearance from brown/green to more silvery looking. The process usually starts when the Parr is around 11-12 cm [4],

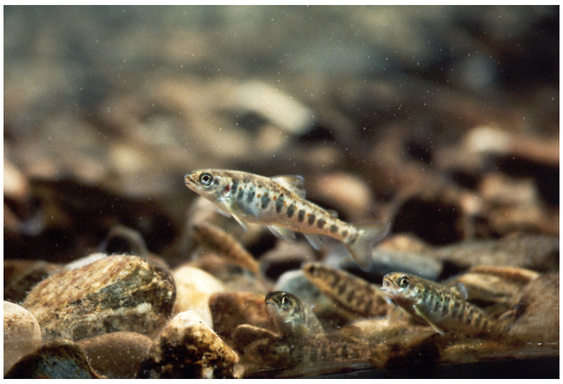


FIGURE B.4: Atlantic Salmon Parr. Image from: Wikimedia Commons.

and the Salmon could grow to as large as 25cm during the smoltification[5]. In the breeding industry, it usually takes 8-18 months to breed farmed smolts at about 100g, whom are ready to enter seawater cages. When the smoltification is done, the Salmon is called Smolt?[41] Fig. B.5 shows an example of a smolt, ready to enter seawater.

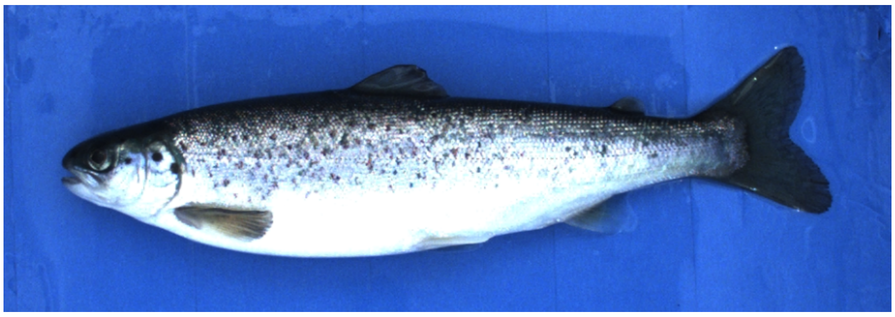


FIGURE B.5: A breed Atlantic Salmon smolt, ready for seawater. Image: Specifically for this work.

Adult

After entering sea water, the Atlantic Salmon has an enormous growth. It becomes mature and ready to mate earliest one year after entering seawater. It will eventually return to the freshwater rivers to spawn. Some return after one year, as ”small salmon” at around 1.5-3kg. Others stay in the ocean for longer time, and are then medium or large-sized when they return to the river.[41]

The spawning migration takes place during late spring, to the autumn. The spawning itself happens late in the autumn. In Norway, most salmon spawns the same year they have travelled up the rivers, but in some other places, parts of the salmons does the migration the year before. The salmon mostly turn back to the same rivers as they spawned themselves, and even to the same part of the river. This Phenomenon is known as ”homing”, which is still not understood, although there are recent research in the field, involving theories that the fish interprets complex signals from various stimuli on its travels, to later use these to find the way back home.[41]

Many of the adults become exhausted after the spawning, and die shortly after. But some of them actually return to the ocean and continues the process a couple of times. This is unique for the Atlantic Salmon. None of the other salmon species does this.

In breeding it usually takes 12-18 months of adult life for the fish to be ready for slaughter at about 3-6kg. It is also desirable that the fish has not reached maturity, as this has some negative effects both on quality and economy [6]. Fig. B.6 shows an adult breed Atlantic Salmon.

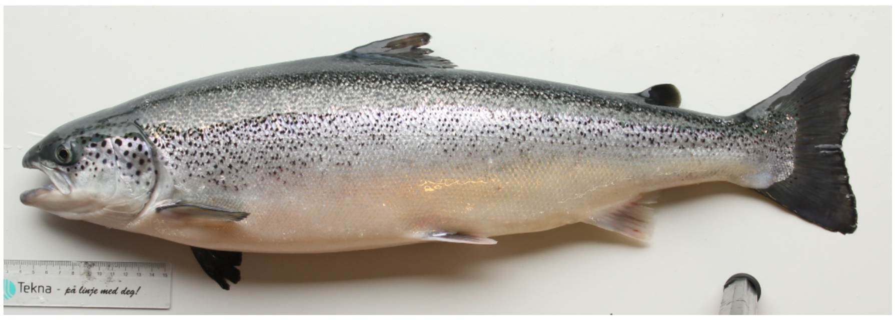


FIGURE B.6: An adult breed Atlantic Salmon, ready for slaugther. Image: Obtained from IMR.

Appendix C

Rough IFarm initialization costs with RFID Technology

W/RFID Total: 590,000 NOK Annual costs: 1M NOK (pit tags, work hours, service)

- 3 cameras a 10,000 NOK
- PC/Server system a 100,000 NOK
- Construction costs a 200,000 NOK
- RFID reader system w/ 2 antennas a 80,000 NOK
- 3 semi-automatic pit tag machines a 60,000 NOK

W/Machine Vision Total: 350,000 NOK Annual Costs: 30 000 NOK (only service needed)

- 3 cameras a 10,000 NOK
- PC/Server system a 100,000 NOK
- Construction costs a 200,000 NOK

camera: https://www.ptgrey.com/blackfly-50-mp-color-usb3-vision-sharp-rj32s4aa0dt

5MP camera from pt-grey: 10 000 NOK x3 Fast computer: 50 000. Total of: 80 000 NOK

Since one probably could use a super computer for several cages, it is hard to estimate what the exact cost would be here. This is a rough estimate.

We will not go into detail here, about the potential cost of rebuilding the cages to fit these specific applications. For simplicity, we assume the costs would be similar.

Appendix D

Code

This part contains all relevant code made for this project.

D.1 Extract Fish in image

```
def find_largest_foreground_with_kmeans_noCrop(self, image, filename, background='arbitrary'):
    print('attempting to extract only the fish in the image')
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt
    from sklearn.cluster import KMeans
    ## Using K-means clustering to get "exact" position of the fish in the image.
    ## Requires:
    # Distinct and monotone background color. Different from foreground object.
    # Relatively big object.
    # The wanted object HAS to be the largest in the scene

    # params:
    # image: input image to be clustered.

    image = image.astype('uint8')

    ## CONVERT TO LAB COLOR SPACE
    print('converting image from BGR to LAB color space')
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    LAB = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)

    ## CREATE INPPUT ND-ARRAY FROM COLOR VECTORS A&B.
    print('creating input array for the k-means algorithm')
    A = LAB[:, :, 1]
    B = LAB[:, :, 2]
    A_array = A.ravel()
    B_array = B.ravel()
    X = np.zeros([len(A_array), 2])
```

```
X[:, 0] = A_array
X[:, 1] = B_array

##CREATE INITIALIZATION VECTOR, SINCE WE ROUGHLY KNOW THE COLORS (based on the images from 2
init_ndarray = np.array([[128, 116], [150, 69]])

## RUN K-MEANS ALGORITHM AND PREDICT CLUSTER CENTERS
print('running k-means algorithm')
if background == 'blue':
    kmeans = KMeans(n_clusters=2, init=init_ndarray, n_init=1, max_iter=5, random_state=0).f

else:
    kmeans = KMeans(n_clusters=2, init='k-means++', n_init=1, max_iter=30, random_state=0).f

clustered_array = kmeans.predict(X)

## RESHAPE OUTPUT ARRAY TO ND-ARRAY (MATRIX)
print('reshaping output array to matrix')
rows = image.shape[0]
cols = image.shape[1]
clustered_img = np.reshape(clustered_array, [rows, cols])

## INVERT BINARY IMAGE IF MOST AREA IS WHITE. ASSUMING FISH DOES NOT COVER MORE THAN HALF OF
M, N = clustered_img.shape
img_size = M * N
nWhitePixels = np.sum(clustered_img)
if nWhitePixels >= 0.5 * img_size:
    clustered_img = (clustered_img - 1)
    clustered_img = np.multiply(clustered_img, -1)

clustered_img = np.uint8(clustered_img)

##LABEL THE CLUSTERED BW IMAGE
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(image=clustered_img, connec

## FIND INDEX/LABEL OF LARGEST OBJECT (0th object is the background)
largest_object_label = stats[1:, 4].argmax() + 1

## IF HEIGHT OF OBJECT IS LARGER THAN 900, ASSUME THAT THE SECOND LARGEST OBJECT IS THE FISH
h = stats[largest_object_label, cv2.CC_STAT_HEIGHT]
if h > 900:
    ## FIND SIZE OF SECOND LARGEST OBJECT
    stats2 = stats.copy()
    stats2[largest_object_label, 4] = 0
    second_largest_object_label = stats2[1:, 4].argmax() + 1
    if stats2[second_largest_object_label, cv2.CC_STAT_AREA] > 10000:
        largest_object_label = second_largest_object_label

## CREATE BOUNDING BOX AROUND LARGEST OBJECT
## MAKE SLIGHTLY LARGER
#t = stats[largest_object_label, cv2.CC_STAT_TOP] - 3
#l = stats[largest_object_label, cv2.CC_STAT_LEFT] - 3
#h = stats[largest_object_label, cv2.CC_STAT_HEIGHT] + 3
#w = stats[largest_object_label, cv2.CC_STAT_WIDTH] + 3
```

```
    #cropped_largest = labels[t:t + h, l:l + w]

    ## REMOVE ALL PIXELS THAT ARE NOT LABELLED AS THE LARGEST OBJECT
    onlyLargest = labels.copy()
    onlyLargest[onlyLargest != largest_object_label] = 0

    onlyLargest = np.uint8(onlyLargest / onlyLargest.max())
    onlyLargest, contours, hierarchy = cv2.findContours(onlyLargest, cv2.RETR_CCOMP,
                                                         cv2.CHAIN_APPROX_SIMPLE)

    ## WE WANT ONLY THE EXTERNAL CONTOUR OF THE LARGEST OBJECT
    # Draw External Contours

    # Set up empty array
    external_contours = np.zeros(onlyLargest.shape)

    # For every entry in contours
    for i in range(len(contours)):

        # last column in the array is -1 if an external contour (no contours inside of it)
        if hierarchy[0][i][3] == -1:
            # We can now draw the external contours from the list of contours
            cv2.drawContours(onlyLargest, contours, i, 255, -1)

    onlyLargest = np.uint8(onlyLargest / onlyLargest.max())

    ## CREATE THE OUTPUT IMAGE, CONTAINGING ONLY THE LARGEST OBJECT / FISH
    ## THE OUTPUT IMAGE IS MADE SLIGHTLY LARGER, BECAUSE METHODS LIKE EDGE DETECTION ARE GOING T
    output_img = image.copy()
    output_img[:, :, 0] = np.multiply(output_img[:, :, 0], onlyLargest)
    output_img[:, :, 1] = np.multiply(output_img[:, :, 1], onlyLargest)
    output_img[:, :, 2] = np.multiply(output_img[:, :, 2], onlyLargest)

    writingfilename_onlyLargest = 'D:/Master/AlleBilder/fraPycharm/filt/onlyContour/' + str(file

    ## WE ALSO WANT IMAGE WITH ONLY CONTOUR TO FIND SHAPE FEATURES
    onlyContour = onlyLargest*255
    cv2.imwrite(writingfilename_onlyLargest,onlyContour)


    ##MAKE BACKGROUND WHITE
    output_img[output_img[:, :, 0] == 0] = 255
    output_img[output_img[:, :, 1] == 1] = 255
    output_img[output_img[:, :, 2] == 2] = 255

    ## WRITE IMAGE
    writingfilename = 'D:/Master/AlleBilder/fraPycharm/filt/onlyFish/' + str(filename)

    #print('writing image as: ' + str(writingfilename))
    #cv2.imwrite(filename=writingfilename, img=output_img)
    #print('finished written to file')

    return output_img, onlyContour

def find_center_foreground_with_kmeans_noCrop_IMR(self, image, filename, background='arbitrary')
```

```
print('attempting to extract only the fish in the image')
import cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
## Using K-means clustering to get "exact" position of the fish in the image.
## Requires:
# Distinct and monotone background color. Different from foreground object.
# Relatively big object.
# The wanted object HAS to be the largest in the scene

# params:
# image: input image to be clustered.

image = image.astype('uint8')

## CONVERT TO LAB COLOR SPACE
print('converting image from BGR to LAB color space')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
LAB = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)

## CREATE INPPUT ND-ARRAY FROM COLOR VECTORS A&B.
print('creating input array for the k-means algorithm')
A = LAB[:, :, 1]
B = LAB[:, :, 2]
A_array = A.ravel()
B_array = B.ravel()
X = np.zeros([len(A_array), 2])
X[:, 0] = A_array
X[:, 1] = B_array

##CREATE INITIALIZATION VECTOR, SINCE WE ROUGHLY KNOW THE COLORS (based on the images from 2
init_ndarray = np.array([[128, 116], [150, 69]])

## RUN K-MEANS ALGORITHM AND PREDICT CLUSTER CENTERS
print('running k-means algorithm')
if background == 'blue':
    kmeans = KMeans(n_clusters=2, init=init_ndarray, n_init=1, max_iter=5, random_state=0).f

else:
    kmeans = KMeans(n_clusters=2, init='k-means++', n_init=1, max_iter=30, random_state=0).f

clustered_array = kmeans.predict(X)

## RESHAPE OUTPUT ARRAY TO ND-ARRAY (MATRIX)
print('reshaping output array to matrix')
rows = image.shape[0]
cols = image.shape[1]
clustered_img = np.reshape(clustered_array, [rows, cols])

## INVERT BINARY IMAGE IF MOST AREA IS WHITE. ASSUMING FISH DOES NOT COVER MORE THAN HALF OF
M, N = clustered_img.shape
img_size = M * N
nWhitePixels = np.sum(clustered_img)
#if nWhitePixels >= 0.5 * img_size:
```

```
centerPixel = int(clustered_img[int(M/2),int(N/2)])
print(clustered_img[int(M-1),int(N-1)])
if (centerPixel == 0):
    clustered_img = (clustered_img - 1)
    clustered_img = np.multiply(clustered_img, -1)

clustered_img = np.uint8(clustered_img)

##LABEL THE CLUSTERED BW IMAGE
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(image=clustered_img, connec

## FIND INDEX/LABEL OF LARGEST OBJECT (0th object is the background)
largest_object_label = stats[1:, 4].argmax() + 1
## Or center:
largest_object_label = int(labels[int(M/2),int(N/2)])

## IF HEIGHT OF OBJECT IS LARGER THAN 900, ASSUME THAT THE SECOND LARGEST OBJECT IS THE FISH
h = stats[largest_object_label, cv2.CC_STAT_HEIGHT]
if h > N-1:
    print('jdfuhiushfuihsduifhuihsduihfuihsduihfuihsduihfuihsduihfihsduihfuisdhufhsuidhfiuhs
    ## FIND SIZE OF SECOND LARGEST OBJECT
    stats2 = stats.copy()
    stats2[largest_object_label, 4] = 0
    second_largest_object_label = stats2[1:, 4].argmax() + 1
    if stats2[second_largest_object_label, cv2.CC_STAT_AREA] > 100000:
        largest_object_label = second_largest_object_label

## CREATE BOUNDING BOX AROUND LARGEST OBJECT
## MAKE SLIGHTLY LARGER
#t = stats[largest_object_label, cv2.CC_STAT_TOP] - 3
#l = stats[largest_object_label, cv2.CC_STAT_LEFT] - 3
#h = stats[largest_object_label, cv2.CC_STAT_HEIGHT] + 3
#w = stats[largest_object_label, cv2.CC_STAT_WIDTH] + 3

#cropped_largest = labels[t:t + h, l:l + w]

## REMOVE ALL PIXELS THAT ARE NOT LABELLED AS THE LARGEST OBJECT
onlyLargest = labels.copy()
onlyLargest[onlyLargest != largest_object_label] = 0

onlyLargest = np.uint8(onlyLargest / onlyLargest.max())
onlyLargest, contours, hierarchy = cv2.findContours(onlyLargest, cv2.RETR_CCOMP,
                                                    cv2.CHAIN_APPROX_SIMPLE)

## WE WANT ONLY THE EXTERNAL CONTOUR OF THE LARGEST OBJECT
# Draw External Contours

# Set up empty array
external_contours = np.zeros(onlyLargest.shape)

# For every entry in contours
for i in range(len(contours)):

    # last column in the array is -1 if an external contour (no contours inside of it)
    if hierarchy[0][i][3] == -1:
        # We can now draw the external contours from the list of contours
        cv2.drawContours(onlyLargest, contours, i, 255, -1)

onlyLargest = np.uint8(onlyLargest / onlyLargest.max())

## CREATE THE OUTPUT IMAGE, CONTAINGING ONLY THE LARGEST OBJECT / FISH
## THE OUTPUT IMAGE IS MADE SLIGHTLY LARGER, BECAUSE METHODS LIKE EDGE DETECTION ARE GOING T
output_img = image.copy()
output_img[:, :, 0] = np.multiply(output_img[:, :, 0], onlyLargest)
output_img[:, :, 1] = np.multiply(output_img[:, :, 1], onlyLargest)
output_img[:, :, 2] = np.multiply(output_img[:, :, 2], onlyLargest)

writingfilename_onlyLargest = 'D:/Master/AlleBilder/fraPycharm/prikkFisk/onlyContour2/' + st

## WE ALSO WANT IMAGE WITH ONLY CONTOUR TO FIND SHAPE FEATURES
onlyContour = onlyLargest*255
cv2.imwrite(writingfilename_onlyLargest,onlyContour)


##MAKE BACKGROUND WHITE
output_img[output_img[:, :, 0] == 0] = 255
output_img[output_img[:, :, 1] == 1] = 255
output_img[output_img[:, :, 2] == 2] = 255

## WRITE IMAGE
writingfilename = 'D:/Master/AlleBilder/fraPycharm/prikkFisk/onlyFish2/' + str(filename)

#cv2.circle(output_img,(int(N / 2), int(M / 2)),21,(0,0,255),-1)

#print('writing image as: ' + str(writingfilename))
cv2.imwrite(filename=writingfilename, img=output_img)
#print('finished written to file')

return output_img, onlyContour
```

