

INTELLIGENT OBJECT RECOGNITION IN UNDER WATER IMAGE PROCESSING

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Electronics & Communication Engineering

by

VALLAM SREE NAGANJALI (Reg. No. 37130427)

SUMASRI M(Reg. No. 37130399)



**DEPARTMENT OF ELECTRONICS & COMMUNICATION
ENGINEERING**

SCHOOL OF ELECTRICAL & ELECTRONICS ENGINEERING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **VALLAM SREE NAGANJALI (Reg. No. 37130427)** and **SUMASRI M (Reg. No. 37130399)** who carried out the project entitled **"INTELLIGENT OBJECT RECOGNITION IN UNDER WATER IMAGE PROCESSING"** under our supervision from December 2020 to March 2021.

INTERNAL GUIDE

Dr. G.Sundari

HEAD OF THE DEPARTMENT

Dr.T Ravi

Submitted for Viva voce Examination held on **17-04-2021**

Internal Examiner

External Examiner

DECLARATION

We, **VALLAM SREE NAGANJALI (Reg. No. 37130427)** and **SUMASRI M(Reg. No. 37130399)** hereby declare that the Project Report entitled “**INTELLIGENT OBJECT RECOGNITION IN UNDER WATER IMAGE PROCESSING**” done by us under the guidance of Dr.G.Sundari, at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of **Bachelor of Engineering** degree in **Electronics and Communication Engineering**.

1.

V. SreeNaganjali

2.

Sumasri.M

DATE:17-04-21

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATES

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. N.M.NANDITHA, Dean**, School of Electrical and Electronics and **Dr.T.RAVI, Head of the Department**, Department of Electronics & Communication Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. G.SUNDARI**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the Department of **Electronics and Communication Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Live fish recognition is one of the most crucial elements of fisheries survey applications where vast amount of data are rapidly acquired. Different from general scenarios, challenges to underwater image recognition are posted by poor image quality, uncontrolled objects and environment, as well as difficulty in acquiring representative samples. Also, most existing feature extraction techniques are hindered from automation due to involving human supervision. Toward this end, we propose an underwater fish recognition framework that consists of a fully unsupervised feature learning technique and an error-resilient classifier. Object parts are initialized based on saliency and relaxation labeling to match objects parts correctly. A non-rigid part model is then learned based on fitness, separation and discrimination criteria. For the classifier, an unsupervised clustering approach generates a binary class hierarchy, where each node is a classifier. To exploit information from ambiguous images, the notion of partial classification is introduced to assign coarse labels by optimizing the “benefit” of indecision made by the classifier. Experiments show that the proposed framework achieves high accuracy on both public and self-collected underwater fish images with high uncertainty and class imbalance.

TABLE OF CONTENT

CHAPTER NO	TITLE	PAGE NO
	ABSTRACT	V
	LIST OF FIGURES	VII
	LIST OF TABLES	VIII
1	INTRODUCTION	
	1.1 Problem Description	3
	1.2 Morphological difference between salmon & sea trout	4
	1.3 Basic image analysis	5
	1.3.1 Filtering	6
	1.3.2 Segmentation	7
	1.3.3 Video and motion analysis	8
	1.3.4 Feature extraction & representation	8
	1.3.5 Mathematical morphology	9
2	PREVIOUS WORK	10
3	DATASET	13
4	METHOD	18
	4.1 Segmentation & Filtering	18
	4.2 Separating salmon from sea trout	21
5	RESULTS	26
	5.1 Separating wild from farmed salmon	26
	5.2 Separating salmon from sea trout	28
6	DISCUSSION	31
7	CONCLUSION AND FUTURE WORK	37

LIST OF FIGURES

FIGURE NO	TITLE	PAGE NO
1	Sketch of trap that selectively catches farmed salmon	1
2	The adipose fin	2
3	Morphological structures of importance in this study	3
4	Salmon	4
5	Sea trout	4
6	Noisy video sequence	5
7	Segmentation of yellow and green peas	6
8	Convex hull of white object on black background	8
9	Two different sized structuring elements, the origin is marked with an x	9
10	Original image and result after dilation operator using structuring	10
11	VAKI infrared sensor frame	10
12	Examples of silhouettes captured by the VAKI infrared sensor frame	11
13	Image acquisition setup used by Zion et al.	11
14	Outlined fish in 70 mm wide	12
15	Image acquisition setup used by White et al	13
16	Flatfish divided into cells	13
17	Image acquisition setup	14
18	Six typical frames showing six different fish	14
19	Nine typical frames from one video, showing a farmed salmon	16
20	Nine typical frames from one video, showing a sea trout	17
21	Difference image	18
22	Edge image E_{diff}	19
23	Edge image E_{all}	19
24	Edge image after filtering	20
25	Illustration of parameters	22
26	Examples of upper contour between dorsal and caudal fins	22
27	Illustration of parameters $s, h_{left}, h_{right}, k_{min}, k_{left}$ and k_{right}	23
28	height of fish at dorsal fin	24
29	Illustration of manual process	25
30	Percentage of correctly classified contours per fish	26
31	Number of contours found per fish where the dorsal and caudal fin were found.	27

LIST OF TABLES

TABLE NO	TITLE	PAGE NO
1	Number of videos analysed and removed	15
2	Number of contours per fish	27
3	Resulting ratios for the ratio r_f .	28
4	Resulting ratios for the ratio r_d .	29
5	Classification results for the ratio r_f .	29
6	Classification results for the ratio r_d .	30
7	Weighted accuracies and split points for r_f	30
8	Weighted accuracies and split points for r_d .	30

CHAPTER 1

INTRODUCTION

IMAGE PROCESSING and analysis techniques for underwater cameras have drawn increasing attention since they enable a non-extractive and non-lethal approach to fisheries survey. For instance, by using a combination of cameras and mid-water trawl, known as the Cam-trawl, fish schools are sampled by capturing images or videos while they pass through the trawl. The camera-based sampling approach not only conserves depleted fish stocks but also provides an effective way to sample a greater diversity of marine animals. This approach, however, generates vast amounts of data very rapidly. An automatic image processing system is thus critically required to make such a sampling approach practical. Toward this end, we have developed techniques that analyze the collected data by automatic object segmentation, size estimation, counting and tracking. Based on this, an automatic camera-based fisheries survey system can be realized by developing a reliable species identification algorithm that allows for monitoring the species composite and assessing fish stocks as well as the ecosystem. While object recognition in various contexts has been well investigated in image processing and computer vision communities, there exist fundamental challenges to identifying live fish in an unconstrained natural habitat. Like most underwater imagery scenarios, one challenge is posted by the low image quality caused by fast attenuation of light in the water, poor control over illumination, the ubiquitous organic debris, etc. While capturing images for freely-swimming fish, there is a high uncertainty in many of the data due to low image quality, non-lateral fish views or curved body shapes. Aquatic ecosystems are today highly affected by human activity, such as overfishing, hydro power stations, oxygen-free bottoms and eutrophication. One effect of hydro power stations is that the natural migration to their natal river streams for salmon (*Salmo salar* L., 1758), sea trout (*Salmo trutta* L., 1758) and other species have been obstructed. Therefore, fish ladders have been built at the power stations to reduce this particular problem. In Sweden 2012, about 2.9 million smolt of salmon and sea trout were farmed and released to compensate for the obstruction of the natural fish migration [14]. The compensation farmed salmon have a reasonable genetic variation, but it competes with the wild fish and will dilute the natural genetic pool. Only wild salmon should be allowed to enter the rivers to help preserve the natural ecosystem.

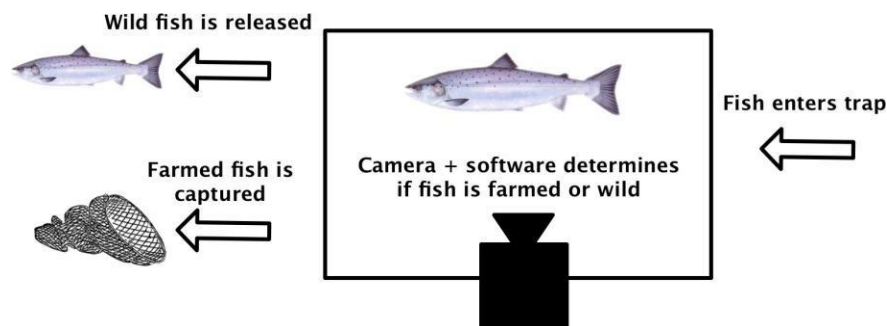


Figure 1: Sketch of trap that selectively catches farmed salmon.

When fish is farmed in net cages diseases can spread and medicine can only be administered to the fish inside the farm. If a sick fish escapes it can repeatedly reinfect the other healthy fish inside the farm since the fish often stays close to the farm after it has escaped. Traps that only catch the farmed fish could be installed outside the farm, see Figure 1 for a sketch of such a trap. However, currently no such traps exist. These types of traps could also be installed at the fish ladders near the hydro power stations to prevent the farmed salmon to enter up stream.

In Sweden farmed salmon is caught selectively, in order to separate the farmed from wild salmon the adipose fin can be cut off on the farmed salmon when they are young, see Figure 2 for comparison. This method is inexpensive and fast. At the present time the farmed salmon can be separated manually by landing the fish and looking for an existing adipose fin, but it is costly and stressful for the fish.

Traditionally the adipose fin was believed to be a non-functional trait. However, more recent studies show that the fin has extensive nervous tissue [3, 13]. Lack of the fin makes the fish less attractive to its mates and decreases their chance of mating which promotes the natural genetic variance [8].

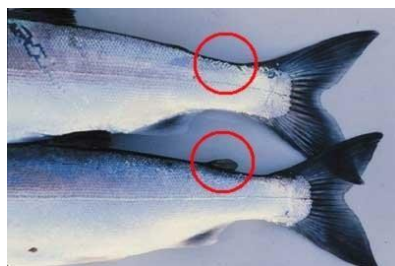


Figure 2: The fish at the top is farmed and has its adipose fin cut off. The fish at the bottom is wild and still has the fin left, the adipose fin is marked by a circle.

At some of the Swedish fish ladders, cameras are installed to record the passing of the fish. These videos are manually inspected to gather information about the biological state of the river, such as how many fish pass through and what species they are. In this project some of these videos have been used to analyse the fish passing up into the river using image analysis.

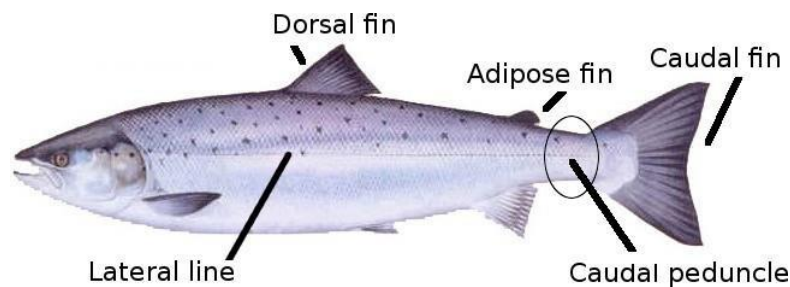


Figure 3: Morphological structures of importance in this study.

