

Marius Frantzen Tjore

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Engineering Cybernetics

Marius Frantzen Tjore

Machine vision for quality sorting of salmonid fish eggs

June 2020



Norwegian University of
Science and Technology

Machine vision for quality sorting of salmonid fish eggs

Marius Frantzen Tjore

Cybernetics and Robotics

Submission date: June 2020

Supervisor: Morten Omholt Alver

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Project description

The main purpose of this thesis is to develop a vision system to identify and localise dead roe in a hatchery tub. Furthermore, the thesis will investigate methods for measuring the depth of the roe and possible optimisations of the programs developed during the specialisation project TTK4551.

The following topics and challenges should be considered in more detail:

1. Apply reduced neural networks to the identification problem in order to find a simpler model that still has acceptable accuracy.
2. Explore the possibility of reducing the input image size of the model to increase the speed of the predictions.
3. Further develop algorithms that preprocess the input image in order to reduce the workload of the neural network. The algorithms should be used on a Raspberry Pi 4, with reasonable execution time.
4. Investigate methods for measuring depth of the roe in the hatching tub.
5. Conclude the findings in a report.

The official Norwegian master thesis description is shown on the next two pages.



MASTEROPPGAVE

Kandidatens navn: Marius Tjore

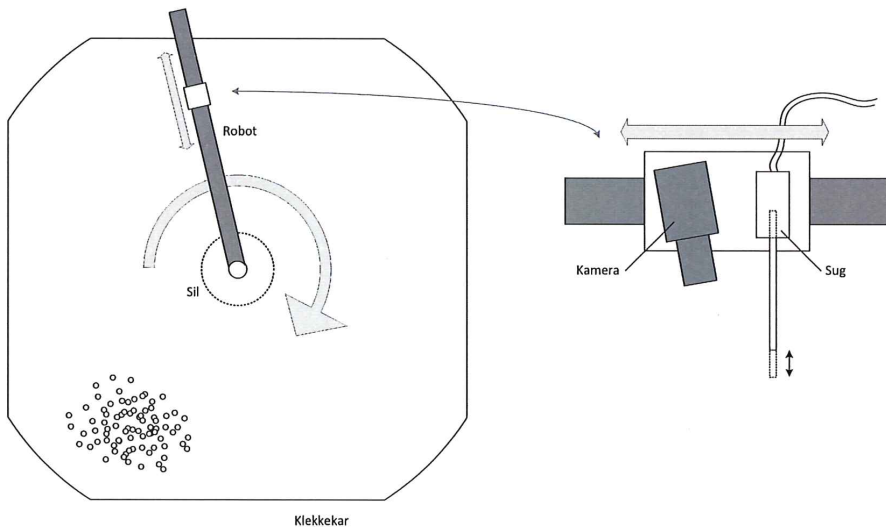
Fag: Teknisk Kybernetikk


Opggavens tittel (norsk): Maskinsyn for kvalitetssortering av øyerogn av laksefisk

Opggavens tittel (engelsk): Machine vision for quality sorting of salmonid fish eggs

Opggavens tekst:

Sopp er et problem på øyerogn av laksefisk og sprer seg raskt mellom rognkorn dersom de infiserte ikke blir fjernet raskt nok. SINTEF Ocean ønsker å utvikle en robot som benytter maskinsyn for klassifisering av infisert / døde rognkorn (se figur). Robotarmen vil være utstyrt med et kamera og en utsugingsenhet. Denne monteres over klekkebakken, og robotarmen flytter seg rundt til hele arealet er scannet. Rognkorn av dårlig kvalitet identifiseres, posisjonsbetømmes og suges ut av roboten.



 SINTEF	Eggrobot, øyerogn			
	21.02.2019	SIZE	PROSJEKTID	DRYKTOID
Torfinn S	SCALE		ESHEET	3 OF 1

Denne masteroppgaven bygger på en prosjektoppgave innenfor samme tema. Sentrale oppgaver i masteroppgaven er:

- Videreutvikle og teste algoritme for deteksjon av døde rognkorn. Algoritmen skal kjøre på Raspberry PI 4 med rimelig kjøretid. Ta i bruk forenkling og optimalisering. Vurder om hardwareplattformen er tilstrekkelig, eller om annen hardware må tas i bruk.
- Utvikle metode for avstandsestimering ved plukking av rognkorn.
- Teste integrasjon av maskinsynsystemet i robotprototype (forutsatt at prototypen blir klar til test)