Listing D.1: Python example

D.2 Code: Find nose and Tail Tips

```
def find_nose_and_backfintips(self, cnt_image, image, filename):
    ## Import library stuff
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt
    from PIL import Image

    img = cnt_image
    img = np.uint8(img)
    binary = np.uint8(img / img.max())

    print('teat')
    print(binary.shape)

    ## GET CONTOUR OF FISH
    binary, contours, hierarchy = cv2.findContours(binary, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMP
    cnt = contours[0]

    ## GET MOMENTS OF CONTOUR AND FIND CENTROID
    M = cv2.moments(cnt)
    cx = int(M['m10'] / M['m00'])
    cy = int(M['m01'] / M['m00'])
    centroid = [cx, cy]

    ##CREATE OUTLINE TO FIND CONTOUR POINTS
    black_img = np.zeros(img.shape)
    cv2.drawContours(black_img, [cnt], 0, 1)

    cntPoints = cnt.copy()

    ## SINCE THERE IS ONLY 1 CONTOUR IN INPUT IMAGE
    cntPoints[cnt == 1]
    len(cntPoints)

    cntPoints = cnt.copy()
    cntPoints[cnt == 1]

    ## FINDING ALL DISTANCES FROM THE CENTROID OF THE FISH
    dists_from_centroid = []
    i = 0
    for points in cntPoints:
        dist_vector = points - centroid
        dists_from_centroid.append(dist_vector)

    distances = np.round(np.sqrt(np.sum(np.square(dists_from_centroid), -1)), 8)
    distances = np.asarray(distances)

    ## SHIFT THE ARRAYS, SO THAT A POTENTIAL MAXIMA IS NOT LOST IN THE ENDPOINTS
    shift_amount = np.where(distances == distances.min())[0]
    distances = np.roll(distances, -shift_amount)
    ## x2 BECAUASE IS DOUBLE THE LENGTH
    cntPoints = np.roll(cntPoints, -(shift_amount * 2))

    ##FIND MEAN DISTANCES BECAUSE OF S T Y
    meanDistances = []
    for i in range(0, len(distances) - 3):
        meanDist = (distances[i - 3] + distances[i - 2] + distances[i - 1] + distances[i] + dist
                    distances[i + 2] + distances[i + 3]) / 7
        meanDistances.append(meanDist)

    meanDistances = np.asarray(meanDistances)

    ##FIND LOCAL MAXIMA:
    local_maxima_idx = []
    for i in range(0, len(meanDistances) - 1):
        ##LOCAL MAXIMA IF:
        if (meanDistances[i] > meanDistances[i - 1]) & (meanDistances[i] > meanDistances[i + 1])
```

```
            local_maxima_idx.append(i)

    local_maximas = meanDistances[local_maxima_idx]
    # local_maximas.sort()

    local_maximas3 = []
    local_maximas3_idx = []
    for idx1 in local_maxima_idx:
        for idx2 in local_maxima_idx:

            ## INDEXES THAT ARE TOO CLOSE, ARE ALSO CLOSE ON THE ACTUAL FISH. ONLY THE LARGEST O
            ## CONSIDERED
            if (np.abs(idx1 - idx2) < 30) & (idx1 != idx2):
                print(idx1 - idx2)
                ## KEEP ONLY THE LARGES MAXIMA
                if meanDistances[idx1] >= meanDistances[idx2]:
                    local_maximas[local_maximas == meanDistances[idx2]] = 0

    local_maximas = local_maximas[local_maximas != 0]

    ## KEEP ONLY TOP 3 LOCAL MAXIMAS. THIS SHOULD BE THE NOSE AND THE TWO TIPS ON THE BACK FIN
    for i in range(0, 3):
        local_maximas3.append(local_maximas.max())
        local_maximas[np.argmax(local_maximas)] = 0
        idx = np.where(meanDistances == local_maximas3[i])[0]
        local_maximas3_idx.append(idx)

    # cntPoints[local_maximas3_idx[:,0]]
    local_maximas3_idx = np.asarray(local_maximas3_idx)
    local_maximas3_idx = local_maximas3_idx.ravel()

    print(local_maximas3_idx)
    keyPoints = cntPoints[local_maximas3_idx]

    RGB = image

    ## THE FOLLOWING IS USED TO LABEL THE POINTS
    gray = cv2.cvtColor(RGB, cv2.COLOR_RGB2GRAY)
    ret, binary = cv2.threshold(gray, 100, 255, cv2.THRESH_BINARY_INV)
    binary = np.asarray(binary)
    binary, contours, hierarchy = cv2.findContours(binary, cv2.RETR_CCOMP, cv2.CHAIN_APPROX_SIMP
    ## FIND LARGEST CONTOUR
    c = max(contours, key=cv2.contourArea)
    ## GET MOMENTS OF CONTOUR AND FIND CENTROID
    c_M = cv2.moments(c)
    c_cx = int(c_M['m10'] / c_M['m00'])
    c_cy = int(c_M['m01'] / c_M['m00'])
    c_centroid = [c_cx, c_cy]

    ##Find the distances from the centroid of the dark parts of the fish. This should be closer
    ## than the bottom

    ## FINDING POINT ON THE OUTLINE

    dists_from_c_centroid = []
```

```
for points in cntPoints:
    dist_c_vector = points - c_centroid
    dists_from_c_centroid.append(dist_c_vector)

c_distances = np.round(np.sqrt(np.sum(np.square(dists_from_c_centroid), -1)), 8)
c_distances = np.asarray(c_distances)

outLinePoint = cntPoints[c_distances.argmin()]
cv2.circle(RGB, (outLinePoint[0][0], outLinePoint[0][1]), 10, (255, 0, 255), -1)

diff = np.zeros(3)
for i in range(0, 3):
    a = np.abs(keyPoints[i] - outLinePoint)
    a = np.square(a)
    a = np.sqrt(np.sum(a))
    diff[i] = a

for i in range(0, 3):
    if diff[i] == min(diff):
        topBackfinPoint = keyPoints[i]
    elif diff[i] == max(diff):
        nosePoint = keyPoints[i]
    else:
        botBackfinPoint = keyPoints[i]

cv2.circle(RGB, (botBackfinPoint[0][0], botBackfinPoint[0][1]), 10, (0, 0, 255), -1)
cv2.putText(RGB, 'Bot', (botBackfinPoint[0][0], botBackfinPoint[0][1]), cv2.FONT_HERSHEY_SIM
            (0, 0, 255), 2, cv2.LINE_AA)

cv2.circle(RGB, (topBackfinPoint[0][0], topBackfinPoint[0][1]), 10, (0, 255, 0), -1)
cv2.putText(RGB, 'Top', (topBackfinPoint[0][0], topBackfinPoint[0][1]), cv2.FONT_HERSHEY_SIM
            (0, 255, 0), 2, cv2.LINE_AA)

cv2.circle(RGB, (nosePoint[0][0], nosePoint[0][1]), 10, (255, 0, 0), -1)
cv2.putText(RGB, 'Nose', (nosePoint[0][0] - 50, nosePoint[0][1] - 20), cv2.FONT_HERSHEY_SIMP
            2, cv2.LINE_AA)

cv2.circle(RGB, (c_centroid[0], c_centroid[1]), 10, (255, 255, 0), -1)

cv2.circle(RGB, (centroid[0], centroid[1]), 10, (0, 255, 255), -1)

cv2.drawContours(RGB, [keyPoints], 0, 255, 2)

#writingfilename = 'D:/Master/AlleBilder/fraPycharm/filt/noseAndBackFin/v1_20190329/' + str(
#print(writingfilename)

RGB = cv2.cvtColor(RGB, cv2.COLOR_RGB2BGR)
output_w_drawing = RGB

## FOR SIMPLICITY
nosePoint = [nosePoint[0][0], nosePoint[0][1]]
topBackfinPoint = [topBackfinPoint[0][0], topBackfinPoint[0][1]]
botBackfinPoint = [botBackfinPoint[0][0], botBackfinPoint[0][1]]
```

```
    return output_w_drawing, nosePoint, topBackfinPoint, botBackfinPoint
```