The videos contain a side view of wild salmon, farmed salmon and sea trout that has climbed the ladder and is entering the river. The videos have low resolution and contain constantly moving sea grass in front and behind the fish.

The first part of the project was to segment the fish and filter out the sea grass. Then, it was examined if it was possible to automatically separate the farmed from wild fish by searching for an existing adipose fin. Figure 3 shows common parts of the fish which are mentioned repeatedly in this project. In order to locate the adipose fin, first the dorsal and caudal fins were located and the area between the two bigger fins was extracted. Thereafter it was examined if it was possible to separate two common and similar looking fish species, in this case salmon from sea trout. Two morphometric measurements were compared by how well they can separate the species. This information could give rivers interesting information about fish entering the rivers.

1.1 Problem description

Using image analysis on a video stream from an underwater camera are the two following problems possible to solve?

Automatically separate farmed from wild salmon by searching for an existing adipose fin.

- Separate salmon from sea trout using visual morphometric measurements.

1.2 Morphological differences between salmon and sea trout



Figure 4: Salmon (*Salmo salar* L., 1758).



Figure 5: Sea trout (*Salmo trutta* L., 1758).

Both salmon and sea trout belong to the genus *Salmo* in the family *Salmonidae*, see Figures 4 and 5 for a visual comparison. Separating these species in low resolution images is not a trivial task even for an human expert, see Figure 18 for a comparison of the species in this dataset. There are, however, useful characteristic differences in size, shape and visual texture. Some notable visual differences consist of [6]:

The caudal peduncle is narrower in salmon than in sea trout. From a side view the ratio between the height of the caudal peduncle and the tallest middle part is different between the species, compare Figure 4 and Figure 5

Salmon commonly has fewer black spots, especially below the lateral line

- I. Salmon is generally larger than sea trout
- II. The posterior part of the caudal fin is more concave in salmon and more flat in sea trout
- III. Salmon and sea trout can hybridize and produce progeny with characters from both parents, but these will not be covered in this project.

1.3 Basic image analysis

When analysing images it is often necessary to divide the process into smaller problems that can be solved individually. There is no standard process for all images and each problem often requires a customized process. Often the image is filtered, the interesting objects are extracted, features are extracted from the objects and the features are finally classified or analysed. Below are descriptions of some common elements in image analysis which are included in this project.

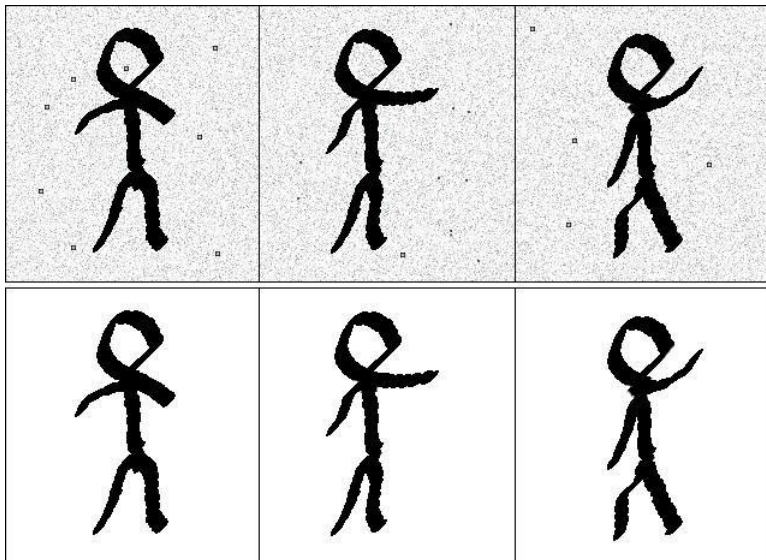


Figure 6: Noisy video sequence on the top and result after temporal filtering below.

1.3.1 Filtering

If the image is noisy, reducing the noise is often the first step, this can be done using many different techniques depending on the type of image and noise. Temporal filtering is an effective way to reduce noise in videos. The basic idea is that the pixel values are relatively constant between the consecutive frames. However, this assumption can only be made where no motion is present. For example, if a pixel at the same location has the values 5, 4, 255, 4, 3 it is then likely that the value 255 is noise. To filter the noise, the mean between the preceding and following values can be used to replace the suspected noisy value, the result could then be 5, 4, 4, 4, 5. If the motion of a object can be gauged, it is possible to use temporal filtering even when motion is present, by tracking the object and translating the history of the pixel values. Figure 6 shows a three frame video sequence before and after temporal filtering. There could also be unwanted objects like sea grass that might occlude the objects of interest. Such objects needs to be identified and excluded from the analysis process. Figure 18 show parts of the fish which are occluded by sea grass. Naturally if an interesting object is completely occluded nothing can be done to restore it.



Figure 7: Segmentation of yellow and green peas (*Pisum sativum* L., Fabaceae) using colour alone.

1.3.2 Segmentation

Image segmentation is the method of dividing an image into parts which represents the meaningful real-world objects or areas in the image. Only the interesting parts or areas are extracted which reduces the amount of information which needs to be analysed. One of the most basic techniques is called intensity thresholding where all pixels above or below a certain value are kept and the rest are discarded. The result is a binary image. This technique can be useful when a bright object is placed over a darker background or the opposite. Objects can also be segmented using features such as colour as illustrated in Figure 7 or by using clustering algorithms like K-means [10] to find similar parts of an image.

1.3.3 Video and motion analysis

A video is a time sequence of images and the number of images per second is called the frame rate. A higher frame rate means higher temporal resolution, thus smaller changes in motion can be captured than with a lower frame rate. There are many techniques to segment the foreground from background. The simplest technique is to subtract each frame from the previous one. The result is the change between the frames which can be seen in Figure 21. Many methods build a model for each pixel and then calculate how likely it is that pixel is part of the background or foreground. A Mixture of Gaussian background model simulates each background pixel by a mixture of a number of Gaussian distributions. The weights of the mixture represent the time proportions that those colours stay in the scene. The probable background colours are the ones which stay longer and more static [12].

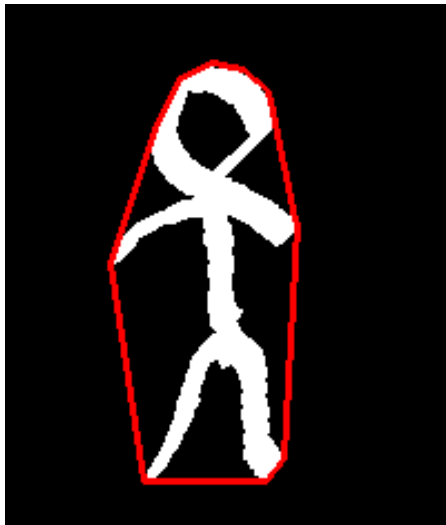


Figure 8: Convex hull of white object on black background.

1.3.4 Feature extraction and representation

When the interesting object or area has been segmented it can be described in various ways. Different objects usually need different representations. For example, the area might be a good feature when analysing very simple objects. The convex hull is widely used in image analysis and it is the smallest convex set that contain the object, see Figure 8. The difference between the convex hull of an object and the object can for example be used to describe the shape of objects. In a 2D square grid, an object is said to be four-connected if it is a group of pixels which have at least one horizontal or one vertical neighbour. When the pixels of an object also has a diagonal neighbour it is said to be eight-connected.

When the frequency information is interesting the Fourier transform [16] can be used on contours such as describing the shape of leaves, then it is often called Fourier descriptors.

Generally only the edge information is stored when analysing shapes and many methods to extract edges have been developed. One of the most simple and common methods is to calculate the gradient magnitude of the image in some direction, usually in both the X and Y-direction and then compute the overall magnitude. Another common method to locate the edges is to use the Canny edge detector [5]. It begins by smoothing the image, calculates the gradient magnitude in a number of directions of the smoothed image. Then thinning the edge responses from the gradient using non-maximal suppression. Finally, tracking the edges using two thresholds to ensure that noisy edges are not broken up into multiple fragments. The result of the Canny edge detector on the image shown in Figure 21 can be seen in Figure 22.

1.3.5 Mathematical morphology

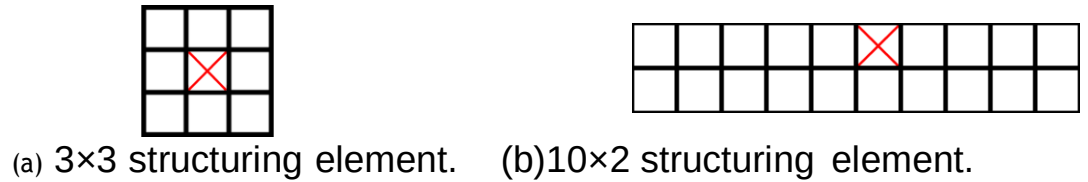


Figure 9: Two different sized structuring elements, the origin is marked with an x.

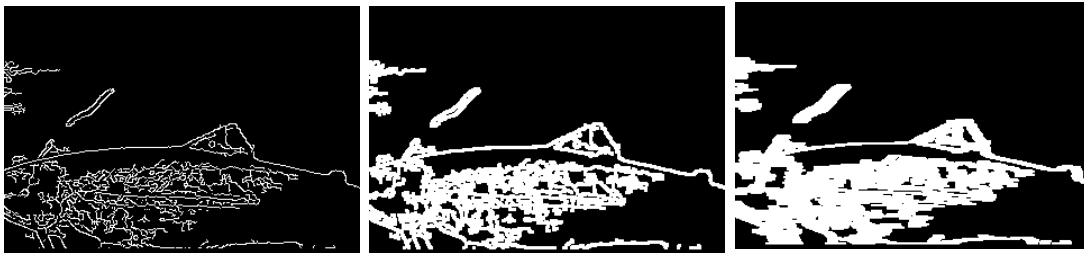


Figure 10: Original image and result after dilation operator using structuring elements shown in Figure 9. Note that the shapes in Figure 10c are expanded mostly in the horizontal direction.

Mathematical morphology is widely used in image analysis, it is based on non-linear operations on object shapes. It is often used when dealing with binary or grey-scale shapes in images and when low computational cost is important. Morphological operations can be used for many tasks including noise reduction, segmentation and object representation. A morphological operation involves the interaction between the image and a small set called a structuring element. A structuring element has a origin often located in the centre of symmetric structuring elements, like Figure 9a. Two of the elementary operations are called erosion and dilation. In binary images dilation works by moving the structuring element over the image. If at least one pixel in the structuring element overlaps a foreground pixel the origin at the current location is set as foreground. Erosion is similar to dilation, it is calculated by moving the structuring element over the image. However, the pixel at the current location is only set as foreground if every pixel in the structuring element overlaps with foreground pixels in the input image. Figure 10 shows a binary image and the result after dilation with the two structuring elements displayed in Figure 9. Figure 9b shows the structuring element used in this project for filtering sea grass.