Oppgaven gitt: 6. januar 2020

Besvarelsen leveres innen: 1. juni 2020

Utført ved Institutt for teknisk kybernetikk

Veileder: Morten Omholt Alver

Trondheim, januar 2020



Fagærer

Abstract

In fish farming, one of the first step of the production process is the quality sorting of roe. Quality sorting involves removal of dead, damaged or fungal infected roe. This is done because dead roe allows for growth of aquatic fungus which causes rapid death of surrounding roe as the fungal infection spreads. The amount of roe that produces fish is a result of how well and often the quality sorting is performed. However, the removal of dead roe is a time-consuming and menial task, and there is therefore an interest in automating it.

This thesis is a continuation of a project from autumn 2019. The focus of the project was to determine if a vision system was able to detect the dead roe. The focus of this thesis was to optimise the vision system so it had a reasonable execution time on a Raspberry Pi (RPi) 4, and test methods for determining the depth of the roe in the hatchery tubs.

This thesis used a Convolutional Neural Network (CNN) with a sliding window approach to identify dead roe. In order to identify what image resolution and network structure size that gave both accurate results and low execution time, testing were done with different models structures and image resolutions. All models were of the sequential type and was based on the structure of VGGNet. The results found was that an input image resolution of 10x10 pixels was most accurate. The end result was a 92.6% decrease in execution time for the software compared to the system developed in autumn 2019. The system was also made more robust to differing light levels by use of dynamic thresholding.

Two noncontact methods for depth measurement was tested. The first involved a time of flight Infrared (IR)-sensor. The results showed poor accuracy and noisy data when used on a metal surface. The second method was a combination of an ultrasonic time of flight sensor and a line laser. The laser method used the camera setup in conjunction with Snell's law to find the water depth. The results show that the ultrasonic sensor was able to measure the distance to the water surface. However, the depth measurement done with the laser was inaccurate due to the high precision required from the mounting of the equipment.

In conclusion, the system is functional if the water depth is measured manually, but further testing of methods for depth measurement is recommended.

Sammen drag

I fiskeoppdrett er kvalitetskontroll og fjerning av døde fiskeegg, også omtalt som rogn, første steg i produksjonsprosessen. Mengden av rogn som overlever dette stadiet og vokser til fiskeyngel er tungt knyttet til hvor ofte og hvor nøye dette blir gjennomført. Grunnlaget for dette er at død rogn gir forurensing i vannet og gir mulighet for at sopp kan begynne å gro. Dette vil føre til rask død av nærliggende rogn. Ettersom prosessen med kvalitetskontroll og fjerning av død rogn er svært tidkrevende og repetativt arbeid, er det ønskelig å automatisere dette.

Denne avhandlingen er en videreføring av et prosjekt fra høsten 2019. Prosjektets fokus var å avgjøre om et visjonssystem var i stand til å identifisere døde rogn. Fokuset for denne avhandlingen var å optimalisere visjonssystemet, slik at det hadde en rimelig kjøretid på en Raspberry Pi 4 (RPI), samt å teste metoder for å måle dybden til rogn i klekkebakkene.

Denne avhandlingen brukte et Konvolusjonelt nevralnettverk (CNN) i kombinasjon med en teknikk kalt "Sliding Window" for å identifisere død rogn. For å identifisere hvilken bildeoppløsning og nettverksstruktur som ga både nøyaktige resultater og lav kjøretid ble det utført testing med forskjellige modellstrukturer og bildeoppløsninger. Alle modellene var av sekvensiell type og var basert på strukturen til VGGNet. Resultatene som ble funnet var at en bildeoppløsning på 10x10 piksler på inngangen til nettet var mest nøyaktig. Sluttresultatet var en nedgang på 92,6% i kjøretid til programvaren sammenlignet med systemet utviklet høsten 2019. Systemet ble også gjort mer robust mot forskjellige lysnivåer ved bruk av dynamisk terskling.

To metoder for dybdemåling som ikke krevde kontakt ble testet. Den første involverte en infrarød-sensor som måler sendetid. Resultatene viste dårlig nøyaktighet og støyende data ved avstandsmålinger mot en metalloverflate. Den andre metoden var en kombinasjon av en ultrasonisk sensor og en linjelaser. Linjelaseren ble brukt i kombinasjon med kameraet og Snells lov for å finne vann dybden. Resultatene viser at ultralydsensoren var i stand til å måle avstanden til vannoverflaten. Imidlertid var dybdemålingen som ble utført med laseren unøyaktig på grunn av den høye presisjonen som kreves i montering av utstyret.