LISTING D.2: Python example

D.3 Code: Fin Detection

```
def threshold_fin(image, nosepoint, image_w_features):
        global o
        global b
        global w
        global b170

        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)
        #display(thresh)

        ret, labels, stats, centroids = cv2.connectedComponentsWithStats(thresh\
                                                     , connectivity = 8)
        area = sorted(stats[:,4], reverse = True)

        area_3_largest = np.where(stats[:,4] == area[2])
        area_4_largest = np.where(stats[:,4] == area[3])
        area_5_largest = np.where(stats[:,4] == area[4])

        area_3_largest = np.asarray(area_3_largest)
        area_4_largest = np.asarray(area_4_largest)
        area_5_largest = np.asarray(area_5_largest)

        area_3_largest = area_3_largest.ravel()
        area_4_largest = area_4_largest.ravel()
        area_5_largest = area_5_largest.ravel()

        three_areas = [area[2], area[3], area[4]]

        centroid_3_area = centroids[area_3_largest]
        centroid_4_area = centroids[area_4_largest]
        centroid_5_area = centroids[area_5_largest]

        vector3 = centroid_3_area-nosepoint
        vector4 = centroid_4_area-nosepoint
        vector5 = centroid_5_area-nosepoint

        distance3 = math.sqrt(vector3[0,0]**2+vector3[0,1]**2)
        distance4 = math.sqrt(vector4[0,0]**2+vector4[0,1]**2)
        distance5 = math.sqrt(vector5[0,0]**2+vector5[0,1]**2)

        if (three_areas[0] & three_areas[1] & three_areas[2])>800:

            if (distance3 < distance4) & (distance3 < distance5):
                top = min(stats[area_3_largest,cv2.CC_STAT_TOP],\
                    stats[area_3_largest,cv2.CC_STAT_TOP])
                left = min(stats[area_3_largest,cv2.CC_STAT_LEFT],\
                    stats[area_3_largest,cv2.CC_STAT_LEFT])
                bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]+stats[area_3_largest,\
                    cv2.CC_STAT_HEIGHT],stats[area_3_largest,cv2.CC_STAT_TOP]\
                    +stats[area_3_largest,cv2.CC_STAT_HEIGHT])
                right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]+stats\
                            [area_3_largest,cv2.CC_STAT_WIDTH],\
                    stats[area_3_largest,cv2.CC_STAT_LEFT]+stats\
                            [area_3_largest,cv2.CC_STAT_WIDTH])
                boundingrect = cv2.rectangle(image_w_features, (left, top),\
                            (right, bot),(0,0,255),3)
                o = o+1


            elif (distance4 < distance3) & (distance4<distance5):


                top = min(stats[area_4_largest,cv2.CC_STAT_TOP],\
                          stats[area_4_largest,cv2.CC_STAT_TOP])
                left = min(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                           ,stats[area_4_largest,cv2.CC_STAT_LEFT])
                bot = max(stats[area_4_largest,cv2.CC_STAT_TOP]\
                          +stats[area_4_largest,cv2.CC_STAT_HEIGHT]\
                          ,stats[area_4_largest,cv2.CC_STAT_TOP]\
                          +stats[area_4_largest,cv2.CC_STAT_HEIGHT])
                right = max(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                            +stats[area_4_largest,cv2.CC_STAT_WIDTH]\
                            ,stats[area_4_largest,cv2.CC_STAT_LEFT]\
                            +stats[area_4_largest,cv2.CC_STAT_WIDTH])
                boundingrect = cv2.rectangle(image_w_features, (left, top)\
                            , (right, bot),(0,0,255),3)
                o = o+1

            elif (distance5 < distance3) & (distance5 < distance4):

                top = min(stats[area_5_largest,cv2.CC_STAT_TOP]\
                          ,stats[area_5_largest,cv2.CC_STAT_TOP])
                left = min(stats[area_5_largest,cv2.CC_STAT_LEFT]\
                           ,stats[area_5_largest,cv2.CC_STAT_LEFT])
                bot = max(stats[area_5_largest,cv2.CC_STAT_TOP]\
                          +stats[area_5_largest,\
                    cv2.CC_STAT_HEIGHT],stats[area_5_largest,cv2.CC_STAT_TOP]+\
                    stats[area_5_largest,cv2.CC_STAT_HEIGHT])
                right = max(stats[area_5_largest,cv2.CC_STAT_LEFT]\
                            +stats[area_5_largest,cv2.CC_STAT_WIDTH],\
                    stats[area_5_largest,cv2.CC_STAT_LEFT]\
                            +stats[area_5_largest,cv2.CC_STAT_WIDTH])

                boundingrect = cv2.rectangle(image_w_features, \
                                (left, top), (right, bot),(0,0,255),3)
                o = o+1

        elif (three_areas[0] & three_areas[1])>800:

            if (distance3 < distance4):
                top = min(stats[area_3_largest,cv2.CC_STAT_TOP]\
                        ,stats[area_3_largest,cv2.CC_STAT_TOP])
                left = min(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                         ,stats[area_3_largest,cv2.CC_STAT_LEFT])
                bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]\
                        +stats[area_3_largest,cv2.CC_STAT_HEIGHT],\
                    stats[area_3_largest,cv2.CC_STAT_TOP]\
                        +stats[area_3_largest,cv2.CC_STAT_HEIGHT])
                right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                          +stats[area_3_largest,cv2.CC_STAT_WIDTH],\
                    stats[area_3_largest,cv2.CC_STAT_LEFT]\
                          +stats[area_3_largest,cv2.CC_STAT_WIDTH])
                boundingrect = cv2.rectangle(image_w_features, \
                          (left, top), (right, bot),(0,0,255),3)
                o = o+1


            elif (distance4 < distance3):
                top = min(stats[area_4_largest,cv2.CC_STAT_TOP]\
                        ,stats[area_4_largest,cv2.CC_STAT_TOP])
                left = min(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                         ,stats[area_4_largest,cv2.CC_STAT_LEFT])
                bot = max(stats[area_4_largest,cv2.CC_STAT_TOP]\
                        +stats[area_4_largest,cv2.CC_STAT_HEIGHT],\
                    stats[area_4_largest,cv2.CC_STAT_TOP]\
                        +stats[area_4_largest,cv2.CC_STAT_HEIGHT])
                right = max(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                          +stats[area_4_largest,cv2.CC_STAT_WIDTH],\
                    stats[area_4_largest,cv2.CC_STAT_LEFT]\
                          +stats[area_4_largest,cv2.CC_STAT_WIDTH])
                boundingrect = cv2.rectangle(image_w_features,\
                          (left, top), (right, bot),(0,0,255),3)
                o = o+1

        elif three_areas[0]>800:

            top = min(stats[area_3_largest,cv2.CC_STAT_TOP]\
                    ,stats[area_3_largest,cv2.CC_STAT_TOP])
            left = min(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                     ,stats[area_3_largest,cv2.CC_STAT_LEFT])
            bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]\
                    +stats[area_3_largest,cv2.CC_STAT_HEIGHT],\
                stats[area_3_largest,cv2.CC_STAT_TOP]\
                    +stats[area_3_largest,cv2.CC_STAT_HEIGHT])
            right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                      +stats[area_3_largest,cv2.CC_STAT_WIDTH],\
                stats[area_3_largest,cv2.CC_STAT_LEFT]\
                      +stats[area_3_largest,cv2.CC_STAT_WIDTH])
            boundingrect = cv2.rectangle(image_w_features, \
                      (left, top), (right, bot),(0,0,255),3)
            o = o+1


        else:
            ret, thresh = cv2.threshold(gray, 100, 255\
                                        , cv2.THRESH_BINARY_INV)
```

```
ret, labels, stats, centroids = \
cv2.connectedComponentsWithStats(thresh, \
                                 connectivity = 8)
area = sorted(stats[:,4], reverse = True)

area_3_largest = np.where(stats[:,4] == area[2])
area_4_largest = np.where(stats[:,4] == area[3])
area_5_largest = np.where(stats[:,4] == area[4])

three_areas = [area[2], area[3], area[4]]

area_3_largest = np.asarray(area_3_largest)
area_4_largest = np.asarray(area_4_largest)
area_5_largest = np.asarray(area_5_largest)

area_3_largest = area_3_largest.ravel()
area_4_largest = area_4_largest.ravel()
area_5_largest = area_5_largest.ravel()

centroid_3_area = centroids[area_3_largest]
centroid_4_area = centroids[area_4_largest]
centroid_5_area = centroids[area_5_largest]

vector3 = centroid_3_area-nosepoint
vector4 = centroid_4_area-nosepoint
vector5 = centroid_5_area-nosepoint


distance3 = math.sqrt(vector3[0,0]**2+vector3[0,1]**2)
distance4 = math.sqrt(vector4[0,0]**2+vector4[0,1]**2)
distance5 = math.sqrt(vector5[0,0]**2+vector5[0,1]**2)
second_three_areas = [area[2], area[3], area[4]]


if (three_areas[0] & three_areas[1] & three_areas[2])>800:

    if (distance3 < distance4) & (distance3 < distance5):
        top = min(stats[area_3_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_3_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_3_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]\
                  +stats[area_3_largest,\
            cv2.CC_STAT_HEIGHT],stats[area_3_largest,cv2.CC_STAT_TOP]+\
            stats[area_3_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_3_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_3_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_3_largest,cv2.CC_STAT_WIDTH])
        print('Distance 3 is smallest and I"m in here now')
        boundingrect = cv2.rectangle(image_w_features,\
                    (left, top), (right, bot),(0,0,255),3)
        b = b+1
    elif (distance4 < distance3) & (distance4<distance5):


        top = min(stats[area_4_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_4_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_4_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_4_largest,cv2.CC_STAT_TOP]\
                  +stats[area_4_largest,cv2.CC_STAT_HEIGHT]\
            ,stats[area_4_largest,cv2.CC_STAT_TOP]\
                  +stats[area_4_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_4_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_4_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_4_largest,cv2.CC_STAT_WIDTH])

        boundingrect = cv2.rectangle(image_w_features,\
                        (left, top), (right, bot),(0,0,255),3)
        b = b+1

    elif (distance5 < distance3) & (distance5 < distance4):

        top = min(stats[area_5_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_5_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_5_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_5_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_5_largest,cv2.CC_STAT_TOP]\
                  +stats[area_5_largest,cv2.CC_STAT_HEIGHT]\
            ,stats[area_5_largest,cv2.CC_STAT_TOP]\
                  +stats[area_5_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_5_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_5_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_5_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_5_largest,cv2.CC_STAT_WIDTH])

        boundingrect = cv2.rectangle(image_w_features, \
                        (left, top), (right, bot),(0,0,255),3)
        b = b+1

elif (three_areas[0] & three_areas[1])>800:

    if (distance3 < distance4):
        top = min(stats[area_3_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_3_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_3_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]\
                  +stats[area_3_largest,cv2.CC_STAT_HEIGHT]\
            ,stats[area_3_largest,cv2.CC_STAT_TOP]\
                  +stats[area_3_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_3_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_3_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_3_largest,cv2.CC_STAT_WIDTH])
        boundingrect = cv2.rectangle(image_w_features,\
                        (left, top), (right, bot),(0,0,255),3)
        b = b+1


    elif (distance4 < distance3):
        top = min(stats[area_4_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_4_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_4_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_4_largest,cv2.CC_STAT_TOP]\
                  +stats[area_4_largest,cv2.CC_STAT_HEIGHT]\
            ,stats[area_4_largest,cv2.CC_STAT_TOP]\
                  +stats[area_4_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_4_largest,cv2.CC_STAT_LEFT\
                         ]+stats[area_4_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_4_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_4_largest,cv2.CC_STAT_WIDTH])

        boundingrect = cv2.rectangle(image_w_features,\
                            (left, top), (right, bot),(0,0,255),3)
        b = b+1

elif three_areas[0]>800:

    top = min(stats[area_3_largest,cv2.CC_STAT_TOP]\
              ,stats[area_3_largest,cv2.CC_STAT_TOP])
    left = min(stats[area_3_largest,cv2.CC_STAT_LEFT]\
               ,stats[area_3_largest,cv2.CC_STAT_LEFT])
    bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]\
              +stats[area_3_largest,cv2.CC_STAT_HEIGHT]\
        ,stats[area_3_largest,cv2.CC_STAT_TOP]\
              +stats[area_3_largest,cv2.CC_STAT_HEIGHT])
    right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                +stats[area_3_largest,cv2.CC_STAT_WIDTH]\
        ,stats[area_3_largest,cv2.CC_STAT_LEFT]\
                +stats[area_3_largest,cv2.CC_STAT_WIDTH])

    boundingrect = cv2.rectangle(image_w_features,\
                    (left, top), (right, bot),(0,0,255),3)
    b = b+1



else:

    ret, thresh = cv2.threshold(gray, 170, 255,\
                                cv2.THRESH_BINARY_INV)

    ret, labels, stats, centroids = \
    cv2.connectedComponentsWithStats(thresh, connectivity = 8)
    area = sorted(stats[:,4], reverse = True)

    area_3_largest = np.where(stats[:,4] == area[2])  # Usually the pectoral fin
    area_4_largest = np.where(stats[:,4] == area[3])  # Hopefully anal fin
```

```
area_5_largest = np.where(stats[:,4] == area[4])

three_areas = [area[2], area[3], area[4]]

area_3_largest = np.asarray(area_3_largest)
area_4_largest = np.asarray(area_4_largest)
area_5_largest = np.asarray(area_5_largest)

area_3_largest = area_3_largest.ravel()
area_4_largest = area_4_largest.ravel()
area_5_largest = area_5_largest.ravel()

centroid_3_area = centroids[area_3_largest]
centroid_4_area = centroids[area_4_largest]
centroid_5_area = centroids[area_5_largest]

vector3 = centroid_3_area-nosepoint
vector4 = centroid_4_area-nosepoint
vector5 = centroid_5_area-nosepoint


distance3 = math.sqrt(vector3[0,0]**2+vector3[0,1]**2)
distance4 = math.sqrt(vector4[0,0]**2+vector4[0,1]**2)
distance5 = math.sqrt(vector5[0,0]**2+vector5[0,1]**2)
second_three_areas = [area[2], area[3], area[4]]

if (three_areas[0] & three_areas[1] & three_areas[2])>800:

    if (distance3 < distance4) & (distance3 < distance5):
        top = min(stats[area_3_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_3_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_3_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]\
                  +stats[area_3_largest,cv2.CC_STAT_HEIGHT]\
            ,stats[area_3_largest,cv2.CC_STAT_TOP]\
                  +stats[area_3_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_3_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_3_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_3_largest,cv2.CC_STAT_WIDTH])
        boundingrect = cv2.rectangle(image_w_features,\
                          (left, top), (right, bot),(0,0,255),3)
        b170 = b170+1


    elif (distance4 < distance3) &\
    (distance4<distance5):


        top = min(stats[area_4_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_4_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_4_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_4_largest,cv2.CC_STAT_TOP]\
                  +stats[area_4_largest,cv2.CC_STAT_HEIGHT]\
            ,stats[area_4_largest,cv2.CC_STAT_TOP]\
                  +stats[area_4_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_4_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_4_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_4_largest,cv2.CC_STAT_WIDTH])

        boundingrect = cv2.rectangle(image_w_features,\
                               (left, top), (right, bot),(0,0,255),3)
        b170 = b170+1

    elif (distance5 < distance3) & (distance5 < distance4):

        top = min(stats[area_5_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_5_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_5_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_5_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_5_largest,cv2.CC_STAT_TOP]\
                  +stats[area_5_largest,cv2.CC_STAT_HEIGHT]\
            ,stats[area_5_largest,cv2.CC_STAT_TOP]\
                  +stats[area_5_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_5_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_5_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_5_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_5_largest,cv2.CC_STAT_WIDTH])

        boundingrect = cv2.rectangle(image_w_features,\
                    (left, top), (right, bot),(0,0,255),3)
        b170 = b170+1

elif (three_areas[0] & three_areas[1])>800:

    if (distance3 < distance4):
        top = min(stats[area_3_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_3_largest,cv2.CC_STAT_TOP])
        left = min(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                   ,stats[area_3_largest,cv2.CC_STAT_LEFT])
        bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]\
                  +stats[area_3_largest,cv2.CC_STAT_HEIGHT]\
            ,stats[area_3_largest,cv2.CC_STAT_TOP]\
                  +stats[area_3_largest,cv2.CC_STAT_HEIGHT])
        right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_3_largest,cv2.CC_STAT_WIDTH]\
            ,stats[area_3_largest,cv2.CC_STAT_LEFT]\
                    +stats[area_3_largest,cv2.CC_STAT_WIDTH])


        boundingrect = cv2.rectangle(image_w_features,\
                        (left, top), (right, bot),(0,0,255),3)
        b170 = b170+1


    elif (distance4 < distance3):
        top = min(stats[area_4_largest,cv2.CC_STAT_TOP]\
                  ,stats[area_4_largest,cv2.CC_STAT_TOP])
              left = min(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                          ,stats[area_4_largest,cv2.CC_STAT_LEFT])
              bot = max(stats[area_4_largest,cv2.CC_STAT_TOP]\
                        +stats[area_4_largest,cv2.CC_STAT_HEIGHT]\
                  ,stats[area_4_largest,cv2.CC_STAT_TOP]\
                        +stats[area_4_largest,cv2.CC_STAT_HEIGHT])
              right = max(stats[area_4_largest,cv2.CC_STAT_LEFT]\
                          +stats[area_4_largest,cv2.CC_STAT_WIDTH]\
                  ,stats[area_4_largest,cv2.CC_STAT_LEFT]\
                          +stats[area_4_largest,cv2.CC_STAT_WIDTH])

              boundingrect = cv2.rectangle(image_w_features, \
                          (left, top), (right, bot),(0,0,255),3)
              b170 = b170+1

      elif three_areas[0]>800:

          top = min(stats[area_3_largest,cv2.CC_STAT_TOP]\
                    ,stats[area_3_largest,cv2.CC_STAT_TOP])
          left = min(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                     ,stats[area_3_largest,cv2.CC_STAT_LEFT])
          bot = max(stats[area_3_largest,cv2.CC_STAT_TOP]\
                    +stats[area_3_largest,cv2.CC_STAT_HEIGHT]\
              ,stats[area_3_largest,cv2.CC_STAT_TOP]\
                    +stats[area_3_largest,cv2.CC_STAT_HEIGHT])
          right = max(stats[area_3_largest,cv2.CC_STAT_LEFT]\
                      +stats[area_3_largest,cv2.CC_STAT_WIDTH]\
              ,stats[area_3_largest,cv2.CC_STAT_LEFT]\
                      +stats[area_3_largest,cv2.CC_STAT_WIDTH])

          boundingrect = cv2.rectangle(image_w_features, \
                                  (left, top), (right, bot),(0,0,255),3)
          b170 = b170+1

      else:

          font = cv2.FONT_HERSHEY_SIMPLEX
          cv2.putText(image_w_features,'No pectoral Fin detected'\
                      ,(10,700), font, 2,(0,0,255),2,cv2.LINE_AA)
          w = w+1




  cv2.imwrite(r'C:\Users\ablec\Documents\MASTER\FIshFolder\IMR_results/'\
          +str(i)+'.jpg', image_w_features)


image_list = []
print('getting filenames')
for filename in glob.glob(r'C:\Users\ablec\Documents\MASTER\FIshFolder\
                                    IMR_dataset/*.JPG'): #assuming jpg
    im=Image.open(filename)
    image_list.append(im)

b170 = 0 #Binary 170 used
b = 0 # Binary 100 used
o = 0 # Otsus Threshold used
w = 0 # No fin found
i = 1
for image in image_list:
    path = image.filename
    filename = os.path.basename(path)
    image = np.asarray(image)

    output_img, onlyContour = find_largest_foreground_with_kmeans_noCrop(image)
    nosePoint, topBackfinPoint, botBackfinPoint,\
    output_image_w_features = find_nose_and_backfintips(onlyContour,output_img)
    threshold_fin(output_img, nosePoint, output_image_w_features)
    print(str(i) + " images done out of "+str(len(image_list)))
    print(str((i/len(image_list)*100))+ "% done")
    i = i+1

print("Classified images using otsu's threshold: " + str(o))
print("Classified images using binary 100 threshold: " + str(b))
print("Classified images using binary 170 threshold: " + str(b170))
print("Did not find pectoral fin: " + str(w))
```