CHAPTER 2

PREVIOUS WORK

There is currently no automatic way of selectively catching only farmed salmon. There has been some previous work which has successfully been able to determine species from the shape of the fish.

Cadieux et al. [4] separated bowfin (*Amia calva* L., 1766), copper redhorse (*Moxostoma hubbsi* Legendre, 1952), largemouth bass (*Micropterus salmoides* Lacépède, 1802), northern pike (*Esox lucius* L., 1758) and walleye (*Sander vitreus* Mitchill, 1818) using silhouettes acquired by an infrared sensor frame made by the Icelandic company VAKI, see Figure 11. They used features such as invariant moments, Fourier descriptors, area, perimeter, length and convex hull area of the silhouettes, see Figure 12. The features were classified using a multi-classifier which consisted of a combination of a Bayes maximum likelihood classifier, learning vector quantification classifier and a one-class-one-network (OCON) neural network. The majority vote from the classifiers then decided the final species. The highest classification result for all species in this project was 70.8% and 73% when limited to three species.

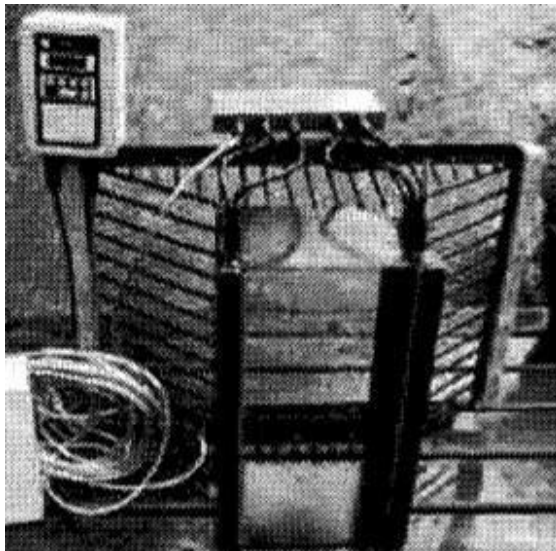


Figure 11: VAKI infrared sensor frame used by Cadieux et al. [4].

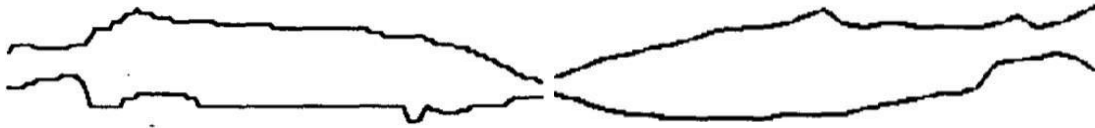


Figure 12: Examples of silhouettes captured by the VAKI infrared sensor frame.

Zion et al. [17] separated the Common carp (*Cyprinus carpio* L., 1758), St. Peters's fish (*Oreochromis spilurus* Günther, 1894) and Grey mullet (*Mugil cephalus* L., 1758) using image analysis in real-time from underwater images. They built a 70 mm wide channel for the image acquisition in order to minimize the variation in distance between the fish and camera, see Figure 13. They calculated several size and orientation invariant features from the silhouettes using polar coordinates. Examples of outlined fish are shown in Figure 14. A minimum Mahalanobis distance classifier was used and average fish species classification accuracies were 95.9%, 92.8% and 93.9% for the St. Peter's fish, Grey mullet and the Common carp respectively.



Figure 13: Image acquisition setup used by Zion et al. [17].

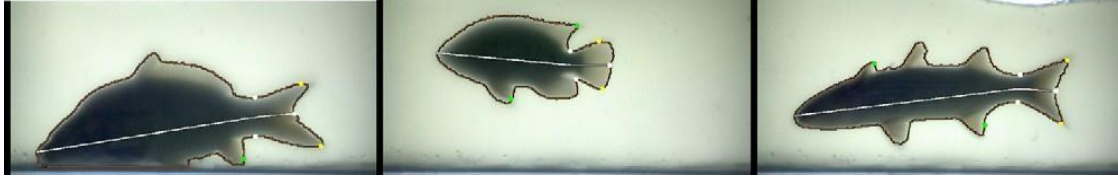


Figure 14: Outlined fish in 70 mm wide channel [17].

White et al. [15] Developed a computer vision system to recognise seven species of dead flatfish with 98.8% accuracy. The species were: long rough dab (*Hippoglossoides platessoides* Fabricius, 1780), sole (*Solea solea* (L., 1758)), lemon sole (*Microstomus kitt* Walbaum, 1792), plaice (*Pleuronectes platessa* L., 1758), golden redfish (*Sebastes norvegicus* Ascanius, 1772), deepwater redfish (*Sebastes mentella* Travin, 1951) and flounder (*Platichthys flesus* L., 1758). The dead fish was put on a backlit conveyor belt and passed under further illumination and the camera, see Figure 15. They used colour thresholding for segmenting the fish. The image of the fish was divided into roughly equal sized cells, see Figure 16. They used moments of the shape and the average color in the cells as features to describe the fish. The length was measured accurately by fitting lines through the centre of the fish.

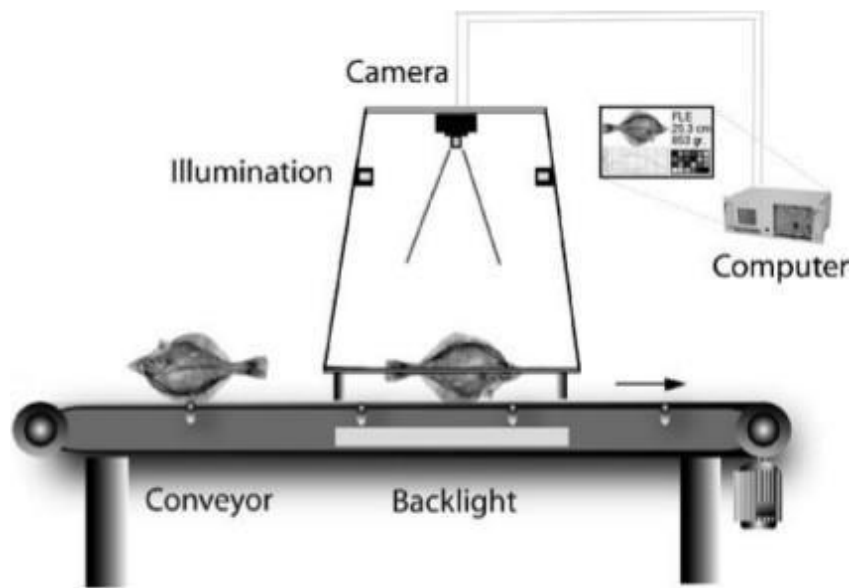


Figure 15: Image acquisition setup used by White et al. [15].

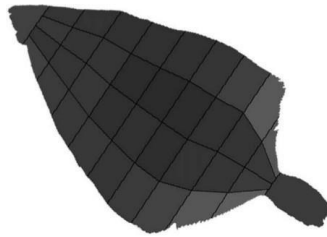


Figure 16: Flatfish divided into cells [15].

CHAPTER 3

DATASET

The dataset consists of videos which are collected from an existing VAKI system but with a custom camera which captures infrared light better. There is also a light source mounted near the camera which projects light on the front of the fish which can be seen in Figure 17. This is opposite to Zion et al. and White et al. which has the light source behind the fish [15, 17]. First, the fish pass through a VAKI Biomest Scanner Frame. The frame works by using beams of infrared light and sensors that record when the beam is broken.

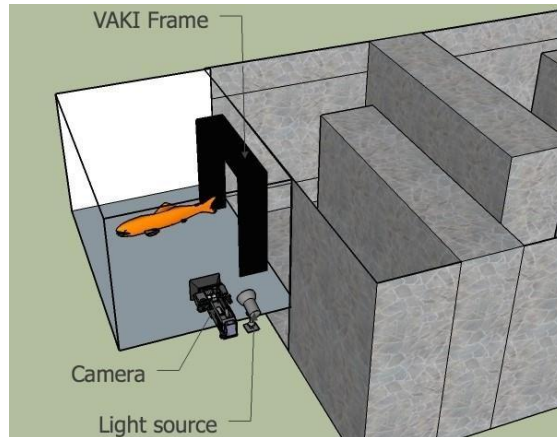


Figure 17: Image acquisition setup, the fish enters the imaging chamber from the right side after climbing the ladder and exits into the river on the left side.

When a fish is detected by the frame it triggers the camera to record the next 14 seconds. The dataset consists of these videos. This means that every recording starts with a fish in the right part of the video. Figure 17 illustrates the setup.



Farmed salmon

Wild salmon

Sea trout

Figure 18: Six typical frames showing six different fish, the two to the left are farmed salmon, the two in the middle are wild salmon and the two to the right are sea trout. Notice that the sea grass occluding part of the fish in some frames.

The videos were encoded with size 320 240 and a frame rate of 29.97 fps. The frames were converted to eight-bit greyscale representation. All videos contained a 20 pixel high black border with a time stamp in the lower part of image, this area was masked out resulting in a effective size of 320 220. The videos were tagged with date, species and the presence of the adipose fin.

Some of the videos contained more than one fish simultaneously, these videos were removed from the dataset. Other videos contained more than one fish but not visible simultaneously, these were kept. The analysed videos were collected between 2012-06-28 to 2012-09-15. In total 53 videos were analysed and four were removed. All videos were captured in the river Vindelälven in Sweden.

Species	No. video s analysed	No. video s removed
Salmon farmed	19	1
Salmon wild	17	3
Sea trout	17	0

Table 1: Number of videos analysed and removed.

Figure 18 shows the three different types of fish which are included in the dataset. Figures 19 and 20 shows typical frames from the videos. Note that Figure 19 only shows a short period in the beginning of the 14 second long video.