Listing D.3: Python example

D.4 Code: Get Length and Find Gill Edge

```
def getLength(self,nose, top, bot):
    import numpy as np
    #FIND POINT BETWEEN BACK TIPS.
    middle = [(top[0]+bot[0])/2,(top[1]+bot[1])/2]
    a, b = np.subtract(nose, middle)
    length = np.sqrt(np.square(a) + np.square(b))
    print(length)
    return length

def findGillCoverEdge_v3_2(self, image, cnt_image, nose, filename, gillColumn, length):
    ## Import library stuff
    import cv2
    import numpy as np
    import matplotlib.pyplot as plt
    from scipy import stats
    version = 'v3_2_20190506/test_vert_blur/test_vert_blur_andWidth_Tallness_V4/'

    cnt = cnt_image
    BGR = image
    RGB = cv2.cvtColor(BGR, cv2.COLOR_BGR2RGB)
```

```
## CURRENT WAY TO ESTIMATE THE ROI. MIGHT DO IT DIFFERENT LATER. FOR EXAMPLE BY USING THE
## TOTAL LENGTH OF FISH AND ESTIMATE FROM THERE
start = int(np.round(length * 0.13))
est = int(np.round(length * 0.19))

verticalRoi = cnt[:, nose[0] + start:nose[0] + est]

## highest
height = np.sum(verticalRoi[100])

height, width = verticalRoi.shape
shortest_idx = np.sum(verticalRoi, axis=0).argmin()
## Finds first value of 255?
upper = verticalRoi[:, shortest_idx].argmax()
#upper = nose[1] - 30



## Find last value of 255
lower = height - verticalRoi[:, shortest_idx][::-1].argmax()
#lower = nose[1] + 30

roi = RGB[upper:lower, nose[0] + start:nose[0] + est]

gray = cv2.cvtColor(roi, cv2.COLOR_RGB2GRAY)
blur = cv2.medianBlur(gray, 3)

blur = np.float32(blur)

## VERTICAL BLUR
kernel = np.zeros((5, 3), np.float32)

kernel[0] = [1, 2, 1]
kernel[1] = [1, 2, 1]
kernel[2] = [0, 0, 0]
kernel[3] = [1, 2, 1]
kernel[4] = [1, 2, 1]

dst = cv2.filter2D(blur/16, -1, kernel)
dst = dst - dst.min()
dst = (dst / dst.max()) * 255

blur = dst
blur = np.uint8(blur)
blurMedian = np.median(blur)

ret, th1 = cv2.threshold(blur, blurMedian, 255, cv2.THRESH_BINARY_INV)

th4 = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
cv2.THRESH_BINARY_INV, 15, 4)

##MAKE OWN EDGE DETECTOR

M, N = gray.shape
```

```
gray_w_frame = np.zeros((M + 2, N + 2))
gray_w_frame[1:M + 1, 1:N + 1] = gray

blur_w_frame = np.zeros((M + 2, N + 2))
blur_w_frame[1:M + 1, 1:N + 1] = blur
## MAKE BLACK FRAME FOR COMPATABILITY

##
vertical_edge = np.zeros(gray.shape)

for m in range(0, M):
    for n in range(0, N):
        vertical_edge[m, n] = np.sum(blur_w_frame[m:m + 3, n + 2]\
- blur_w_frame[m:m + 3, n]) + blur_w_frame[m + 1, n + 2] - 2 * gray_w_frame[m + 1, n]

vertical_edge2 = vertical_edge[0:M, 1:N - 1]

vertical_edge3 = vertical_edge2.copy()

## MEDIAN
median = np.median(vertical_edge2)

median3 = np.median(vertical_edge3)

ret, veth3 = cv2.threshold(vertical_edge3, median3 * 1.03,\
vertical_edge3.max(), cv2.THRESH_BINARY)

#blended_copy = blended.copy()
#M, N = blended.shape
#blended_copy = blended_copy[:, 1:(N - 1)]

#th1_copy = th1.copy()
#M, N = th1.shape
#th1_copy = th1_copy[:, 1:(N - 1)]

th4_copy = th4.copy()
M, N = th4.shape
th4_copy = th4_copy[:, 1:(N - 1)]

veth3 = np.uint8(veth3)
blended43=cv2.addWeighted(src1=th4_copy,alpha=0.5,src2=veth3,beta=0.5,gamma=0)

combined43 = blended43.copy()
combined43[combined43 != combined43.max()] = 0

## MAKE OBJECTS
ret, labels43, stats, centroids = cv2.connectedComponentsWithStats(image=combined43, connect
# display(labels)

areas = stats[:, cv2.CC_STAT_AREA]
areas = np.sort(areas)
area = areas[len(areas) - 2]

## index of largest edge area
idx = np.where(stats[:, cv2.CC_STAT_AREA] == area)
```

```
##Using tallest instead:
tallness = stats[:, cv2.CC_STAT_HEIGHT]
tallness = np.sort(tallness)
tallest = tallness[len(tallness) - 2]

## index of largest/tallest edge area
idx = np.where(stats[:, cv2.CC_STAT_HEIGHT] == tallest)

##DISPLAY ONLY LARGEST/TALLEST EDGE

edge43 = labels43.copy()
edge43[edge43 != idx] = 0
edge43[edge43 == idx] = 1
# display(edge)
# labels[labels!=6] = 0
# display(labels)
# stats

prev = 0

heights = np.sum(edge43, axis=0)
highest = heights.max()
print(heights)
gillEnd = len(heights)
for i in range(0, len(heights) - 1):
    ## IF THIS IS VERY MUCH SHORTER THAN THE HIGHEST
    if (heights[i + 1] == 0) & (heights[i] != 0):
        print(i)
        gillEnd = i
        break

safetyMargin = 0

gillAt = nose[0] + start + gillEnd + safetyMargin

##CREATE NEW EDGE IMAGE THAT FITS THE ROI IN THE BEGINNING
M, N = gray.shape

edge43_resized = np.zeros(gray.shape)
edge43_resized[0:M, 1:N - 1] = edge43
edge43_resized = np.abs(edge43_resized - 1)

roi_copy = roi.copy()
roi_copy[:, :, 0] = np.multiply(roi_copy[:, :, 0], edge43_resized)
roi_copy[roi_copy[:, :, 0] == 0] = (0, 0, 255)

roi_copy2 = roi_copy.copy()

roi_copy = roi.copy()
# np.multiply(roi_copy2,edge)

RGB_copy = RGB.copy()
RGB_copy[upper:lower, nose[0] + start:nose[0] + est] = roi_copy2
```

```
    output_w_drawing = RGB_copy

    return gillAt, upper, lower, output_w_drawing
```

Listing D.4: Python example

D.5 Code: Simple Blob Detector Implementation

```
def findBlobSpotsV1(self,input_image, cnt_image, nose, gillPositionX, upper, lower, filename):
    import numpy as np
    import cv2

    image = input_image.copy()

    head = image[upper:lower, nose[0]:gillPositionX]
    cnt_head = cnt_image[upper:lower, nose[0]:gillPositionX]

    blank = np.zeros((1, 1))

    # Setup SimpleBlobDetector parameters.
    params = cv2.SimpleBlobDetector_Params()

    # Change thresholds
    params.minThreshold = 10;
    params.maxThreshold = 170;

    # Filter by Area.
    params.filterByArea = True
    params.minArea = 15

    # Filter by Circularity
    params.filterByCircularity = True
    params.minCircularity = 0.7

    # Filter by Convexity
    params.filterByConvexity = True
    params.minConvexity = 0.9

    # Filter by Inertia
    params.filterByInertia = True
    params.minInertiaRatio = 0.2

    # Create a detector with the parameters
    ver = (cv2.__version__).split('.')
    if int(ver[0]) < 3:
        detector = cv2.SimpleBlobDetector(params)
    else:
        detector = cv2.SimpleBlobDetector_create(params)

    # detector = cv2.SimpleBlobDetector_create()
    keypoints = detector.detect(head)
    blobs = cv2.drawKeypoints(head, keypoints, blank, (255, 0, 0), cv2.DRAW_MATCHES_FLAGS_DRAW_R

    head = cv2.cvtColor(head, cv2.COLOR_BGR2RGB)
    blobs = cv2.cvtColor(blobs, cv2.COLOR_BGR2RGB)

    return head, cnt_head, blobs, keypoints
```

LISTING D.5: Python example

D.6 Blob Detection: Dark Spot Detector

```
def findCustomSpotsV1C(self,input_image, cnt_head):
    import numpy as np
    import cv2

    ## METHOD PARAMS:

    #kernel size for gaussian blur before adaptive threshold
    k = 9
    #factor to multiply gaussian blur with:
    q = 1.0
    #factor to multiply k with, when looking for local minimas.
    #This should be increased along with increased kernel size.
    p = 0.8
    #Sigma for gaussian blur:
    sgm = 1

    #kernel size for blur2
    k2 = 3
    #sgm for blur 2
    sgm2 = 3

    # values for adaptive thresh to find possible spot areas:
    a = 21
    b = 14


    ## 1 INIT AND PREPARE IMAGES
    gbr = input_image
    cnt = cnt_head
    ## 2 CONVERT TO GRAYSCALE
    gray = cv2.cvtColor(gbr, cv2.COLOR_BGR2GRAY)

    ## 3 CREATE MEDIAN FOR FACE OF FISH TO BE USED FOR THRESHOLDING.
    median = np.median(gray[cnt != 0])

    ## 4 CREATE GAUSSIAN BLUR OF IMAGE TO CREATE ADAPTIVE THRESHOLD
    gauss_blur = cv2.GaussianBlur(gray, (k, k), sgm, sgm)
    th_a = cv2.adaptiveThreshold(gauss_blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BIN

    ## 5 CREATE SECOND GAUSSIAN BLUR AND APPLY NORMAL THRESHOLD
    gauss_blur2 = cv2.GaussianBlur(gray, (k2, k2), sgm2, sgm2)
```

```
ret, th = cv2.threshold(gauss_blur2, median*q, 255, cv2.THRESH_BINARY_INV)

## 6 BLEND THE NORMAL AND ADAPTIVE THRESHOLDS TOGETHER, KEEP ONLY BRIGHTEST
blended = cv2.addWeighted(src1=th_a, alpha=0.3, src2=th, beta=0.7, gamma=0)
img2 = blended.copy()
img2[blended != blended.max()] = 0

## 7 INVERT THE CNT IMAGE AN APPLY ADAPTIVE THRESHOLD. BLEND TOGETHER AND KEEP BRIGHTEST
cnt2 = cnt.copy()
cnt2[cnt == cnt.min()] = cnt.max()
cnt2[cnt == cnt.max()] = cnt.min()
th_a_cnt = cv2.adaptiveThreshold(cnt2, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINAR

blended2 = cv2.addWeighted(src1=th_a_cnt, alpha=0.3, src2=img2, beta=0.7, gamma=0)

img3 = blended2.copy()

img3[blended2 != blended2.max()] = 0


## 8 FIND LOCAL MINIMAS; WHICH WILL BE SPOT CANDIDATES
M, N = gray.shape
# g = cv2.cvtColor(gray,cv2.COLOR_RGB2GRAY)
g = gauss_blur.copy()
g_frame = np.zeros((M + 2, N + 2))
g_frame[1:M + 1, 1:N + 1] = g
g = g_frame
black_img = np.zeros((M, N))
for m in range(0, M):
    for n in range(0, N):
        # print(g[m,n])
        if g[m, n] < (median*p):
            if (g[m, n] <= g[m - 1, n + 1]) & (g[m, n] <= g[m - 1, n]) & (g[m, n]
            <= g[m - 1, n - 1]) & (
                    g[m, n] <= g[m, n - 1]) & (g[m, n]
                    <= g[m + 1, n - 1]) & (g[m, n] <= g[m + 1, n]) & (
                    g[m, n] <= g[m + 1, n + 1]):
                # print(g[m,n])
                black_img[m, n] = 255

black_img = np.uint8(black_img)

blended3 = cv2.addWeighted(src1=black_img, alpha=0.3, src2=img3, beta=0.7, gamma=0)

result1 = blended3.copy()

result1[blended3 != blended3.max()] = 0

result2 = cv2.addWeighted(src1=gray, alpha=0.3, src2=result1, beta=0.7, gamma=0)

## CREATE OBJECTS
ret, labels, stats, centroids = cv2.connectedComponentsWithStats(result1, 8)

keypoints = centroids[1:, :]
```

```
    ## Remove the typical eye-points
    for i in range(0, len(keypoints)):
        #print(keypoints[i])
        if keypoints[i, 0] < 50:
            #print(keypoints[i])
            keypoints[i] = [0, 0]

    gbr_copy = gbr.copy()

    rounded = keypoints.round()

    for i in range(0, len(rounded)):
        if rounded[i, 0] != 0:
            cv2.circle(gbr_copy, (int(rounded[i, 0]), int(rounded[i
            , 1])), 2, [0, 0, 255], -1)

    print(median*p)

    return keypoints, gbr_copy
```

Listing D.6: Python example

D.7 Code: Manual Spot Detection, Manual Gill Detection

```
def spotsResultsV1(self,head_image):
    import cv2

    img = head_image.copy()

    M, N = img.shape[:2]

    #################
    ##FUNCTION#######
    #################

    global ixList
    global iyList

    global nxList
    global nyList

    global bxList
    global byList

    gxList = []
    gyList = []

    nxList = []
    nyList = []

    bxList = []
    byList = []
    def draw_circle(event, x, y, flags, param):
        global ix, iy
        if event == cv2.EVENT_LBUTTONDOWN:
            cv2.circle(img, (x, y), 5, (0, 255, 0), -1)
            print([x, y])
            gxList.append(x)
            gyList.append(y)
        if event == cv2.EVENT_RBUTTONDOWN:
            cv2.circle(img, (x, y), 3, (255, 0, 255), -1)
            print([x, y])
            nxList.append(x)
            nyList.append(y)
        if (event == cv2.EVENT_MBUTTONDOWN):
            cv2.circle(img, (x, y), 1, (0, 0, 255), -1)
            print([x, y])
            bxList.append(x)
            byList.append(y)

    cv2.namedWindow(winname='my_drawing', flags=cv2.WINDOW_NORMAL)
    cv2.resizeWindow('my_drawing', N * 9, M * 9)

    cv2.setMouseCallback('my_drawing', draw_circle)

    while True:
        cv2.imshow('my_drawing', img)

        # if (cv2.waitKey(20)) & (0xFF == 27):
        # break
        if cv2.waitKey(20) & 0xFF == 27:
            break
        # if event == cv2.EVENT_LBUTTONDBLCLK:
        #   break

    # print(column_diff)
    cv2.destroyAllWindows()
    return gxList, gyList, nxList, nyList, bxList, byList



def gillDiffResultsV3_2(self,image,gillPositionX, nose):
## TO COMPARE RESULTS: MANUALLY MEASURED AND AUTOMATICALLY.
    import cv2

    RGB = image
    M, N = RGB.shape[:2]

    #################
    ##FUNCTION#######
    #################
    column_diff = -1

    def draw_circle(event, x, y, flags, param):
        global ix, iy
        if event == cv2.EVENT_LBUTTONDOWN:
            cv2.circle(RGB, (x, y), 3, (0, 255, 0), -1)
            ix, iy = x, y

    cv2.namedWindow(winname='my_drawing', flags=cv2.WINDOW_NORMAL)
    cv2.resizeWindow('my_drawing', N * 2, M * 2)

    cv2.setMouseCallback('my_drawing', draw_circle)

    ##############################
    ##SHOWING IMAGE WITH OPEN CV##
    ##############################

    while True:
        cv2.imshow('my_drawing', RGB)

        # if (cv2.waitKey(20)) & (0xFF == 27):
        # break
        if cv2.waitKey(20) & 0xFF == 27:
            break
        # if event == cv2.EVENT_LBUTTONDBLCLK:
        #   break

    # print(column_diff)
    cv2.destroyAllWindows()

    column_diff = gillPositionX - ix

    ##Manually measur Head Length
    headLength_mm = ix - nose[0]
    headLength_auto = gillPositionX - nose[0]

    print(column_diff)
    print(headLength_mm)
    print(headLength_auto)

    return column_diff, headLength_mm, headLength_auto
```

Listing D.7: Python example

D.8 Spot Coordinates Comparison

```
path = 'C:/Users/joach/Desktop/Master/Pycharm/IMR_manual_spots_center.csv'



custom_path = 'C:/Users/joach/Desktop/Master/Pycharm/IMR_k59q10p10sgm3_C3.txt'



blob_path = 'C:/Users/joach/Desktop/Master/Pycharm/IMR_auto_spots_center_comb4.txt'
```

```
df1 = pd.read_csv(custom_path,na_values='[]',quotechar='"')

df2 = pd.read_csv(blob_path,na_values='[]',quotechar='"')


df.goodX  = df.goodX.str.replace(' ',"")

df.goodX  = df.goodX.str.replace('[',"")

df.goodX  = df.goodX.str.replace(']',"")

df.goodY  = df.goodY.str.replace(' ',"")

df.goodY  = df.goodY.str.replace('[',"")

df.goodY  = df.goodY.str.replace(']',"")

df.neutralX  = df.neutralX.str.replace(' ',"")

df.neutralX  = df.neutralX.str.replace(']',"")

df.neutralX  = df.neutralX.str.replace('[',"")

df.neutralY  = df.neutralY.str.replace(' ',"")

df.neutralY  = df.neutralY.str.replace(']',"")

df.neutralY  = df.neutralY.str.replace('[',"")

df.badX  = df.badX.str.replace(' ',"")

df.badX  = df.badX.str.replace(']',"")

df.badX  = df.badX.str.replace('[',"")

df.badY  = df.badY.str.replace(' ',"")

df.badY  = df.badY.str.replace(']',"")

df.badY  = df.badY.str.replace('[',"")

df = df.fillna(0)


df.goodX[1].split(',')


df1.X  = df1.X.str.replace(' ',"")
```

```
df1.X  = df1.X.str.replace('[',"")

df1.X  = df1.X.str.replace(']',"")

df1.Y  = df1.Y.str.replace(' ',"")

df1.Y  = df1.Y.str.replace('[',"")

df1.Y  = df1.Y.str.replace(']',"")


df1 = df1.fillna(0)


df2.X  = df2.X.str.replace(' ',"")

df2.X  = df2.X.str.replace('[',"")

df2.X  = df2.X.str.replace(']',"")

df2.Y  = df2.Y.str.replace(' ',"")

df2.Y  = df2.Y.str.replace('[',"")

df2.Y  = df2.Y.str.replace(']',"")


df2 = df2.fillna(0)


# CUSTOM:

## LENGTH OF VECTOR MADE

t1 = 180


Ax1 = np.zeros((len(df1),t1))

Ay1 = np.zeros((len(df1),t1))


for i in range(0,len(df1)):

    if df1.X[i] == 0:

        continue

    print(i)
    arrX = df1.X[i].split(',')
    #arrX = float(arrX)
    arrY = df1.Y[i].split(',')
    #arrY = float(arrY)
    print(arrX)
    for j in range(0,len(arrX)):
        Ax1[i,j] = float(arrX[j])
        Ay1[i,j] = float(arrY[j])


# BLOB:
## LENGTH OF VECTOR MADE
t2 = 40


Ax2 = np.zeros((len(df2),t2))
Ay2 = np.zeros((len(df2),t2))


for i in range(0,len(df2)):
    if df2.X[i] == 0:
        continue
    print(i)
    arrX = df2.X[i].split(',')
    #arrX = float(arrX)
    arrY = df2.Y[i].split(',')
    #arrY = float(arrY)
    print(arrX)
    for j in range(0,len(arrX)):
        Ax2[i,j] = float(arrX[j])
        Ay2[i,j] = float(arrY[j])


## Manual Good:


MGx = np.zeros((len(df2),t))

MGy = np.zeros((len(df2),t))


for i in range(0,len(df2)):

    print(i)

    if df.goodX[i] == 0:

        continue

    arrX = df.goodX[i].split(',')

    #arrX = float(arrX)

    arrY = df.goodY[i].split(',')

    #arrY = float(arrY)

    print(arrX)

    for j in range(0,len(arrX)):

        MGx[i,j] = float(arrX[j])

        MGy[i,j] = float(arrY[j])


## Manual Neutral (neutral/hard/complex)

MNx = np.zeros((len(df2),t))

MNy = np.zeros((len(df2),t))


for i in range(0,len(df2)):

    print(i)

    if df.neutralX[i] == 0:

        continue
    arrX = df.neutralX[i].split(',')

    #arrX = float(arrX)

    arrY = df.neutralY[i].split(',')

    #arrY = float(arrY)

    print(arrX)

    for j in range(0,len(arrX)):

        MNx[i,j] = float(arrX[j])

        MNy[i,j] = float(arrY[j])


## Manual Bad (Not even sure if they are spots at all)

MBx = np.zeros((len(df2),t))

MBy = np.zeros((len(df2),t))


for i in range(0,len(df2)):

    print(i)

    if df.badX[i] == 0:

        continue

    arrX = df.badX[i].split(',')

    #arrX = float(arrX)

    arrY = df.badY[i].split(',')

    #arrY = float(arrY)

    print(arrX)

    for j in range(0,len(arrX)):

        MBx[i,j] = float(arrX[j])

        MBy[i,j] = float(arrY[j])


## Euclidean distance threshold to define successfull detection.
#In pixels. 5 pixels = ca 1mm. Althoug its mostly closer.
```

```
disth = 20


Total_good_detected = 0

Total_neutral_detected = 0

Total_bad_detected = 0

Total_false_detections = 0


Total_good_observed = 0

Total_neutral_observed = 0

Total_bad_observed = 0


# GOOD SPOTS DETECTED:


#FAIL LISTS:

distinctFail = []

complexFail = []

excessFail = []



#nAuto = len(Ax[number][Ax[number]!=0])



for k in range(0,len(df)):

    print(k)

    number=k


    ##Total number of automatic detected spots for this fish:
```

```
nAuto = len(Ax[number][Ax[number]!=0])


good_matches = 0

neutral_matches = 0

bad_matches = 0


for i in range(0,t):

    detected = False

    XtoMatch = MGx[number,i]

    YtoMatch = MGy[number,i]

    for j in range(0,t):

        Xtry = Ax[number,j]

        Ytry = Ay[number,j]

        #print(XtoMatch)

        #print(Xtry)

        if (Xtry!=0) & (detected==False):

            dist = np.sqrt(np.square(XtoMatch-Xtry)+np.square
            (YtoMatch-Ytry))

            if dist<disth:

                good_matches += 1

                detected = True

                #print([Xtry,Ytry])


print(df.Filename[number])

print('Distinct spots detected: ' + str(good_matches) + '/' +
str(df.nGood[number]))


if good_matches!=df.nGood[number]:

    distinctFail.append(df.Filename[number])
```

```
# Neutral/Complex SPOTS DETECTED:

neutral_matches = 0

for i in range(0,t):

    detected = False

    XtoMatch = MNx[number,i]

    YtoMatch = MNy[number,i]

    for j in range(0,t):

        Xtry = Ax[number,j]

        Ytry = Ay[number,j]

        #print(XtoMatch)

        #print(Xtry)

        if (Xtry!=0) & (detected==False):

            dist = np.sqrt(np.square(XtoMatch-Xtry)+np.squar
            e(YtoMatch-Ytry))

            if dist<disth:

                neutral_matches += 1

                detected = True

                #print([Xtry,Ytry])

                #print([XtoMatch,YtoMatch])


print('Complex spots detected: ' + str(neutral_matches) + '/'
+ str(df.nNeutral[number]))

if neutral_matches!=df.nNeutral[number]:

    complexFail.append(df.Filename[number])


# BAD SPOTS DETECTED:
```

```
bad_matches = 0

for i in range(0,t):

    detected = False

    XtoMatch = MBx[number,i]

    YtoMatch = MBy[number,i]

    for j in range(0,t):

        Xtry = Ax[number,j]

        Ytry = Ay[number,j]

        #print(XtoMatch)

        #print(Xtry)

        if (Xtry!=0) & (detected==False):

            dist = np.sqrt(np.square(XtoMatch-Xtry)+np.
            square(YtoMatch-Ytry))

            if dist<disth:

                #print(dist)

                bad_matches += 1

                detected = True

                #print([Xtry,Ytry])

                #print([XtoMatch,YtoMatch])


print('Bad spots detected: ' + str(bad_matches) + '/' + str(df.nBad[number]))


nFalse = nAuto-good_matches-neutral_matches-bad_matches


print('False/Excess Detections: ' + str(nFalse))


Total_good_detected += good_matches

Total_neutral_detected += neutral_matches
```

```
    Total_bad_detected += bad_matches

    Total_false_detections += nFalse

    Total_detections = Total_good_detected+Total_neutral_
    detected+Total_bad_detected+Total_false_detections



    Total_good_observed += df.nGood[number]

    Total_neutral_observed += df.nNeutral[number]

    Total_bad_observed += df.nBad[number]



    if nFalse> 0:

        excessFail.append(df.Filename[number])



print('Total automatic spot detections: '+str(Total_detections-Total_false_detections)
+ ' on ' + str(len(df)) + ' number of images (of fish)')

print('Total false detections: '+ str(Total_false_detections))

print('% false detections: ' + str(Total_false_detections/Total_detections))

print('Total distinct spots detected: ' + str(Total_good_detected) + ' / ' + str(Total_good_obse

print('Total complex spots detected: ' + str(Total_neutral_detected) + ' / ' + str(Total_neutral

print('Total uncertain spots detected (not important, but should not
classify as a false detection either): ' + str(Total_bad_detected) + ' / ' + str(Total_bad_obser
```