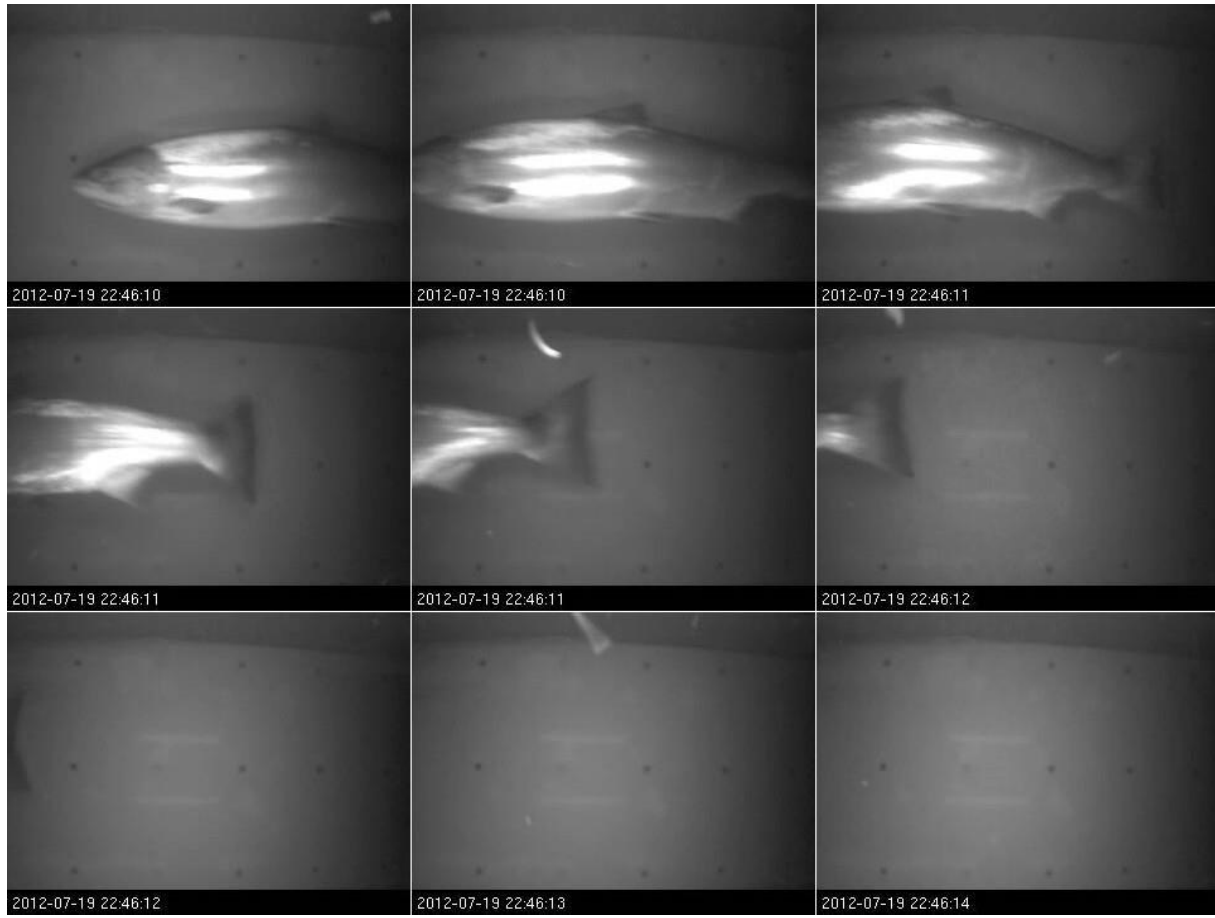


Figure 19: Nine typical frames from one video, showing a farmed salmon passing swiftly from right to left. There is approximately a third of a second between each image. Note that the sequence starts at the top left.

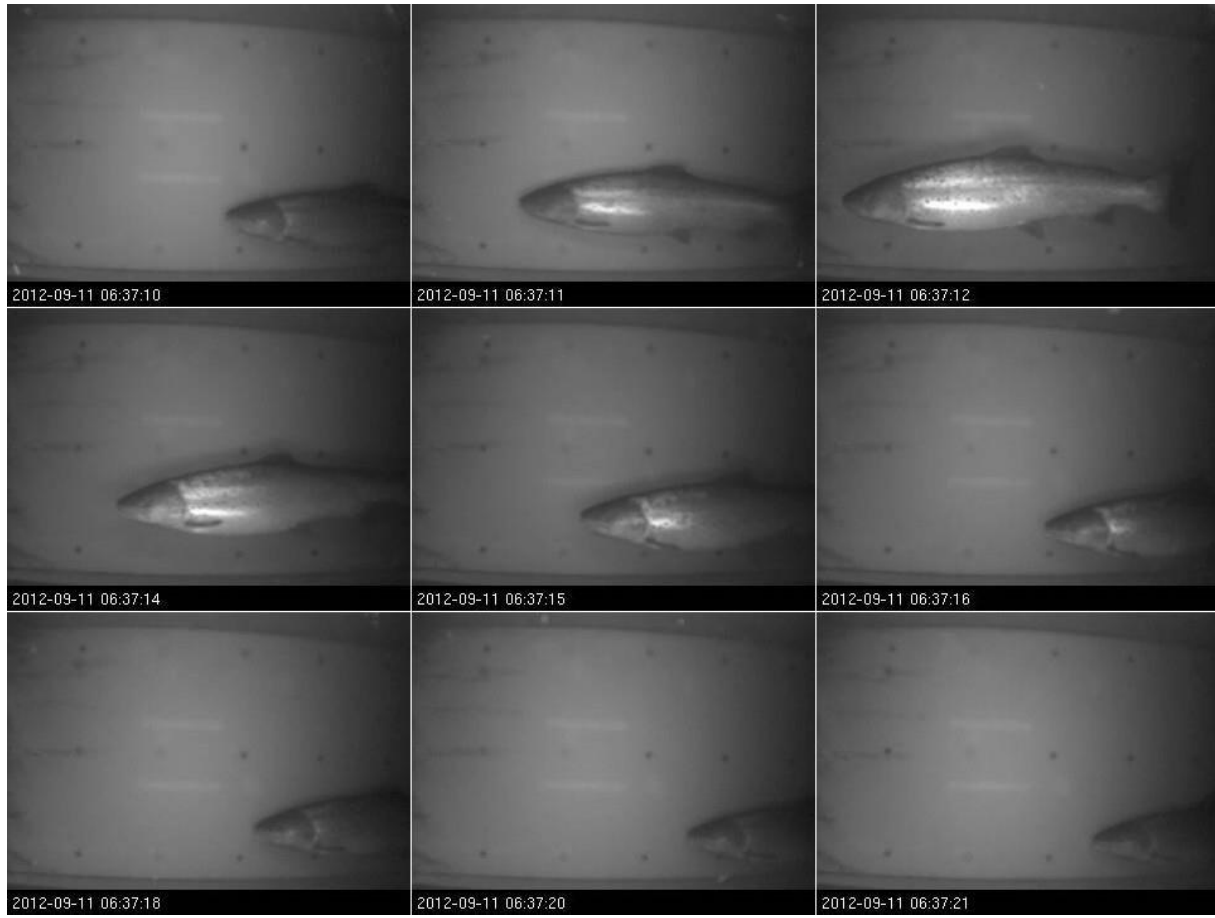


Figure 20: Nine typical frames from one video, showing a sea trout swimming back and forth. The fish never swims past the left side of the frame. There is approximately one to two seconds between each image. Note that the sequence starts at the top left.

CHAPTER 4

METHOD

The first step was to segment the fish excluding the sea grass. The contours were extracted and analysed in order to separate farmed from wild salmon. This was done by extracting the upper contour, localising the area between the dorsal and caudal fins and determining if the fish had an adipose fin or not. In addition manual measurements were made on the salmon and sea trout in order to see if it is possible to separate the species using morphometric measurements.

The videos were analysed off-line but the method was developed with real-time requirements in mind. All computations were fast and simple to compute, resulting in above real-time requirements on a single 3.2GHz Intel Sandy Bridge CPU core. The programming language Python was used together with the library OpenCV [1] for most of the image analysis parts of the project.

4.1 Segmentation and filtering

The first step to locate the fish was to segment the motion if a fish was found, the contours of the fish needed to be extracted. The goal was to only find the edges belonging to the fish. When a good enough contour was found it could be analysed. First, a difference image I_{diff} was acquired by computing the absolute difference between each frame and the mean image of the last 30 images, see Figure 21. In eight of the videos the mean image contained fish which lead to a suboptimal segmentation for those videos. Still this technique gave a better segmentation in this dataset than using a mixture of Gaussian background model [1] to segment the motion.

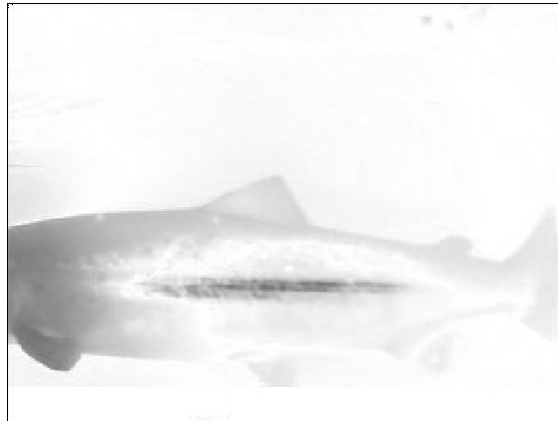


Figure 21: Difference image I_{diff} , notice the low contrast at the bottom right parts. Some parts of the fish have the same intensity as the background.

An edge image E_{diff} was calculated from the difference image I_{diff} using the Canny edge detector [5]. The result is illustrated in Figure 22.

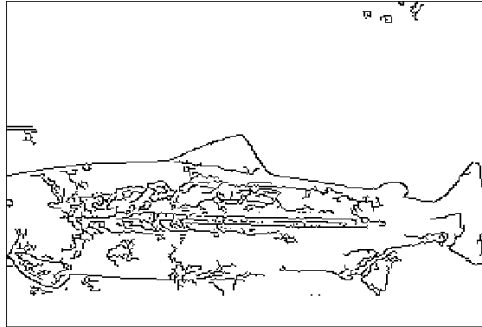


Figure 22: Edge image E_{diff} , containing mostly edges from the fish.

The edges from the unprocessed image were extracted using the same method and were added to E_{diff} to create a new image denoted E_{all} , see Figure 23. This step added edges belonging to the fish but also unwanted contours from the background. In E_{diff} the bottom contour of the fish often had holes in it and in E_{all} the contour was better connected.

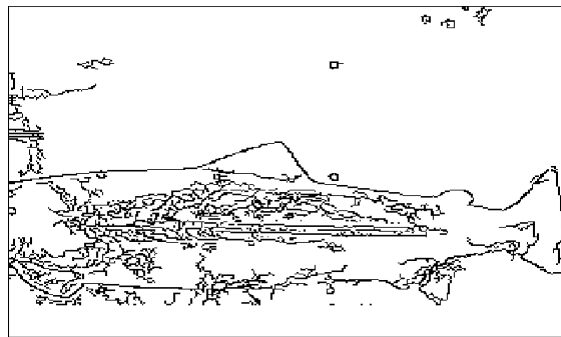


Figure 23: Edge image E_{all} , containing edges from both the background and foreground.

E_{diff} contained mostly edges from the fish. These edges were incomplete since I_{diff} did not segment the fish completely because some parts of the fish had the same intensity as the background

The fish moved less than the sea grass in between the frames, also the contours from E_{diff} were mostly belonging to the fish. Therefore, some temporal information of the contours from E_{diff} was used to reduce the amount of sea grass in E_{all} . E_{diff} was dilated to create E_{tmp} using a structuring element of size 10 2 with the center position at (5,1), see Figure 9b. The connected components were extracted from E_{all} and for each 8-connected component

- All connected components where the maximum length of its bounding box was shorter than 75 pixels were discarded.
- The number of pixels which overlapped between the two previous E_{tmp} and the connected component were calculated. The component was discarded if less than half the pixels did not overlap.
- The combined resulting components contain more edges belonging to the fish than E_{diff} and fewer edges from the background than E_{all} , see Figure 24.

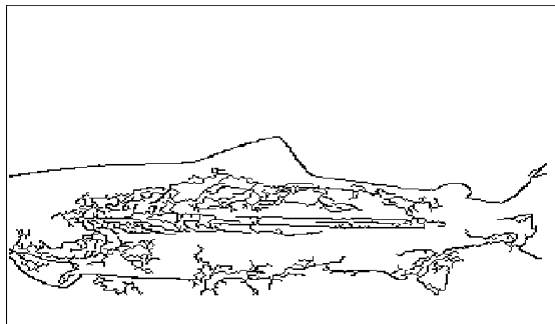


Figure 24: Edge image after filtering, some noise was removed but also the posterior parts of the caudal fin.

The size of the structuring element used in E_{tmp} was chosen to allow for more horizontal movement and less vertical movement, since the fish swims primarily in the horizontal direction and the sea grass moved more randomly

Separating wild from farmed salmon

To separate wild from farmed salmon the presence of an adipose fin was used as the key character. The upper contours were generally quite noisy but resembled a upper contour of a fish. First Fourier descriptors [16] were used as a feature together with various machine learning techniques in order to determine if the contour contain an adipose fin, this did not give any good results. Instead a more specialized solution with low computational cost was developed.

The 1D upper contour was acquired by using the top projection of the contour, the highest vertical-value for each horizontal-value. The dorsal fin and caudal fin were located by finding local peaks in the top contour using the function `signal.find_peaks_cwt` [7]. The function finds the peaks by convolving the contour using differently wide wavelets and locating local maxima with high signal to noise ratio. For simplifying the description of the method the function `avg` is defined as:

$$\text{avg}(X, x, n) = \frac{1}{n} \sum_{i=x}^{x+n-1} X[i]$$

where X is an array, x is the index offset and n specifies the width. The location of the peak is denoted p , the contour is denoted C and the gradient of C is denoted C' . `avg` is applied on the arrays C or C' and the index x is often selected close to the peak p . Since $x + i$ is indexing an array $x + i$ must always be larger or equal to zero and smaller than the length of C . When this is not the case n is set to go from $\max(0, x + n)$ when $x < 0$ and up to $\min(\text{length}(C), x + n)$ when $x + n > \text{length}(C)$. The variable w controls the width of the peak descriptor. In this dataset the width of the frames were 320 pixels and $w = 10$ was used below, this needs to be adjusted according to the size of the input frames. For each peak p the following values were calculated:

- h - Relative height of the peak, height at the peak subtracted by the median of the contour

$$h = C(p) - \text{median}(C)$$

k_{left} - Average gradient to the left side of the peak

$$k_{\text{left}} = \text{avg}(C', p - w, w)$$

k_{right} - Average gradient to the right side of the peak $k_{\text{right}} = \text{avg}(C', p + w, w)$

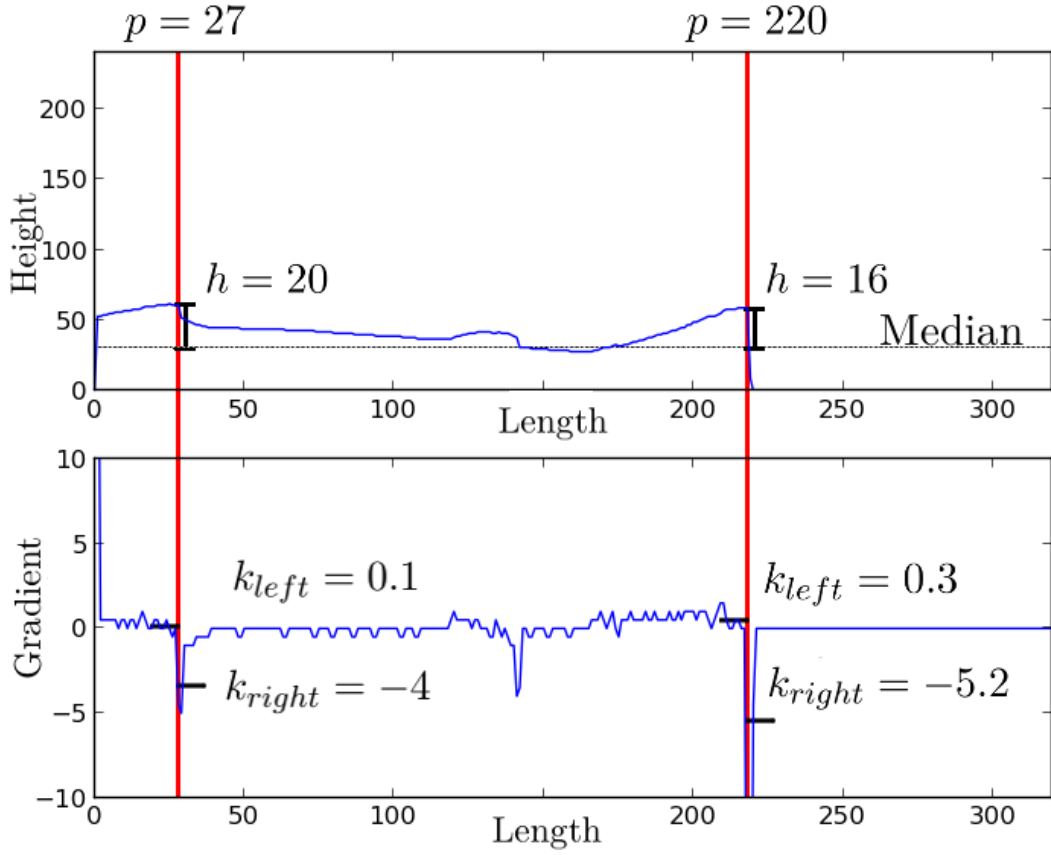
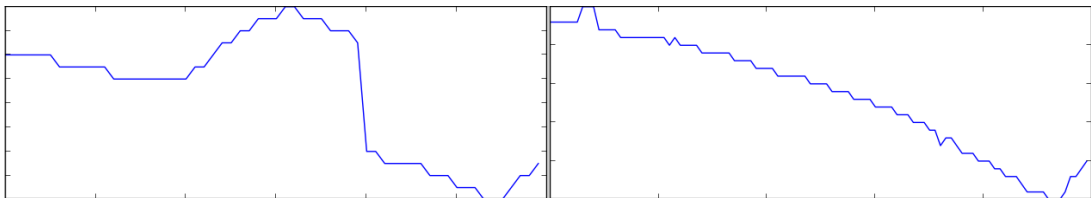


Figure 25: Illustration of above parameters. A good contour of a wild salmon is displayed on the top and corresponding gradient below. The contour shows valid peaks at the dorsal and caudal fins marked with vertical lines. Notice the adipose fin between the two peaks.

An example upper contour, two valid peaks and the corresponding parameters h , k_{left} and k_{right} are shown in Figure 25. The dorsal fin was found when $k_{left} > 0$ and $k_{right} < 0$ and $h > 10$. The caudal fin was found when the rightmost peak fulfilled that $k_{left} > 0.5$ and $k_{right} < 1.0$ and $h > 0$. If two fins were found according to the above criteria, the contour between the two fins was extracted by moving the peaks towards each other until the absolute gradient around the peak was less than 0.1. The result was the contour between the dorsal and caudal fins, two examples are shown in Figure 26.



(a) Wild salmon with adipose fin. (b) Farmed salmon without adipose fin.

Figure 26: Excellent examples of upper contour between dorsal and caudal fins.

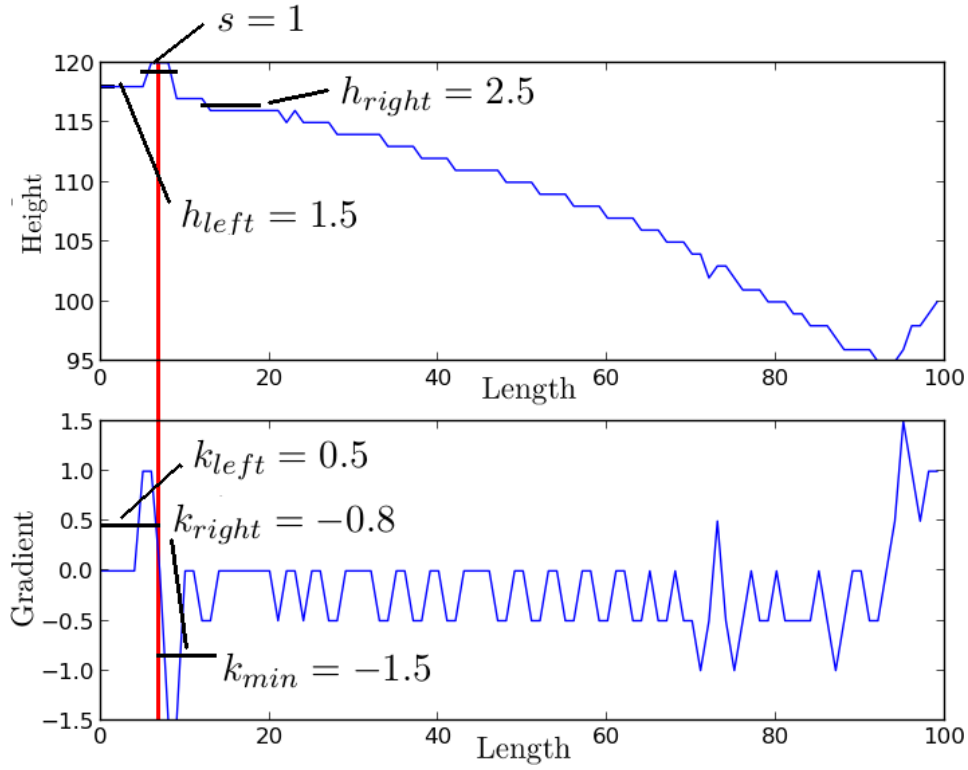


Figure 27: Illustration of parameters s , h_{left} , h_{right} , k_{min} , k_{left} and k_{right} . Example of area between the dorsal and caudal fins on the top and corresponding gradient below. Shows a invalid peak at a non-existing adipose fin

The adipose fin is found when $|s| < 2$ and $h_{left} > 0$ and $h_{right} > 5$ and $k_{min} < -1.5$ and $k_{left} > 0$ and $k_{right} < 0$. An example contour and a illustration of the above parameters in the area between the fins are shown in Figure 27. In this example all parameters except h_{right} pass the above test, the adipose fin must have a higher slope on the right side in order to be classified as a valid fin. It can also be seen that when calculating h_{left} the number of averaging sample points on the contour is reduced since the peak is close to the left border ($x_n < 0$).