Listing D.8: Python example

D.9 Code: create Feature vectors

```
def make_vector(filename, image, nosepoint, topBackfinPoint,\
    botBackfinPoint, pectoral_fin_left, pectoral_fin_top, pectoral_fin_bot\
                , pectoral_fin_right):

    feature_vector = []

    filename, filetype =filename.split(".")
    feature_vector.append(filename)


    #Finding the middle of the pectoral fin vertically
```

```
lower_pec_fin = [pectoral_fin_left, pectoral_fin_top]
higher_pec_fin = [pectoral_fin_left, pectoral_fin_bot]
lower_pec_fin = np.asarray(lower_pec_fin)
lower_pec_fin = lower_pec_fin.ravel()
higher_pec_fin = np.asarray(higher_pec_fin)
higher_pec_fin = higher_pec_fin.ravel()

pec_fin_middle = (lower_pec_fin+higher_pec_fin)/2
pec_fin_middle = int(pec_fin_middle[1])


#Compyting first feature: nose to pectoral fin
pectoral_fin = [pectoral_fin_left, pec_fin_middle]
pectoral_fin_right_loc = [pectoral_fin_right[0], pec_fin_middle]

pectoral_fin = np.asarray(pectoral_fin)
pectoral_fin = pectoral_fin.ravel()
nosepoint = np.asarray(nosepoint)
nosepoint = nosepoint.ravel()

nose_to_pectoral_fin_vector = nosepoint-pectoral_fin
nose_to_pectoral_fin_distance = int(math.sqrt\
    (nose_to_pectoral_fin_vector[0]**2+nose_to_pectoral_fin_vector[1]**2))
feature_vector.append(nose_to_pectoral_fin_distance)
cv2.line(image,(nosepoint[0], nosepoint[1]),\
         (pectoral_fin[0], pectoral_fin[1]),(0,0,255),5)
#DONE with Nose to fin

#From nose to top_back_fin_point
topBackfinPoint = topBackfinPoint.ravel()


#From nose to bot_back_fin_point
botBackfinPoint = botBackfinPoint.ravel()

middle_tips = (topBackfinPoint+botBackfinPoint)/2
#print(str(middle_tips))
nose_middletips_vector = nosepoint-middle_tips
nose_middletips_distance = int(math.sqrt(nose_middletips_vector[0]\
                            **2+nose_middletips_vector[1]**2))
feature_vector.append(nose_middletips_distance)
cv2.line(image,(nosepoint[0], nosepoint[1]),\
         (int(middle_tips[0]), int(middle_tips[1])),(0,0,255),5)

#Right end of pectoral fin to tail middle

pec_fin_right_tail_middle_vector = middle_tips-pectoral_fin_right_loc
pec_fin_right_tail_middle_distance = int(math.sqrt\
    (pec_fin_right_tail_middle_vector[0]**2+pec_fin_right_tail_middle_vector[1]**2))
feature_vector.append(pec_fin_right_tail_middle_distance)

#Adding
df_headlength_file = pd.read_csv\
(r'C:\Users\ablec\Documents\MASTER\head_length_file.txt')
df_headlength_nosepoint = pd.read_csv\
(r'C:\Users\ablec\Documents\MASTER\head_length_nosepoint.txt')
df_headlength_gillpoint = pd.read_csv\
(r'C:\Users\ablec\Documents\MASTER\head_length_gillpoint.txt')
df_headlength_file['nosepoint'] = \
df_headlength_nosepoint['nosepointX'].values
df_headlength_file['gillpoint'] = \
df_headlength_gillpoint['gillPointX'].values
df_headlength_file['headlength'] = \
df_headlength_gillpoint['gillPointX']
df_headlength_file['headlength'] = \
df_headlength_file['gillpoint'] - df_headlength_file['nosepoint']
df_headlength_file.fil  = df_headlength_file.fil.str.replace('.tif',"")

for i in range (0,len(df_headlength_file.fil)):
#Searches through the filenames for a match

    if df_headlength_file.fil[i] == filename:
        feature_vector.append(df_headlength_file.headlength[i])



# Get spot-coordinates from excel file and add to roccect fish
df_spots = pd.read_csv\
(r'C:\Users\ablec\Documents\MASTER\modifiedM_spots_center_rated.txt')
df_spots = df_spots[['Filename','goodX','goodY'\
                    ,'neutralX','neutralY','nGood','nNeutral']]
df_spots.Filename  = df_spots.Filename.str.replace('.tif',"")
df_spots = df_spots.fillna(0)

df_coords = pd.read_csv(r'C:\Users\ablec\Documents\MASTER\upperLeftCoords.txt')
df_spots['orig_coords_x'] = df_coords['Filename'].values
df_spots['orig_coords_y'] = df_coords['Coords'].values
df_spots.orig_coords_x = df_spots.orig_coords_x.str.replace('[',"")
df_spots.orig_coords_y = df_spots.orig_coords_y.str.replace(']',"")

for i in range (0,len(df_spots.Filename)):
#Searches through the filenames for a match

    if df_spots.Filename[i] == filename:
        print('Matched filename found')
        real_coords = [int(df_spots.orig_coords_x\
                           [i]),int(df_spots.orig_coords_y[i])]
        hor_angle = math.atan((-(middle_tips[0]\
                            -nosepoint[0])/(middle_tips[1]-nosepoint[1])))

        if df_spots.nGood[i] == 3:
            xspot1_good, xspot2_good, xspot3_good = df_spots.goodX[i].split(',')
            yspot1_good, yspot2_good, yspot3_good = df_spots.goodY[i].split(',')
            spot1_good = [int(xspot1_good),int(yspot1_good)]
            spot2_good = [int(xspot2_good),int(yspot2_good)]
            spot3_good = [int(xspot3_good),int(yspot3_good)]
            print('The fish had 3 distinct spots')

            spot1_good_rp = [real_coords[0]\
                        +spot1_good[0],real_coords[1]+spot1_good[1]]
            spot2_good_rp = [real_coords[0]\
                        +spot2_good[0],real_coords[1]+spot2_good[1]]
            spot3_good_rp = [real_coords[0]\
                        +spot3_good[0],real_coords[1]+spot3_good[1]]


            angle_spot1_good = math.\
            atan((-(spot1_good_rp[0]-nosepoint[0])/\
                  (spot1_good_rp[1]-nosepoint[1])))
            real_angle_spot1_good = \
            angle_spot1_good-hor_angle
            nose_spot1_vector = nosepoint-spot1_good_rp
            nose_spot1_distance = \
            int(math.sqrt(nose_spot1_vector[0]**2+nose_spot1_vector[1]**2))
            feature_vector.append(nose_spot1_distance)
            feature_vector.append(real_angle_spot1_good)

            angle_spot2_good = math.atan\
            ((-(spot2_good_rp[0]-nosepoint[0])/(spot2_good_rp[1]\
                                            -nosepoint[1])))
            real_angle_spot2_good = angle_spot2_good-hor_angle
            nose_spot2_vector = nosepoint-spot2_good_rp
            nose_spot2_distance = int\
            (math.sqrt(nose_spot2_vector[0]**2+nose_spot2_vector[1]**2))
            feature_vector.append(nose_spot2_distance)
            feature_vector.append(real_angle_spot2_good)

            angle_spot3_good = math.atan\
            ((-(spot3_good_rp[0]-nosepoint[0])/(spot3_good_rp[1]\
                                            -nosepoint[1])))
            real_angle_spot3_good = angle_spot3_good-hor_angle
            nose_spot3_vector = nosepoint-spot3_good_rp
            nose_spot3_distance = int(math.sqrt(nose_spot3_vector[0]\
                                            **2+nose_spot3_vector[1]**2))
            feature_vector.append(nose_spot3_distance)
            feature_vector.append(real_angle_spot3_good)


        elif df_spots.nGood[i] == 2:
            xspot1_good, xspot2_good = df_spots.goodX[i].split(',')
            yspot1_good, yspot2_good = df_spots.goodY[i].split(',')
            spot1_good = [int(xspot1_good),int(yspot1_good)]
            spot2_good = [int(xspot2_good),int(yspot2_good)]
            print('The fish had 2 distcint spots')

            spot1_good_rp = [real_coords[0]+spot1_good[0]\
                            ,real_coords[1]+spot1_good[1]]
            spot2_good_rp = [real_coords[0]+spot2_good[0]\
                            ,real_coords[1]+spot2_good[1]]

            angle_spot1_good = math.atan\
            ((-(spot1_good_rp[0]-nosepoint[0])/(spot1_good_rp[1]-nosepoint[1])))
            real_angle_spot1_good = angle_spot1_good-hor_angle
            nose_spot1_vector = nosepoint-spot1_good_rp
            nose_spot1_distance = int\
            (math.sqrt(nose_spot1_vector[0]**2+nose_spot1_vector[1]**2))
            feature_vector.append(nose_spot1_distance)
            feature_vector.append(real_angle_spot1_good)

            angle_spot2_good = math.atan\
            ((-(spot2_good_rp[0]-nosepoint[0])/(spot2_good_rp[1]-nosepoint[1])))
            real_angle_spot2_good = angle_spot2_good-hor_angle
            nose_spot2_vector = nosepoint-spot2_good_rp
            nose_spot2_distance = int\
            (math.sqrt(nose_spot2_vector[0]**2+nose_spot2_vector[1]**2))
            feature_vector.append(nose_spot2_distance)
            feature_vector.append(real_angle_spot2_good)

            feature_vector.append(0)
            feature_vector.append(0)


        elif df_spots.nGood[i] == 1:
            #xspot1 = df_spots.goodX[i].split(’,’)
            #yspot1 = df_spots.goodY[i].split(’,’)
            spot1_good = [int(df_spots.goodX[i]),int(df_spots.goodY[i])]
            print(’The fish had 1 distinct spot’)

            spot1_good_rp = [real_coords[0]+spot1_good[0],real_coords[1]+spot1_good[1]]

            angle_spot1_good = math.atan\
            ((-(spot1_good_rp[0]-nosepoint[0])/(spot1_good_rp[1]-nosepoint[1])))
            real_angle_spot1_good = angle_spot1_good-hor_angle
            nose_spot1_vector = nosepoint-spot1_good_rp
            nose_spot1_distance = int\
            (math.sqrt(nose_spot1_vector[0]**2+nose_spot1_vector[1]**2))
            feature_vector.append(nose_spot1_distance)
            feature_vector.append(real_angle_spot1_good)

            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)

        else:
            print(’No distinct spots’)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)

        if df_spots.nNeutral[i] == 5:
            xspot1_neu, xspot2_neu, xspot3_neu\
            , xspot4_neu, xspot5_neu = df_spots.neutralX[i].split(’,’)
            yspot1_neu, yspot2_neu, yspot3_neu\
            , yspot4_neu, yspot5_neu = df_spots.neutralY[i].split(’,’)
            spot1_neu = [int(xspot1_neu),int(yspot1_neu)]
            spot2_neu = [int(xspot2_neu),int(yspot2_neu)]
```

```
spot3_neu = [int(xspot3_neu),int(yspot3_neu)]
spot4_neu = [int(xspot4_neu),int(yspot4_neu)]
spot5_neu = [int(xspot5_neu),int(yspot5_neu)]
print('The fish has 5 complex spots')

spot1_neu_rp = [real_coords[0]+spot1_neu[0]\
                ,real_coords[1]+spot1_neu[1]]
spot2_neu_rp = [real_coords[0]+spot2_neu[0]\
                ,real_coords[1]+spot2_neu[1]]
spot3_neu_rp = [real_coords[0]+spot3_neu[0]\
                ,real_coords[1]+spot3_neu[1]]
spot4_neu_rp = [real_coords[0]+spot4_neu[0]\
                ,real_coords[1]+spot4_neu[1]]
spot5_neu_rp = [real_coords[0]+spot5_neu[0]\
                ,real_coords[1]+spot5_neu[1]]

angle_spot1_neu = math.atan((-(spot1_neu_rp[0]\
    -nosepoint[0])/(spot1_neu_rp[1]-nosepoint[1])))
real_angle_spot1_neu = angle_spot1_neu-hor_angle
nose_spot1_vector_neu = nosepoint-spot1_neu_rp
nose_spot1_distance_neu = int(math.sqrt\
    (nose_spot1_vector_neu[0]**2+nose_spot1_vector_neu[1]**2))
feature_vector.append(nose_spot1_distance_neu)
feature_vector.append(real_angle_spot1_neu)

angle_spot2_neu = math.atan((-(spot2_neu_rp[0]\
        -nosepoint[0])/(spot2_neu_rp[1]-nosepoint[1])))
real_angle_spot2_neu = angle_spot2_neu-hor_angle
nose_spot2_vector_neu = nosepoint-spot2_neu_rp
nose_spot2_distance_neu = int(math.sqrt\
        (nose_spot2_vector_neu[0]**2+nose_spot2_vector_neu[1]**2))
feature_vector.append(nose_spot2_distance_neu)
feature_vector.append(real_angle_spot2_neu)

angle_spot3_neu = math.atan((-(spot3_neu_rp[0]\
            -nosepoint[0])/(spot3_neu_rp[1]-nosepoint[1])))
real_angle_spot3_neu = angle_spot3_neu-hor_angle
nose_spot3_vector_neu = nosepoint-spot3_neu_rp
nose_spot3_distance_neu = int(math.sqrt\
            (nose_spot3_vector_neu[0]**2+nose_spot3_vector_neu[1]**2))
feature_vector.append(nose_spot3_distance_neu)
feature_vector.append(real_angle_spot3_neu)

angle_spot4_neu = math.atan((-(spot4_neu_rp[0]\
            -nosepoint[0])/(spot4_neu_rp[1]-nosepoint[1])))
real_angle_spot4_neu = angle_spot4_neu-hor_angle
nose_spot4_vector_neu = nosepoint-spot4_neu_rp
nose_spot4_distance_neu = int(math.sqrt\
            (nose_spot4_vector_neu[0]**2+nose_spot4_vector_neu[1]**2))
feature_vector.append(nose_spot4_distance_neu)
feature_vector.append(real_angle_spot4_neu)

angle_spot5_neu = math.atan((-(spot5_neu_rp[0]\
            -nosepoint[0])/(spot5_neu_rp[1]-nosepoint[1])))
real_angle_spot5_neu = angle_spot5_neu-hor_angle
```

```
        nose_spot5_vector_neu = nosepoint-spot5_neu_rp
        nose_spot5_distance_neu = int(math.sqrt\
                    (nose_spot5_vector_neu[0]**2+nose_spot5_vector_neu[1]**2))
        feature_vector.append(nose_spot5_distance_neu)
        feature_vector.append(real_angle_spot5_neu)


    elif df_spots.nNeutral[i] == 4:
        xspot1_neu, xspot2_neu, xspot3_neu, xspot4_neu \
        = df_spots.neutralX[i].split(',')
        yspot1_neu, yspot2_neu, yspot3_neu, yspot4_neu\
        = df_spots.neutralY[i].split(',')
        spot1_neu = [int(xspot1_neu),int(yspot1_neu)]
        spot2_neu = [int(xspot2_neu),int(yspot2_neu)]
        spot3_neu = [int(xspot3_neu),int(yspot3_neu)]
        spot4_neu = [int(xspot4_neu),int(yspot4_neu)]
        print('The fish has 4 complex spots')


        spot1_neu_rp = [real_coords[0]+spot1_neu[0]\
                        ,real_coords[1]+spot1_neu[1]]
        spot2_neu_rp = [real_coords[0]+spot2_neu[0]\
                        ,real_coords[1]+spot2_neu[1]]
        spot3_neu_rp = [real_coords[0]+spot3_neu[0]\
                        ,real_coords[1]+spot3_neu[1]]
        spot4_neu_rp = [real_coords[0]+spot4_neu[0]\
                        ,real_coords[1]+spot4_neu[1]]

        angle_spot1_neu = math.atan\
        ((-(spot1_neu_rp[0]-nosepoint[0])/(spot1_neu_rp[1]-nosepoint[1])))
        real_angle_spot1_neu = angle_spot1_neu-hor_angle
        nose_spot1_vector_neu = nosepoint-spot1_neu_rp
        nose_spot1_distance_neu = \
        int(math.sqrt(nose_spot1_vector_neu[0]**2+nose_spot1_vector_neu[1]**2))
        feature_vector.append(nose_spot1_distance_neu)
        feature_vector.append(real_angle_spot1_neu)

        angle_spot2_neu = math.atan((-(spot2_neu_rp[0]\
                -nosepoint[0])/(spot2_neu_rp[1]-nosepoint[1])))
        real_angle_spot2_neu = angle_spot2_neu-hor_angle
        nose_spot2_vector_neu = nosepoint-spot2_neu_rp
        nose_spot2_distance_neu = int(math.sqrt\
                (nose_spot2_vector_neu[0]**2+nose_spot2_vector_neu[1]**2))
        feature_vector.append(nose_spot2_distance_neu)
        feature_vector.append(real_angle_spot2_neu)

        angle_spot3_neu = math.atan((-(spot3_neu_rp[0]\
                -nosepoint[0])/(spot3_neu_rp[1]-nosepoint[1])))
        real_angle_spot3_neu = angle_spot3_neu-hor_angle
        nose_spot3_vector_neu = nosepoint-spot3_neu_rp
        nose_spot3_distance_neu = int(math.sqrt\
                (nose_spot3_vector_neu[0]**2+nose_spot3_vector_neu[1]**2))
        feature_vector.append(nose_spot3_distance_neu)
        feature_vector.append(real_angle_spot3_neu)
        angle_spot4_neu = math.atan((-(spot4_neu_rp[0]\
                -nosepoint[0])/(spot4_neu_rp[1]-nosepoint[1])))
        real_angle_spot4_neu = angle_spot4_neu-hor_angle
        nose_spot4_vector_neu = nosepoint-spot4_neu_rp
        nose_spot4_distance_neu = int(math.sqrt\
                (nose_spot4_vector_neu[0]**2+nose_spot4_vector_neu[1]**2))
        feature_vector.append(nose_spot4_distance_neu)
        feature_vector.append(real_angle_spot4_neu)

        feature_vector.append(0)
        feature_vector.append(0)


    elif df_spots.nNeutral[i] == 3:
        xspot1_neu, xspot2_neu, xspot3_neu = df_spots.\
        neutralX[i].split(',')
        yspot1_neu, yspot2_neu, yspot3_neu = df_spots.\
        neutralY[i].split(',')
        spot1_neu = [int(xspot1_neu),int(yspot1_neu)]
        spot2_neu = [int(xspot2_neu),int(yspot2_neu)]
        spot3_neu = [int(xspot3_neu),int(yspot3_neu)]
        print('The fish has 3 complex spots')

        spot1_neu_rp = [real_coords[0]+spot1_neu[0],\
                        real_coords[1]+spot1_neu[1]]
        spot2_neu_rp = [real_coords[0]+spot2_neu[0],\
                        real_coords[1]+spot2_neu[1]]
        spot3_neu_rp = [real_coords[0]+spot3_neu[0],\
                        real_coords[1]+spot3_neu[1]]

        angle_spot1_neu = math.atan((-(spot1_neu_rp[0]\
                        -nosepoint[0])/(spot1_neu_rp[1]-nosepoint[1])))
        real_angle_spot1_neu = angle_spot1_neu-hor_angle
        nose_spot1_vector_neu = nosepoint-spot1_neu_rp
        nose_spot1_distance_neu = int(math.sqrt\
                (nose_spot1_vector_neu[0]**2+nose_spot1_vector_neu[1]**2))
        feature_vector.append(nose_spot1_distance_neu)
        feature_vector.append(real_angle_spot1_neu)

        angle_spot2_neu = math.atan((-(spot2_neu_rp[0]\
                        -nosepoint[0])/(spot2_neu_rp[1]-nosepoint[1])))
        real_angle_spot2_neu = angle_spot2_neu-hor_angle
        nose_spot2_vector_neu = nosepoint-spot2_neu_rp
        nose_spot2_distance_neu = int(math.sqrt\
                (nose_spot2_vector_neu[0]**2+nose_spot2_vector_neu[1]**2))
        feature_vector.append(nose_spot2_distance_neu)
        feature_vector.append(real_angle_spot2_neu)

        angle_spot3_neu = math.atan((-(spot3_neu_rp[0]\
                -nosepoint[0])/(spot3_neu_rp[1]-nosepoint[1])))
        real_angle_spot3_neu = angle_spot3_neu-hor_angle
        nose_spot3_vector_neu = nosepoint-spot3_neu_rp
        nose_spot3_distance_neu = int(math.sqrt\
                (nose_spot3_vector_neu[0]**2+nose_spot3_vector_neu[1]**2))
        feature_vector.append(nose_spot3_distance_neu)
```

```
            feature_vector.append(real_angle_spot3_neu)

            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)


        elif df_spots.nNeutral[i] == 2:
            xspot1_neu, xspot2_neu = df_spots.neutralX[i].split(',')
            yspot1_neu, yspot2_neu = df_spots.neutralY[i].split(',')
            spot1_neu = [int(xspot1_neu),int(yspot1_neu)]
            spot2_neu = [int(xspot2_neu),int(yspot2_neu)]
            print('The fish has 2 complex spots')

            spot1_neu_rp = [real_coords[0]+spot1_neu[0],\
                            real_coords[1]+spot1_neu[1]]
            spot2_neu_rp = [real_coords[0]+spot2_neu[0],\
                            real_coords[1]+spot2_neu[1]]

            angle_spot1_neu = math.atan((-(spot1_neu_rp[0]\
                            -nosepoint[0])/(spot1_neu_rp[1]-nosepoint[1])))
            real_angle_spot1_neu = angle_spot1_neu-hor_angle
            nose_spot1_vector_neu = nosepoint-spot1_neu_rp
            nose_spot1_distance_neu = int(math.sqrt\
                    (nose_spot1_vector_neu[0]**2+nose_spot1_vector_neu[1]**2))
            feature_vector.append(nose_spot1_distance_neu)
            feature_vector.append(real_angle_spot1_neu)

            angle_spot2_neu = math.atan((-(spot2_neu_rp[0]\
                            -nosepoint[0])/(spot2_neu_rp[1]-nosepoint[1])))
            real_angle_spot2_neu = angle_spot2_neu-hor_angle
            nose_spot2_vector_neu = nosepoint-spot2_neu_rp
            nose_spot2_distance_neu = int(math.sqrt\
                    (nose_spot2_vector_neu[0]**2+nose_spot2_vector_neu[1]**2))
            feature_vector.append(nose_spot2_distance_neu)
            feature_vector.append(real_angle_spot2_neu)

            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)


        elif df_spots.nNeutral[i] == 1:
            spot1_neu = [int(df_spots.neutralX[i])\
                        ,int(df_spots.neutralX[i])]
            print('The fish had 1 complex spot')

            spot1_neu_rp = [real_coords[0]+spot1_neu[0],\
                            real_coords[1]+spot1_neu[1]]

            angle_spot1_neu = math.atan((-(spot1_neu_rp[0]\
                    -nosepoint[0])/(spot1_neu_rp[1]-nosepoint[1])))
            real_angle_spot1_neu = angle_spot1_neu-hor_angle
            nose_spot1_vector_neu = nosepoint-spot1_neu_rp
            nose_spot1_distance_neu = int(math.sqrt\
                    (nose_spot1_vector_neu[0]**2+nose_spot1_vector_neu[1]**2))
            feature_vector.append(nose_spot1_distance_neu)
            feature_vector.append(real_angle_spot1_neu)

            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)


        else:
            print('No complex spots')
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)
            feature_vector.append(0)


    outF = open("feature_vectors_labels_whole_dataset.txt", "a")
    outF.write(str(feature_vector))
    outF.write("\n")
    outF.close()
```