Because many contours were found for each fish, each contour was classified as either farmed or wild using the method described above. The final classification was done by choosing the class which got the highest vote.

4.2 Separating salmon from sea trout

The species in the previous work had vastly different shapes. This is not the case for salmon and sea trout. Therefore, more specific morphometric differences were investigated to separate the species.

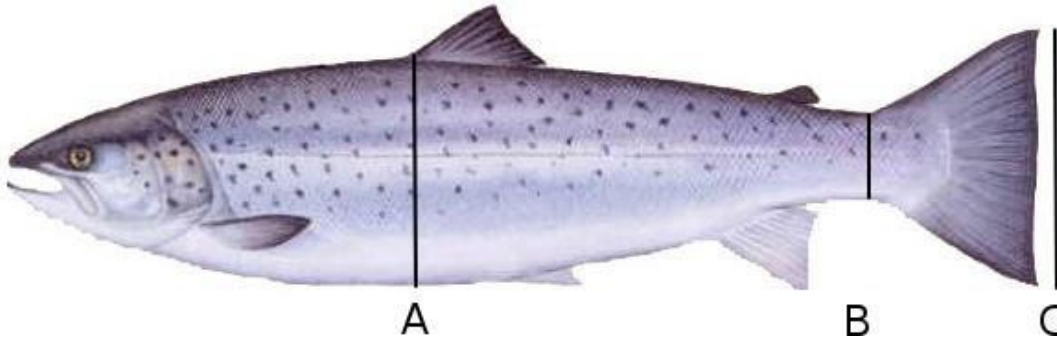


Figure 28: A: height of fish at dorsal fin, B: caudal peduncle height, C: caudal fin height

Two morphometric differences between salmon and sea trout are:

Height of caudal fin divided by the height of caudal peduncle: $r_f = \frac{C}{B}$, see

Figure 28

Height of the fish at the dorsal fin divided by the height of the caudal peduncle:

$r_d = \frac{A}{B}$, see Figure 28

An automatic method for measuring the ratios was first developed. The method used both the lower and upper contours of the fish, the position of the dorsal (A) and caudal (C) fins were located in the same way as in the method used when localising the adipose fin. The caudal peduncle (B) was located by looking for a minimum in the upper contour to the left of the caudal fin. The heights were calculated by subtracting the value of the upper contour by the value of the lower contour. The quality of the extracted lower contours were significantly inferior to the upper contours due to lower contrast between the fish and the background at the lower part of the fish. Therefore this automatic approach was abandoned for the available dataset.

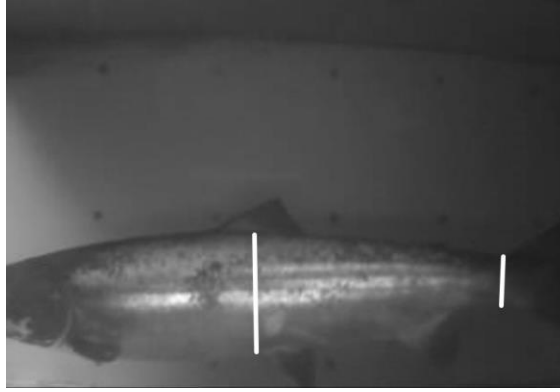


Figure 29: Illustration of manual process when measuring A and B .

Instead all ratios were measured manually to evaluate if the two ratios r_f and r_d could help discriminate between the species. The manual process consisted of drawing lines located at A , B and C for frames where the respective points were visible in the fish. In Figure 29 lines are drawn at A and B , it was not possible to measure C in this frame since the end of the caudal fin is not completely visible.

The heights were calculated by subtracting the value of the upper contour by the value of the lower contour. The quality of the extracted lower contours were significantly inferior to the upper contours due to lower contrast between the fish and the background at the lower part of the fish. Therefore this automatic approach was abandoned for the available dataset.

Objects appear larger when closer to the camera and smaller when further away due to the perspective. Because the caudal fin is used for propulsion it is moved back and forth thus changing both ratios. The measurements could also be combined into a single scalar using simple mathematical functions such as min, max, mean and median. This might give a more precise value of the ratio and a better classification result since there was an uncertainty of measured height, both due to the perspective and the low contrast.

CHAPTER 5

RESULTS

Using this dataset it was possible to separate wild and farmed salmon by searching for presence or not of the adipose fin. However, a requirement for the presented method was that both the dorsal and caudal fins must have been visible at least once in the video.

It was also possible to separate salmon from sea trout using the manual measurements of the the ratio r_f and to a lesser extent also with r_d .

5.1 Separating wild from farmed salmon

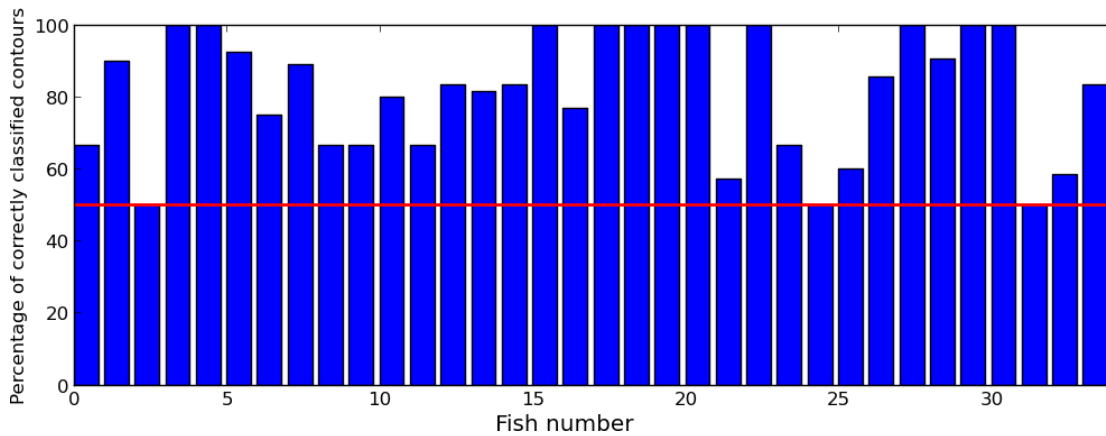


Figure 30: Percentage of correctly classified contours per fish where the dorsal and caudal fin were found, the horizontal line marks 50%.

Contours from 39 fish were extracted from the nineteen salmon videos. In 34 of these fish both the dorsal and caudal fin were visible, the percentage of correctly classified contours of these 34 is shown in Figure 30. 31 of the 34 fish had more than half of the contours correctly classified, thus the fish could be correctly classified. The other three have exactly half of the contours correctly classified. Eleven fish had all contours correctly classified. On average 81% of the contours for each fish were correctly classified, the standard deviation was 17%. The average, the number of found contours was 84 where the both the dorsal and caudal was found. The number of found contours per fish varied between twelve and 233, see Figure 31. The correlation coefficient between the percentage of correctly classified contours and the number of contours found per fish was $r = 0.048$. thus there was no dependence between the two and more contours are not necessary better for determining the presence of an adipose fin.

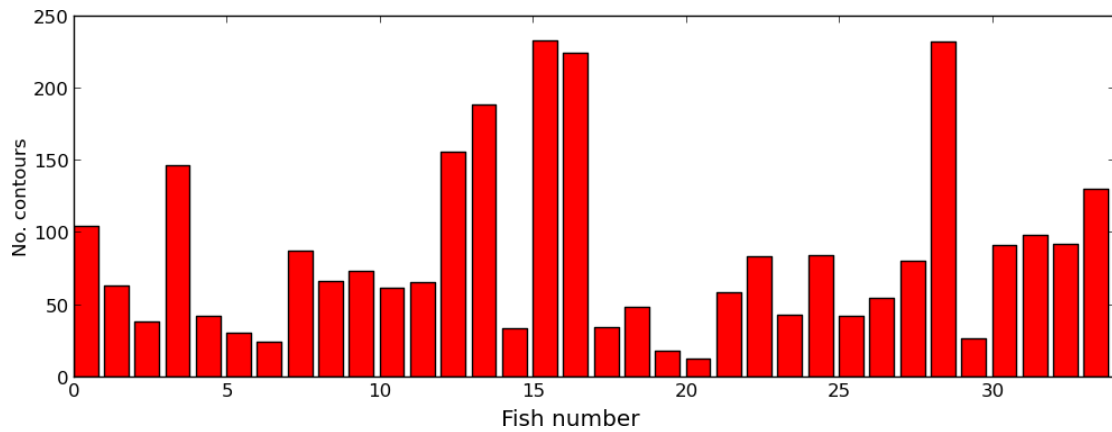


Figure 31: Number of contours found per fish where the dorsal and caudal fin were found.

The five fish for which the dorsal and caudal fins were not found had a mean of 150. This is significantly higher than 84 which was the mean for the fish where both fins were found. These fish were mostly stationary at the right part of the frame similar to the fish shown in Figure 20.

Fish No.	No. contours
4	338
20	39
23	53
24	206
30	115

Table 2: Number of contours per fish where both the dorsal and caudal fin were not found.

5.2 Separating salmon from sea trout

Using the ratio r_f the salmon had a mean of 3.51 with a standard deviation of 0.48, the sea trout had a mean of 3.0 with a standard deviation 0.3. A linear discriminator at 3.25 separated the classes with 75% accuracy.

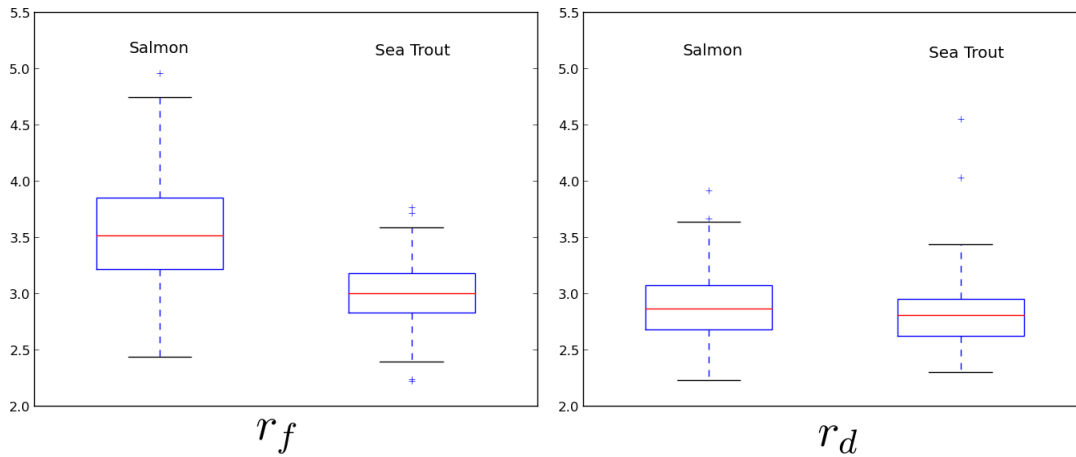


Figure 32: Boxplot of the ratios r_f to the left and r_d to the right for salmon and sea trout. Sea trout has a lower ratio than salmon, a significant of overlap in r_d and some in r_f . Note that there are fewer measurements for sea trout.