Listing D.9: Python example

D.10 Code: Individual Recognition

```
df_spots = pd.read_csv(r'C:\Users\ablec\Documents\MASTER\modifiedM_spots_center_rated.txt')
df_spots = df_spots[['Filename','goodX','goodY','neutralX','neutralY','nGood','nNeutral']]
df_spots.Filename  = df_spots.Filename.str.replace('.tif',"")
df_spots = df_spots.fillna(0)
df_spots['totSpots'] = df_spots['nGood']+df_spots['nNeutral']

df_labels = pd.read_csv\
(r'C:\Users\ablec\Documents\MASTER\feature_vectors_labels_whole_dataset.txt')
df_train = pd.read_csv(r'C:\Users\ablec\Documents\MASTER\feature_vectors_whole_dataset.txt')

df_labels.filename  = df_labels.filename.str.replace('[',"")
df_train.filename  = df_train.filename.str.replace('[',"")
df_labels.angle8  = df_labels.angle8.str.replace(']',"")
df_train.angle8  = df_train.angle8.str.replace(']',"")
df_labels.filename  = df_labels.filename.str.replace("'","")
df_train.filename  = df_train.filename.str.replace("'","")

df_labels["totSpots"] = ""
df_train["totSpots"] = ""

for i in range(0,len(df_labels.filename)):
    for j in range(0,len(df_spots.Filename)):
        if df_labels.filename[i] == df_spots.Filename[j]:
            df_labels.totSpots[i] = df_spots.totSpots[j]

for i in range(0,len(df_train.filename)):
    for j in range(0,len(df_spots.Filename)):
        if df_train.filename[i] == df_spots.Filename[j]:
            df_train.totSpots[i] = df_spots.totSpots[j]

global correclty_class
global wrongly_class
global length_max_error
global correctly_class_3
global wrongly_class_3
global correctly_class_5
global wrongly_class_5
global correctly_class_7
global wrongly_class_7
global score_vector
global correctly_class_9
global wrongly_class_9
global correctly_class_11
global wrongly_class_11
global correctly_class_13
global wrongly_class_13
global correctly_class_15
global wrongly_class_15

df_spots_length = df_train[['nose_spot1','nose_spot2','nose_spot3'\
                            ,'nose_spot4','nose_spot5','nose_spot6'\
                            ,'nose_spot7','nose_spot8']]
df_spots_angle = df_train[['angle1','angle2','angle3','angle4',\
                           'angle_5','angle6','angle7','angle8']]



score_vector = []
length_max_error = []
correclty_class = 0
wrongly_class = 0
correctly_class_3 = 0
wrongly_class_3 = 0
correctly_class_5 = 0
wrongly_class_5 = 0
correctly_class_7 = 0
```

```
wrongly_class_7 = 0
correctly_class_9 = 0
wrongly_class_9 = 0
correctly_class_11 = 0
wrongly_class_11 = 0
correctly_class_13 = 0
wrongly_class_13 = 0
correctly_class_15 = 0
wrongly_class_15 = 0

for i in range (0,len(df_labels.filename)):
    score_vector = []

    print('Checking fish nr.: '+ str(df_labels.filename[i]))
    for j in range (0,len(df_train.filename)):
        if df_labels.filename[i] != df_train.filename[j]:

            score = float(0)
            features_used = 0

            if (df_labels.nose_spot1[i] == 0 ):
                if (df_labels.nose_spot4[i] == 0):
                    features_used +=1
                    n_p_error = abs((df_labels.nosetoPec[i]\
                                     -df_train.nosetoPec[j]))
                    if n_p_error < 8:  # 23
                        added_score = n_p_error/100
                        score = score +1+(0.07-added_score)

                    features_used +=1
                    length_error = abs((df_labels.length[i]\
                                        -df_train.length[j]))
                    if length_error < 9: #36
                        added_score = length_error/100
                        score = score +1+(0.08-added_score)

                    features_used +=1
                    p_t_error =  abs((df_labels.pectoTail[i]\
                                      -df_train.pectoTail[j]))
                    if p_t_error < 8: # 19
                        added_score = p_t_error/100
                        score = score +1+(0.07-added_score)

                    features_used +=1
                    headlength_error =  abs((df_labels.headlength[i]\
                                             -df_train.headlength[j]))
                    if headlength_error < 4: # 19
                        added_score = headlength_error/100
                        score = score +1+(0.03-added_score)

            else:
                features_used +=1
                n_p_error = abs((df_labels.nosetoPec[i]\
                                 -df_train.nosetoPec[j]))
                if n_p_error < 23:  # 23
                    added_score = n_p_error/100
                    score = score +1+(0.22-added_score)


                features_used +=1
                length_error = abs((df_labels.length[i]\
                                     -df_train.length[j]))
                if length_error < 36:#36
                    added_score = length_error/100
                    score = score +1+(0.35-added_score)


                features_used +=1
                p_t_error =  abs((df_labels.pectoTail[i]\
                                  -df_train.pectoTail[j]))
                if p_t_error < 19: # 19
                    added_score = p_t_error/100
                    score = score +1+(0.18-added_score)

                features_used +=1
                headlength_error =  abs((df_labels.headlength[i]\
                                          -df_train.headlength[j]))
                if headlength_error < 11: # 19
                    added_score = headlength_error/100
                    score = score +1+(0.010-added_score)

                features_used+=1
                spot_diff = abs(df_labels.totSpots[i]\
                                -df_train.totSpots[j])
                #if spot_diff > 1:
                #    if  spot_diff < 7:
                #        added_score = spot_diff/1000
                #        score = score+1+(0.007-added_score)

                if spot_diff < 2:
                    added_score = spot_diff/10
                    score = score+1+(0.3-added_score)


            # Beginning to check against spots
            if df_labels.nose_spot1[i] != 0:
                angle_threshold = math.atan(12/df_labels.nose_spot1[i])

                row_length = df_spots_length.iloc[j]
                row_angle = df_spots_angle.iloc[j]
                f = 0
                a = 0
                l = -1
                for k in row_length:
                    l += 1
                    if (k != 0):
                        if f == 0:
                            features_used += 1
                            f = 1
                        spot1_error = abs(df_labels.nose_spot1[i]-k)
                        if spot1_error < 13:
                            added_score = spot1_error/100
                            spot1_error_angle = abs(df_labels.angle1\
                                                    [i]-float(row_angle[l]))
                            if a == 0:
                                features_used+=1
                                a = 1
                            if spot1_error_angle < angle_threshold:
                                added_score = added_score +\
                                (spot1_error_angle/10)
                                score = score +2+\
                                ((0.12+(angle_threshold/10))-added_score)
                                spot_used = k
                            else:
                                spot_used = 0

                if df_labels.nose_spot2[i] != 0:
                    angle_threshold = math.atan\
                    (12/df_labels.nose_spot2[i])
                    a = 0
                    f = 0
                    l = -1
                    for k in row_length:
                        l += 1
                        if (k != 0) & (k != spot_used):
                            if f == 0:
                                features_used += 1
                                f =1
                            spot2_error = abs(df_labels.nose_spot2[i]-k)
                            if spot2_error < 13:
                                added_score = spot2_error/100
                                spot2_error_angle = abs(df_labels.angle2\
                                                [i]-float(row_angle[l]))
                                if a == 0:
                                    features_used+=1
                                    a = 1
                                if spot2_error_angle < angle_threshold:
                                    added_score = added_score + \
                                    (spot2_error_angle/10)
                                    score = score +2+(0.12+(angle_\
                                                     threshold/10)-added_score)
                                    spot_used2 = k
                                else:
                                    spot_used2 = 0

                    if df_labels.nose_spot3[i] != 0:
                        angle_threshold = math.atan(12/\
                                                    df_labels.nose_spot3[i])
                        a = 0
                        f = 0
                        l = -1
                        for k in row_length:
                            l += 1
                            if (k != 0) & (k != spot_used) & (k != spot_used2):
                                if f == 0:
                                    features_used+=1
                                    f = 1
                                spot3_error = abs(df_labels.nose_spot3[i]-k)
                                if spot3_error < 13:
                                    added_score = spot3_error/100
                                    spot3_error_angle = abs\
                                    (df_labels.angle3[i]-float(row_angle[l]))
                                    if a == 0:
                                        features_used+=1
                                        a = 1
                                    if spot3_error_angle < \
                                    angle_threshold:
                                        added_score = added_\
                                        score+(spot3_error_angle/10)
                                        score = score +2+(0.12+\
                                        (angle_threshold/10)-added_score)
                                        spot_used3 = k
                                    else:
                                        spot_used3 = 0

            else:
                spot_used = 0
                spot_used2 = 0
                spot_used3 = 0


            if df_labels.nose_spot4[i] != 0:
                angle_threshold = math.atan(12/\
                                df_labels.nose_spot4[i])

                row_length = df_spots_length.iloc[j]
                row_angle = df_spots_angle.iloc[j]
                f = 0
                a = 0
                l = -1
                for k in row_length:
                    l += 1
                    if (k != 0) & (k!= spot_used) & \
                    (k!= spot_used2) & (k!= spot_used3):
                        if f==0:
                            features_used+=1
                            f = 1
                        spot4_error = abs(df_labels.nose_spot4[i]-k)
                        if spot4_error < 13:
                            added_score = spot4_error/100
                            spot4_error_angle = abs(df_labels.\
                                            angle4[i]-float(row_angle[l]))
                            if a == 0:
                                features_used+=1
                                a = 1
                            if spot4_error_angle < angle_threshold:
                                added_score = added_score+(spot4_error_angle/10)
                                score = score +2+(0.12+\
                                            (angle_threshold/10)-added_score)
                                spot_used4 = k
                            else:
                                spot_used4 = 0

                if df_labels.nose_spot5[i] != 0:
                    angle_threshold = math.atan(12/df_labels.nose_spot5[i])
                    a = 0
                    f = 0
                    l = -1
                    for k in row_length:
                        l += 1
                        if (k != 0) & (k!= spot_used) & \
                        (k!= spot_used2) & (k!= spot_used3) & (k != spot_used4):
                            if f == 0:
                                features_used+=1
                                f = 1
                            spot5_error = abs(df_labels.nose_spot5[i]-k)
                            if spot5_error < 13:
                                added_score = spot5_error/100
                                spot5_error_angle = abs\
                                (df_labels.angle_5[i]-float(row_angle[l]))
                                if a == 0:
                                    features_used+=1
                                    a = 1
                                if spot5_error_angle < angle_threshold:
                                    added_score = added_score+\
                                    (spot5_error_angle/10)
                                    score = score +2+(0.12+\
                                                (angle_threshold/10)-added_score)
                                    spot_used5 = k
                                else:
                                    spot_used5 = 0

                    if df_labels.nose_spot6[i] != 0:
                        angle_threshold = math.atan(12/df_labels.nose_spot6[i])
                        a = 0
                        f = 0
                        l = -1
                        for k in row_length:
                            l += 1
                            if (k != 0) & (k!= spot_used) & \
                            (k!= spot_used2) & (k!= spot_used3) &\
                            (k != spot_used4) & (k != spot_used5):
                                if f == 0:
                                    features_used+=1
                                    f = 1
                                spot6_error = abs(df_labels.nose_spot6[i]-k)
                                if spot6_error < 13:
                                    added_score = spot6_error/100
                                    spot6_error_angle = abs\
                                    (df_labels.angle6[i]-float(row_angle[l]))
                                    if a == 0:
                                        features_used+=1
                                        a = 1
                                    if spot6_error_angle < angle_threshold:
                                        added_score = added_score +(spot6_error_angle/10)
                                        score = score +2+(0.12+\
                                            (angle_threshold/10)-added_score)
                                        spot_used6 = k
                                    else:
                                        spot_used6 = 0

                        if df_labels.nose_spot7[i] != 0:
                            angle_threshold = math.atan\
                            (12/df_labels.nose_spot7[i])
                            a =0
                            f = 0
                            l = -1
                            for k in row_length:
                                l += 1
                                if (k != 0) & (k!= spot_used) & \
                                (k!= spot_used2) & (k!= spot_used6) & \
                                (k!= spot_used3) & (k != spot_used4) & (k != spot_used5):
                                    if f == 0:
                                        features_used+=1
                                        f = 1
                                    spot7_error = abs(df_labels.nose_spot7[i]-k)
                                    if spot7_error < 13:
                                        added_score = spot7_error/100
                                        spot7_error_angle = abs\
                                        (df_labels.angle7[i]-float(row_angle[l]))
                                        if a == 0:
                                            features_used+=1
                                            a = 1
                                        if spot7_error_angle < angle_threshold:
                                            added_score = added_score+(spot7_error_angle/10)
                                            score = score +2+\
                                            (0.12+(angle_threshold/10)-added_score)
                                            spot_used7 = k
                                        else:
                                            spot_used7 = 0


                            if df_labels.nose_spot8[i] != 0:
                                angle_threshold = math.atan\
                                (12/df_labels.nose_spot8[i])
                                a = 0
                                f = 0
                                l = -1
                                for k in row_length:
                                    l += 1
                                    if (k != 0) & (k!= spot_used) & \
                                    (k!= spot_used7) & (k!= spot_used2) & (k!= spot_used6)\
                                    & (k!= spot_used3) & (k != spot_used4) & (k != spot_used5):
                                        if f == 0:
                                            features_used+=1
                                            f = 1
                                        spot8_error = abs(df_labels.nose_spot8[i]-k)
                                        if spot8_error < 13:
                                            added_score = spot8_error/100
                                            spot8_error_angle = abs\
                                            (float(df_labels.angle8[i])-float(row_angle[l]))
                                            if a==0:
                                                features_used+=1
                                                a = 1
                                            if spot8_error_angle < angle_threshold:
                                                added_score = added_score+(spot8_error_angle/10)
                                                score = score +2+(0.12+\
                                                    (angle_threshold/10)-added_score)



            if score >= features_used:

                score_vector.append(score)

                #print(str(score_vector))

                label = df_labels.filename[i]
                train = df_train.filename[j]

                outF = open("Clasification_results.txt", "a")
                outF.write("'"+df_labels.filename[i]+"',"+str(df_train.filename[j])\
                           +"',"+str(score)+","+str(features_used))
                outF.write("\n")
                outF.close()

df_classy = pd.read_csv\
(r'C:\Users\ablec\Documents/MASTER\Clasification_results.txt')
df_classy.Orig_fish  = df_classy.Orig_fish.str.replace("'","")
df_classy.Suggested_match  = df_classy.Suggested_match.str.replace("'","")

correct_feature_vector = []
wrong_feature_vector = []
correctly_class = 0
wrong_class = 0
max_score = 0
max_vector = []
n_i = 0
#one_fish = 0
for i in range(0,len(df_classy.Orig_fish)):

    one_fish = 0
    for j in range(0,len(one_class_fish)):

        if one_class_fish[j] == df_classy.Orig_fish[i]:
            one_fish = 1

    if i == 0:
        max_vector.append(df_classy.score[i])

    elif df_classy.Orig_fish[i] == df_classy.Orig_fish[i-1]:
        max_vector.append(df_classy.score[i])

    elif df_classy.Orig_fish[i] != df_classy.Orig_fish[i-1]:
        max_vector_sort = sorted(max_vector, reverse = True)
```

```
#print(str(max_vector_sort))
#for j in range(0,len(one_class_fish)):
#    if df_classy.Orig_fish[i] == one_class_fish[j]:
#        max_vector_sort = max_vector_sort[0]


for k in range(n_i,i):
    label_fish = df_classy.Suggested_match[k]

    if label_fish[-1] != "1":
        #print(df_classy.Orig_fish[k])
        max_value = max_vector_sort[0]
        #print("Max_vector_sort="+str(max_vector_sort))
        if len(max_vector_sort)>1:
            second_max_value = max_vector_sort[1]

        if max_value == df_classy.score[k]:
            orig_fish = df_classy.Orig_fish[k]
            suggested_fish = df_classy.Suggested_match[k]

            if orig_fish[:-1] == suggested_fish[:-1]:
                correctly_class += 1
                print(orig_fish+'has been correctly classified as '\
                      + suggested_fish)
                outF = open("Correctly_classified.txt", "a")
                outF.write('Fish number: '+orig_fish+\
                           'had been correctly classified as '\
                           +suggested_fish+". Classified with "\
                           +str(df_classy.features_used[k])+" features.")
                outF.write("\n")
                outF.close()
                correct_feature_vector.append(df_classy.features_used[k])


            if orig_fish[:-1] != suggested_fish[:-1]:
                wrong_class += 1
                print(orig_fish+'has been INNcorrectly classified as '\
                      + suggested_fish)
                outF = open("Wrongly_classified.txt", "a")
                outF.write('Fish number: '+orig_fish+' \
                had been incorrectly classified as '+suggested_fish+\
                    ". Classified with "+str(df_classy.features_used[k])+" features.")
                outF.write("\n")
                outF.close()
                wrong_feature_vector.append(df_classy.features_used[k])

        elif second_max_value == df_classy.score[k]:
            orig_fish = df_classy.Orig_fish[k]
            suggested_fish = df_classy.Suggested_match[k]

            if orig_fish[:-1] == suggested_fish[:-1]:
                correctly_class += 1
                print(orig_fish+'has been correctly classified as '+ \
                      suggested_fish)
                outF = open("Correctly_classified.txt", "a")
```

```
                        outF.write('Fish number: '+orig_fish+' had been \
                        correctly classified as \
                        '+suggested_fish+". Classified with "+\
                                   str(df_classy.features_used[k])+" features.")
                        outF.write("\n")
                        outF.close()
                        correct_feature_vector.append(df_classy.features_used[k])

                    if orig_fish[:-1] != suggested_fish[:-1]:
                        wrong_class += 1
                        print(orig_fish+'has been INNcorrectly classified as '+\
                              suggested_fish)
                        outF = open("Wrongly_classified.txt", "a")
                        outF.write('Fish number: '+orig_fish+'\
                        had been incorrectly classified as \
                        '+suggested_fish+". Classified with "\
                                   +str(df_classy.features_used[k])+" features.")
                        outF.write("\n")
                        outF.close()
                        wrong_feature_vector.append(df_classy.features_used[k])



        max_vector = []
        max_vector.append(df_classy.score[i])
        n_i = i


correct_features_4 = []
correct_features_5 = []
correct_features_6 = []
correct_features_7 = []
correct_features_8 = []
correct_features_9 = []
correct_features_10 = []
correct_features_11 = []
correct_features_12 = []
correct_features_13 = []
correct_features_14 = []
correct_features_15 = []
correct_features_16 = []


for i in correct_feature_vector:
    if i == 4:
        correct_features_4.append(i)
    if i == 5:
        correct_features_5.append(i)
    if i == 6:
        correct_features_6.append(i)
    if i == 7:
        correct_features_7.append(i)
    if i == 8:
        correct_features_8.append(i)
    if i == 9:
        correct_features_9.append(i)
    if i == 10:
        correct_features_10.append(i)
    if i == 11:
        correct_features_11.append(i)
    if i == 12:
        correct_features_12.append(i)
    if i == 13:
        correct_features_13.append(i)
    if i == 14:
        correct_features_14.append(i)
    if i == 15:
        correct_features_15.append(i)
    if i == 16:
        correct_features_16.append(i)

incorrect_features_4 = []
incorrect_features_5 = []
incorrect_features_6 = []
incorrect_features_7 = []
incorrect_features_8 = []
incorrect_features_9 = []
incorrect_features_10 = []
incorrect_features_11 = []
incorrect_features_12 = []
incorrect_features_13 = []
incorrect_features_14 = []
incorrect_features_15 = []
incorrect_features_16 = []


for i in wrong_feature_vector:
    if i == 4:
        incorrect_features_4.append(i)
    if i == 5:
        incorrect_features_5.append(i)
    if i == 6:
        incorrect_features_6.append(i)
    if i == 7:
        incorrect_features_7.append(i)
    if i == 8:
        incorrect_features_8.append(i)
    if i == 9:
        incorrect_features_9.append(i)
    if i == 10:
        incorrect_features_10.append(i)
    if i == 11:
        incorrect_features_11.append(i)
    if i == 12:
        incorrect_features_12.append(i)
    if i == 13:
        incorrect_features_13.append(i)
    if i == 14:
        incorrect_features_14.append(i)
    if i == 15:
        incorrect_features_15.append(i)
    if i == 16:
        incorrect_features_16.append(i)





print("Number of correct classifications: "+str(correctly_class))
print("Number of incorrect classifications: "+str(wrong_class))
print("Correct classifications with 4 features: "+str(len(correct_features_4)))
print("Incorrect classifications with 4 features: "+str(len(incorrect_features_4)))
print("Correct classifications with 5 features: "+str(len(correct_features_5)))
print("Incorrect classifications with 5 features: "+str(len(incorrect_features_5)))
print("Correct classifications with 6 features: "+str(len(correct_features_6)))
print("Incorrect classifications with 6 features: "+str(len(incorrect_features_6)))
print("Correct classifications with 7 features: "+str(len(correct_features_7)))
print("Incorrect classifications with 7 features: "+str(len(incorrect_features_7)))
print("Correct classifications with 8 features: "+str(len(correct_features_8)))
print("Incorrect classifications with 8 features: "+str(len(incorrect_features_8)))
print("Correct classifications with 9 features: "+str(len(correct_features_9)))
print("Incorrect classifications with 9 features: "+str(len(incorrect_features_9)))
print("Correct classifications with 10 features: "+str(len(correct_features_10)))
print("Incorrect classifications with 10 features: "+str(len(incorrect_features_10)))
print("Correct classifications with 11 features: "+str(len(correct_features_11)))
print("Incorrect classifications with 11 features: "+str(len(incorrect_features_11)))
print("Correct classifications with 12 features: "+str(len(correct_features_12)))
print("Incorrect classifications with 12 features: "+str(len(incorrect_features_12)))
print("Correct classifications with 13 features: "+str(len(correct_features_13)))
print("Incorrect classifications with 13 features: "+str(len(incorrect_features_13)))
print("Correct classifications with 14 features: "+str(len(correct_features_14)))
print("Incorrect classifications with 14 features: "+str(len(incorrect_features_14)))
print("Correct classifications with 15 features: "+str(len(correct_features_15)))
print("Incorrect classifications with 15 features: "+str(len(incorrect_features_15)))
print("Correct classifications with 16 features: "+str(len(correct_features_16)))
print("Incorrect classifications with 16 features: "+str(len(incorrect_features_16)))
```