Species	No. samples	Mean	Std
Salmon	171	3.51	0.48
Sea trout	113	3.00	0.30

Table 3: Resulting ratios for the ratio r_f .

The ratio r_d had a mean of 2.88 with a standard deviation of 0.29 for salmon and a mean of 2.84 with a standard deviation of 0.38 for sea trout. A linear discriminator at 2.86 separated the classes with 53.6% accuracy. This is a significant overlap between the two species and not as good a discriminator in this dataset as r_f .

Species	No. samples	Mean	Std
Salmon	143	2.88	0.29
Sea trout	51	2.84	0.38

Table 4: Resulting ratios for the ratio r_d .

To improve the separation between the classes the measurements were combined into a single scalar using the following mathematical functions: min, max, mean and median. Each fish was then classified once using the scalar value. The split point was found by sliding a point in small increments between the smallest and highest value of the ratios, the selected point was the one which gave the highest separation between the species. The separation was defined as the number of correctly classified ratios for both classes divided by the total number of ratios for both classes. Since there are more ratios of salmon than sea trout this method will return a biased split point. The accuracies for each species are shown in Tables 5 and 6. Using the ratio r_f and the median gave the highest classification score of 89.1% accuracy, see Table 5. For the ratio r_d median, mean and max gave the same result which was lower than for all measurements. The salmon class dominated the classification using this method for selecting the split point, see Table 6.

Function	Split point	Accuracy	Sea trout	Salmon
Median	3.22	89.1%	100%	84.4%
Mean	3.1	87%	78.6%	90.6%
Max	3.58	82.6%	78.6%	84.4%
Min	2.59	80.4%	57.1%	90.6%
None	3.18	76.4%	75.2%	77.2%

Table 5: Classification results for the ratio r_f .

Function	Split point	Accuracy	Sea trout	Salmon
Median	2	70%	0%	100%
Mean	2	70%	0%	100%
Max	2	70%	0%	100%
Min	2.36	72%	20%	94.3%
None	2.36	74.2%	5.9%	98.6%

Table 6: Classification results for the ratio r_d .

To investigate the bias of the split point, a new point was selected by maximizing the average accuracy for both species, called weighted accuracy below. Instead the mean of the accuracy for both classes were used, for example an accuracy of 100 percent for salmon and 50 percent for sea trout would give a weighted result of 75 percent. This gives more weight to each measurement in the smaller class, in this case sea trout. The result can be seen in Tables 7 and 8.

Function	Split point	Weighted accuracy	Sea trout	Salmon
Median	3.22	92.2%	100%	84.4%
Mean	3.31	90.6%	100%	81.3%
Max	3.77	85.9%	100%	71.9%
Min	2.82	77.7%	92.9%	62.5%
None	3.3	76.8%	84%	69.6%

Table 7: Weighted accuracies and corresponding split points for the ratio r_f .

Function	Split point	Weighted accuracy	Sea trout	Salmon
Median	2.84	60.4%	66.6%	54.2%
Mean	2.73	60.4%	46.6%	74.2%

Max	3.06	58.6%	60%	% 57.1
Min	2.38	59.0%	26.6%	% 94.2
None	2.96	57.6%	80.3%	% 34.9

Table 8: Weighted accuracies and corresponding split points for the ratio r_d .

CHAPTER 6

DISCUSSION

The presented method does not segment the fish completely reliable. Because of the strong direct lighting the fish can be both brighter and darker than the background. Some parts of the fish absorbs the light more than the bright background while the shiny parts reflects more of the light than the background. This makes segmentation difficult since the location between the dark and light parts of the fish will have the same intensity as the background. An ideal scenario would be to capture the silhouette using back illumination similar to the work done by Zion et al. and White et al. [15, 17].

The method was designed to be used on continuous video stream except for the morphometric measurements which were done manually. The border between the fish and background was sometimes fuzzy which introduced measurement errors. Especially the length at B (caudal peduncle) was affected, but since B was included in both r_f and r_d there should be minimal bias when comparing the two ratios. Using higher quality data an automatic method should be relative straight-forward to implement.

When separating farmed from wild salmon there were five fish which did not swim into the image far enough for the caudal fin to be visible. These fish could not be classified as farmed or wild since the method needs to localize both the dorsal and caudal fins. In a real application this will not be a problem since the fish needs to swim past the camera in order to be sorted.

At the fish ladders near the hydro power stations in northern Sweden there are mainly three species of fish which enters the rivers; salmon, sea trout and common whitefish (*Coregonus lavaretus* L., 1758). It has not been investigated if it was possible to discriminate the common whitefish from salmon and sea trout since no videos of the first was included in the dataset. If a trap would be placed in the sea where more species could enter the trap the software must be able to separate salmon from all other species which could enter the trap.

The gradient was calculated from the raw extracted upper contour, this could lead to high spikes when the contour is very discontinuous. Although the gradient was averaged over multiple values the parameters that determines if a fin exist or not needs to set with this in mind. By removing values or low-pass filtering the contour and fitting a spline the gradient would be more well behaved. It has not been examined if this affects the classification results.

The variables presented in the method are specific to this dataset and needs to be adapted to each dataset. Optimisation of the variables might lead to better results, it might also be possible to set them adaptively depending on external factors like the size, speed or illumination of the fish.

SOFTWARE MODULES:

VIDEO PROCESSING TOOL: OPENCV

PROGRAM: PYTHON

PYTHON (PROGRAMMING LANGUAGE)

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. An interpreted language, Python has a design philosophy which emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax which allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale. Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library. Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation

HISTORY

Python was conceived in the late 1980s, and its implementation began in December 1989 [28] by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Netherlands as a successor to the ABC language (itself inspired by SETL) capable of exception handling and interfacing with the operating system Amoeba. Van Rossum is Python's principal author, and his continuing central role in deciding the direction of Python is reflected in the title given to him by the Python community, benevolent dictator for life (BDFL). About the origin of Python, Van Rossum wrote in 1996: Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office ... would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus). Python 2.0 was released on 16 October 2000 and had many major new features, including a cycle-detecting garbage collector and support for Unicode. With this release the development process was changed and became more transparent and community-backed. Python 3.0 (which early in its development was commonly referred to as Python 3000 or py3k), a major, backwards-incompatible release, was released on 3 December 2008 after a long period of testing. Many of its major features have been backported to the backwards-compatible Python 2.6.x and 2.7.x version series.

speculated was in response to Python 2.7's planned end-of-life but Google cited performance under concurrent workloads as their only motivation. Features and philosophy Python is a multi-paradigm programming language: object-oriented programming and structured programming are fully supported, and many language features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other 6/13/2017 Python (programming language) paradigms are supported via extensions, including design by contract and logic programming. Python uses dynamic typing and a mix of reference counting and a cycle-detecting garbage collector for memory management. An important feature of Python is dynamic name resolution (late binding), which binds method and variable names during program execution. The design of Python offers some support for functional programming in the Lisp tradition. The language has `map()`, `reduce()` and `filter()` functions; list comprehensions, dictionaries, and sets; and generator expressions. The standard library has two modules (`itertools` and `functools`) that implement functional tools borrowed from Haskell and Standard ML. The core philosophy of the language is summarized by the document The Zen of Python (PEP 20), which includes aphorisms such as: Beautiful is better than ugly Explicit is better than implicit Simple is better than complex Complex is better than complicated Readability counts Rather than requiring all desired functionality to be built into the language's core, Python was designed to be highly extensible. Python can also be embedded in existing applications that need a programmable interface. This design of a small core language with a large standard library and an easily extensible interpreter was intended by Van Rossum from the start because of his frustrations with ABC, which espoused the opposite mindset. While offering choice in coding methodology, the Python philosophy rejects exuberant syntax, such as in Perl, in favor of a sparser, less-cluttered grammar. As Alex Martelli put it: "To describe something as clever is not considered a compliment in the Python culture." Python's philosophy rejects the Perl "there is more than one way to do it" approach to language design in favor of "there should be one—and preferably only one—obvious way to do it". Python's developers strive to avoid premature optimization, and moreover, reject patches to non-critical parts of CPython that would offer a marginal increase in speed at the cost of clarity. When speed is important, a Python programmer can move time-critical functions to extension modules written in languages such as C, or try using PyPy, a just-in-time compiler. Cython is also available, which translates a Python script into C and makes direct C-level API calls into the Python interpreter. An important goal of Python's developers is making it fun to use. This is reflected in the origin of the name, which comes from Monty Python, and in an occasionally playful approach to tutorials and reference materials, such as using examples that refer to spam and eggs instead of the standard `foo` and `bar`. A common neologism in the Python community is *pythonic*, which can have a wide range of meanings related to program style. To say that code is *pythonic* is to say that it uses Python idioms well, that it is natural or shows fluency in the language, that it conforms with Python's minimalist philosophy and emphasis on readability. In contrast, code that is difficult to understand or reads like a rough transcription from another programming language is called *unpythonic*. Users and admirers of Python, especially those considered knowledgeable or experienced, are often referred to as Pythonists, Pythonistas, and Pythoneers. The third slice parameter, called *step* or *stride*, allows elements to be skipped and reversed. Slice indexes may be omitted, for example `a[:]` returns a copy of the entire list. Each element of a slice is a shallow copy. In Python, a distinction between expressions and statements is rigidly enforced, in contrast to languages such as Common Lisp,

Scheme, or Ruby. This leads to duplicating some functionality. For example: List comprehensions vs. for-loops Conditional expressions vs. if blocks The eval() vs. exec() built-in functions (in Python 2, exec is a statement); the former is for expressions, the latter is for statements. 6/13/2017 Python (programming language) - Statements cannot be a part of an expression, so list and other comprehensions or lambda expressions, all being expressions, cannot contain statements. A particular case of this is that an assignment statement such as `a = 1` cannot form part of the conditional expression of a conditional statement. This has the advantage of avoiding a classic C error of mistaking an assignment operator `=` for an equality operator `==` in conditions: `if (c = 1) { ... }` is syntactically valid (but probably unintended) C code but `if c = 1: ...` causes a syntax error in Python. Methods Methods on objects are functions attached to the object's class; the syntax `instance.method(argument)` is, for normal methods and functions, syntactic sugar for `Class.method(instance, argument)`. Python methods have an explicit `self` parameter to access instance data, in contrast to the implicit `self` (or `this`) in some other object-oriented programming languages (e.g., C++, Java, Objective-C, or Ruby). Typing Python uses duck typing and has typed objects but untyped variable names. Type constraints are not checked at compile time; rather, operations on an object may fail, signifying that the given object is not of a suitable type. Despite being dynamically typed, Python is strongly typed, forbidding operations that are not well-defined (for example, adding a number to a string) rather than silently attempting to make sense of them. Python allows programmers to define their own types using classes, which are most often used for objectoriented programming. New instances of classes are constructed by calling the class (for example, `SpamClass()` or `EggsClass()`), and the classes are instances of the metaclass type (itself an instance of itself), allowing meta programming and reflection. Before version 3.0, Python had two kinds of classes: old-style and new-style. The syntax of both styles is the same, the difference being whether the class object is inherited from, directly or indirectly (all new-style classes inherit from `object` and are instances of `type`). In versions of Python 2 from Python 2.2 onwards, both kinds of classes can be used. Old-style classes were eliminated in Python 3.0. The long term plan is to support gradual typing and as of Python 3.5, the syntax of the language allows specifying static types but they are not checked in the default implementation, CPython. An experimental optional static type checker named `mypy` supports compile-time type checking.

Mathematics Python has the usual C arithmetic operators (`+`, `-`, `*`, `/`, `%`). It also has `**` for exponentiation, e.g. `5**3 == 125` and `9**0.5 == 3.0`, and a new matrix multiply `@` operator is included in version 3.5. [71] Additionally, it has a unary operator (`~`), which essentially inverts all the bytes of its one argument. For integers, this means `~x == -x-1`. Other operators include bitwise shift operators `x << y`, which shifts `x` to the left `y` places, the same as `x*(2**y)`, and `x >> y`, which shifts `x` to the right `y` places, the same as `x/(2**y)`. [73] The behavior of division has changed significantly over time: Python 2.1 and earlier use the C division behavior. The `/` operator is integer division if both operands are integers, and floating-point division otherwise. Integer division rounds towards 0, e.g. `7 / 3 == 2` and `-7 / 3 == -2`. Python 2.2 changes integer division to round towards negative infinity, e.g. `7 / 3 == 2` and `-7 / 3 == -3`. The floor division `//` operator is introduced. So `7 // 3 == 2`, `-7 // 3 == -3`, `7.5 // 3 == 2.0` and 6/13/2017 Python (programming language) - Wikipedia [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language)) 8/19 `-7.5 // 3 == -3.0`. Adding `from_futue_import division` causes a module to use Python 3.0 rules for

division (see next). Python 3.0 changes / to be always floating-point division. In Python terms, the pre-3.0 / is classic division, the version-3.0 / is real division, and // is floor division. Rounding towards negative infinity, though different from most languages, adds consistency. For instance, it means that the equation $(a+b) // b == a // b + 1$ is always true. It also means that the equation $b * (a // b) + a \% b == a$ is valid for both positive and negative values of a. However, maintaining the validity of this equation means that while the result of $a \% b$ is, as expected, in the half-open interval $[0, b)$, where b is a positive integer, it has to lie in the interval $(b, 0]$ when b is negative. Python provides a round function for rounding a float to the nearest integer. For tie-breaking, versions before 3 use round-away-from-zero: round(0.5) is 1.0, round(-0.5) is -1.0. Python 3 uses round to even: round(1.5) is 2, round(2.5) is 2. Python allows boolean expressions with multiple equality relations in a manner that is consistent with general use in mathematics. For example, the expression $a < b < c$ tests whether a is less than b and b is less than c. C-derived languages interpret this expression differently: in C, the expression would first evaluate $a < b$, resulting in 0 or 1, and that result would then be compared with c. Python has extensive built-in support for arbitrary precision arithmetic. Integers are transparently switched from the machine-supported maximum fixed-precision (usually 32 or 64 bits), belonging to the python type int, to arbitrary precision, belonging to the python type long, where needed. The latter have an "L" suffix in their textual representation. (In Python 3, the distinction between the int and long types was eliminated; this behavior is now entirely contained by the int class.) The Decimal type/class in module decimal (since version 2.4) provides decimal floating point numbers to arbitrary precision and several rounding modes. The Fraction type in module fractions (since version 2.6) provides arbitrary precision for rational numbers. Due to Python's extensive mathematics library, and the third-party library NumPy which further extends the native capabilities, it is frequently used as a scientific scripting language to aid in problems such as numerical data processing and manipulation. Libraries Python has a large standard library, commonly cited as one of Python's greatest strengths, providing tools suited to many tasks. This is deliberate and has been described as a "batteries included" Python philosophy.

OPENCV

OpenCV (*Open source computer vision*) is a [library of programming functions](#) mainly aimed at real-time [computer vision](#). Originally developed by [Intel](#), it was later supported by [Willow Garage](#) then Itseez (which was later acquired by Intel). The library is [cross-platform](#) and free for use under the [open-source BSD license](#).

OpenCV supports the [deep learning](#) frameworks [TensorFlow](#), [Torch/PyTorch](#) and [Caffe](#)

HISTORY:

Officially launched in 1999 the OpenCV project was initially an [Intel Research](#) initiative to advance [CPU](#)-intensive applications, part of a series of projects including [real-time ray tracing](#) and [3D display](#) walls. The main contributors to the project included a

number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also [optimized code](#) for basic vision infrastructure. No more [reinventing the wheel](#).
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making [portable](#), performance-optimized code available for free – with a license that did not require code to be open or free itself.

The first alpha version of OpenCV was released to the public at the [IEEE Conference on Computer Vision and Pattern Recognition](#) in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the [C++](#) interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months¹ and development is now done by an independent Russian team supported by commercial corporations.

In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

On May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV.

PROGRAMMING LANGUAGE

OpenCV is written in [C++](#) and its primary interface is in C++, but it still retains a less comprehensive though extensive older [C interface](#). There are bindings in [Python](#), [Java](#) and [MATLAB/OCTAVE](#). The API for these interfaces can be found in the online documentation. Wrappers in other languages such as [C#](#), [Perl](#), [Ch](#), [Haskell](#), and [Ruby](#) have been developed to encourage adoption by a wider audience.

Since version 3.4, OpenCV.js is a [JavaScript](#) binding for selected subset of OpenCV functions for the web platform.

All of the new developments and algorithms in OpenCV are now developed in the C++ interface.

HARDWARE ACCELERATION

If the library finds Intel's [Integrated Performance Primitives](#) on the system, it will use these proprietary optimized routines to accelerate itself.

A [CUDA](#)-based [GPU](#) interface has been in progress since September 2010.

CHAPTER 7

CONCLUSION AND FUTURE WORK

It was possible to separate farmed from wild salmon by searching for presence or not of the adipose fin. The only requirement was that both the dorsal and caudal fin needed to be visible. It was also possible to separate salmon from sea trout by looking at the ratio between the height of the caudal fin and caudal peduncle. The ratio of the height of the fish at the dorsal fin and the height of the caudal peduncle was found to be inferior for separating the two species in this dataset. This method used a manual process to do the measurements, this process needs to be automated. The methods presented should also be evaluated on videos which use background illumination instead of front illumination in order to get sharper contours.

There can be large biological variations in the appearance of the fish for different rivers. The presented results should be evaluated on fish from another river, especially the species classification.

In the future texture information could also be used to help separate the species. Some experiments were performed using Local Binary Patterns (LBP) [11] and a Random Forest classifier [2] to automatically separate the two species in this dataset with promising results.

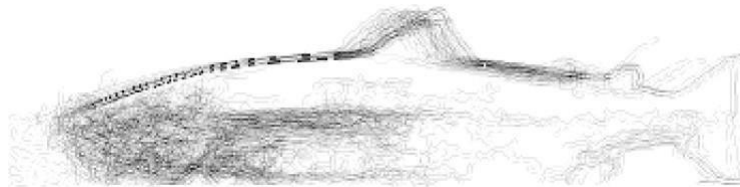


Figure 33: Multiple contours of one fish from this dataset stitched together into a single image.

Image registration techniques could also be used to stitch together one image for each fish, figure 33 shows an example from this dataset. A single image of each fish makes it easier to locate the fins if the fish is large and does not fit into one frame. Length or other measurements are also easier to acquire if depth information is available as well.

Finally, cutting the adipose fin is inexpensive but cruel to the fish, alternative techniques such as electronic tagging should be investigated as a option to separate the wild and farmed fish.

REFERENCES:

- [1] D. G. Hankin and G. H. Reeves, "Estimating total fish abundance and total habitat area in small streams based on visual estimation methods," *Canadian journal of fisheries and aquatic sciences*, vol. 45, no. 5, pp. 834-844, 1988.
- [2] C. Spampinato, D. Giordano, R. Di Salvo, Y.-H. J. Chen-Burger, R. B. Fisher, and G. Nadarajan, "Automatic fish classification for underwater species behavior understanding," in *1st ACM International Workshop on ARTEMIS*, 2010, pp. 45-50.
- [3] D.-J. Lee, R. B. Schoenberger, D. Shiozawa, X. Xu, and P. Zhan, "Contour matching for a fish recognition and migration-monitoring system," in *Optics East*, 2004, pp. 37-48.
- [4] P. X. Huang, B. J. Boom, and R. B. Fisher, "Hierarchical classification with reject option for live fish recognition," *Machine Vision and Applications*, vol. 26, no. 1, pp. 89-102, 2014.
- [5] D.-J. Lee, S. Redd, R. Schoenberger, X. Xu, and P. Zhan, "An automated fish species classification and migration monitoring system," in *Industrial Electronics Society, 2003. IECON'03. The 29th Annual Conference of the IEEE*, 2003, pp. 1080-1085.
- [6] C. Spampinato, Y.-H. Chen-Burger, G. Nadarajan, and R. B. Fisher, "Detecting, tracking and counting fish in low quality unconstrained underwater videos," *VISAPP (2)*, vol. 2008, pp. 514-519, 2008.
- [7] K. Williams, R. Towler, and C. Wilson, "Cam-trawl: A combination trawl and stereo-camera system," *Sea Technology*, vol. 51, no. 12, pp. 45-50, 2010.
- [8] M.-C. Chuang, J.-N. Hwang, K. Williams, and R. Towler, "Automatic fish segmentation via double local thresholding for trawl-based underwater camera systems," in *Image Processing (ICIP), 2011 18th IEEE International Conference on*, 2011, pp. 3145-3148.
- [9] M.-C. Chuang, J.-N. Hwang, K. Williams, and R. Towler, "Multiple fish tracking via viterbi data association for low-frame-rate underwater camera systems," in *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, 2013, pp. 2400-2403.
- [10] M.-C. Chuang, J.-N. Hwang, K. Williams, and R. Towler, "Tracking live fish from low-contrast and low-frame-rate stereo videos," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 25, no. 1, pp. 167-179, Jan. 2015