Listing D.10: Python example

Bibliography

[1] NRK. Over 20 prosent av oppdrettslaksen dør i merdene, March 2018. URL https://www.nrk.no/trondelag/over-20-prosent-av-oppdrettslaksen-dor-i-merdene-1.13952684.

[2] Bloomberg. Over 20 prosent av oppdrettslaksen dør i merdene, October 2018. URL https://www.bloomberg.com/news/features/2018-10-08/salmon-farmers-are-scanning-fish-faces-to-fight-killer-lice.

[3] Simone Marini Emanuela Fanelli Valerio Sbragaglia Ernesto Azzurro Joaquin Del Rio Fernandez Jacopo Aguzzi. Tracking fish abundance by underwater image recognition. *Scientific Reportsvolume 8, Article number: 13748 (2018)*, September 2018.

[4] Yi-Hao Hsiao Chaur-Chin Chen Sun-In Lin Fang-Pang Lin. Real-world underwater fish recognition and identification, using sparse representation. *Ecological Informatics*, 23:13–21, September 2014.

[5] HongweiQin Xiu Li Jian Liang Yigang Peng Changshui Zhang. Deepfish: Accurate underwater live fish recognition with a deep architecture. *Neurocomputing*, 187, april 2016.

[6] Dah-Jye Lee; Robert B. Schoenberger; Dennis Shiozawa; Xiaoqian Xu; Pengcheng Zhan. Contour matching for a fish recognition and migration-monitoring system. *Two- and Three-Dimensional Vision Systems for Inspection, Control, and Metrology II;*, Proceedings 5606, 2004.

[7] Robert B. Fisher Yun-Heh Chen-Burger, Gayathri Nadarajan. Detecting ,tracking and counting fish in low quality unconstrained underwater videos. 2008.

[8] Sanjeev Kumar Harpreet Kaur. Face recognition techniques: Classification and comparisons. *International Journal of Information Technology and Knowledge Management*, Volume 5(2):361–363, July-December 2012.

[9] Rabia Jafri Hamid R. Arabnia. A survey of face recognition techniques. *Journal of Information Processing Systems*, Vol. 5(No. 2), June 2009.

[10] T. Kanade. Picture processing system by computer complex and recognition of human faces. *Kyoto University, Japan*, 1973.

[11] R. Brunelli and T. Poggio. Face recognition: features versus templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15(NO 10), 1993.

[12] J. Ghosn I. J. Cox and P. N. Yianilos. Featurebased face recognition using mixture-distance. *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1996.

[13] Jørgensen et. al. Judging a salmon by its spots: environmental variation is the primary determinant of spot patterns in salmo salar. 2018.

[14] Stien et al. Consistent melanophore spot patterns allow long-term individual recognition of atlantic salmon salmo salar. *Journal of Fish Biology*, 2017. URL https://www.ncbi.nlm.nih.gov/pubmed/29094766.

[15] Jason Ku Ali Harakeh Steven L. Waslander. In defense of classical image processing: Fast depth completion on the cpu. *arXiv:1802.00036v1*, january 2018. Mechanical and Mechatronics Engineering Department.

[16] Mithi. Vehicle detection with hog and linear svm, 2017. URL https://medium.com/@mithi/vehicles-tracking-with-hog-and-linear-svm-c9f27eaf521a.

[17] Paul Viola Michael Jones. Robust real-time face detection. *International Journal of Computer Vision 57(2), 137–154, 2004*, July 2003.

[18] S.M. Juds. *Photoelectric Sensors and Controls*. Mechanical Engineering (Book 63). Marcel Dekker, Inc, 1988.

[19] Koppal S.J. Lambertian reflectance. *Computer Vision*, 2014. URL https://link.springer.com/referenceworkentry/10.1007%2F978-0-387-31439-6_534.

[20] Maria Petrou Costas Petrou. *IMAGE PROCESSING The Fundamentals*. John Wiley Sons LTD, 2nd edition, 2010.

[21] doxygen. Image thresholding. URL https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html.

[22] Bikramjot Singh Hanzra. Adaptive thresholding. January 2015. URL http://hanzratech.in/2015/01/21/adaptive-thresholding.html.

[23] Rajeev Raizada T. Florian Jaeger Dave F. Kleinschmidt. Supervised and unsupervised learning in phonetic adaptation. *Department of Brain and Cognitive Sciences, Department of Computer Science, and Department of Linguistics*, 07 2015.

[24] David Arthur Sergei Vassilvitskii. k-means++: The advantages of careful seeding. *SODA '07 Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035, 2007.

[25] E. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics*, 21(3):768–769, 1965.

[26] Lihong Wang Jinglei Liu Jianpeng Qi, Yanwei Yu. K*-means: An effective and efficient k-means clustering algorithm. *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pages 242–249, 2016.

[27] T. Lindeberg. Detecting salient blob-like image structures and their scales with a scale-space primal sketch: A method for focus-of-attention. *International Journal of Computer Vision, Volume 11, no. 3, page 283-318*, 1993.

[28] T. Lindeberg. Image matching using generalized scale-space interest points. *Journal of Mathematical Imaging and Vision, Volume 52, number 1*, 2015.

[29] H. Kong et. al. A generalized laplacian of gaussian filter for blob detection and its applications. *IEEE Transactions on Cybernetics, Volume 43, page 1719-1733*, 2013.

[30] T. Lindeberg. Feature tracking with automatic selection of spatial scales. *Computer Vision and Image Understanding, Volume 71, page 385-392*, 1998.

[31] SATYA MALLICK. Blob detection using opencv (python, c++), 2015. URL `https://www.learnopencv.com/blob-detection-using-opencv-python-c/`.

[32] doxygen. cv::simpleblobdetector class reference, 2018. URL `https://docs.opencv.org/3.4.2/d0/d7a/classcv_1_1SimpleBlobDetector.html`.

[33] UN. Undata. URL `http://data.un.org/`.

[34] IPPC. Ar5 climate change 2014: Mitigation of climate change, 2014. URL `https://www.ipcc.ch/report/ar5/wg3/`.

[35] National Geographics. Can the ocean feed a growing world?, August 2018. URL `https://www.nationalgeographic.com/environment/2018/08/news-fisheries-aquaculture-food-security/`.

[36] EY Norway. The norwegian aquaculture analysis 2018. 2018.

[37] EPA. Global greenhouse gas emissions data. URL `https://www.epa.gov/ghgemissions/global-greenhouse-gas-emissions-data`.

[38] MOWI. Home page. URL https://www.mowi.com/.

[39] Statistisk Sentral Byraa. Akvakultur. URL https://www.ssb.no/jord-skog-jakt-og-fiskeri/statistikker/fiskeoppdrett/aar.

[40] Oppdrettsnæringen er en tikkende bombe, April 2019. URL https://www.dagbladet.no/kultur/oppdrettsnaeringen-er-en-tikkende-bombe/70961082.

[41] IMR. Tema: Laks, 2019. URL https://www.hi.no/hi/temasider/arter/laks.