Konklusjonen er at systemet er funksjonelt hvis vann dybden måles manuelt, men ytterligere testing av metoder for dybdemåling anbefales.

Preface

This project is carried out in the Department of Engineering Cybernetics, at NTNU in Trondheim, the spring of 2020. It is submitted as a Master's Thesis for the Engineering Cybernetics course TTK4900. Parts of chapters 1-3 are based on the specialisation project TTK4551, which was submitted the fall of 2019.

During this project, the outbreak of the COVID-19 pandemic happened. This affected the resources available, as NTNU closed its facilities and travelling was advised against to reduce the spread of the virus. Testing conducted in this project was therefore performed after best ability at home. Another consequence of COVID-19 was that the conditional task in the problem description that involved the integration and testing of the machine vision system with the robotic prototype was not started.

I would like to thank Sintef and Norwegian Fish Farms for the opportunity to work on this project. I would also like to thank Morten Alver for constructive feedback, Torfinn Solvang for technical guidance and Stian Aspaas for being available for questions concerning Arctic char and aquacultural farming practises.

Lastly, I would like to thank my partner and family for their great support and for housing me during this project.

Contents

Project description	i
Abstract	v
Sammendrag	vii
Preface	ix
List of figures	xv
List of tables	xvii
Acronyms	xix
1 Introduction	1
1.1 Motivation	1
1.2 Introduction of the project	1
1.3 Objectives	2
1.4 Outline	3
2 Theory	5
2.1 Aquaculture	5
2.1.1 Arctic char	5
2.1.2 Aquacultural farming	5
2.1.3 Roe mortality	6
2.1.4 Aquatic fungus	6
2.2 Digital images	7
2.3 Image processing and Computer vision	8
2.3.1 Erosion and dilation	8
2.3.2 Otsu's thresholding method	9
2.3.3 Hough line transform	10
2.3.4 Structural Similarity Index	11
2.4 Neural Network	12
2.4.1 Neurons	12
2.4.2 Layers	13
2.4.3 Loss function	14

2.4.4	Backpropagation	15
2.4.5	Gradient descent	15
2.4.6	Convolutional Neural Networks	15
2.5	Snell's law	18
2.6	Software optimisation	19
2.6.1	Multiprocessing	19
2.6.2	Profiling	19
2.6.3	Time complexity	20
2.7	Tensorflow	20
2.7.1	Tensorflow Lite	20
3	Previous work	21
3.1	Requirements analysis	21
3.2	Hardware	21
3.3	Camera and lens	23
3.4	Software	24
3.5	Image gathering	25
3.5.1	Labelling of dataset	27
3.6	Data augmentation	29
3.7	Model	29
3.7.1	Training	30
3.8	Image input and preprocessing	31
3.8.1	Thresholding	31
3.8.2	Edge detection and contouring	32
3.9	Reduction of hitboxes	35
3.9.1	Non-maximum suppression	35
4	Optimisation and refinement of previous work	37
4.1	Test setup	37
4.1.1	Profiling of the software designed in previous work	38
4.2	Dynamic thresholding	39
4.3	Removing overlap	39
4.4	Reduction of large bounding boxes	42
4.5	Finding centre of roe	44
4.6	Increased processor utilisation with multiprocessing	44
4.7	Tensorflow lite	45
4.8	Analysis of neural network	46
5	Methods	51
5.1	Distance measurement	51
5.1.1	Test setup	51
5.1.2	Infrared sensor	52
5.1.3	Ultrasonic sensor	53
5.1.4	Laser	53
5.2	Peripheral components	58
5.2.1	Power regulation	59

5.2.2	IR-light control	59
5.2.3	Laser	60
5.2.4	Ultrasonic distance sensor	61
6	Results and discussions	63
6.1	Software	63
6.1.1	Removing overlap	63
6.1.2	Reduction of large bounding boxes	64
6.1.3	Increased processor utilisation with multiprocessing	65
6.1.4	Analysis of neural network	66
6.1.5	Finding centre of roe	70
6.1.6	TensorFlow lite and multiprocessing	72
6.1.7	Discussion of the performance on the Raspberry Pi 4	72
6.2	Depth measurement	73
6.2.1	Infrared sensor	73
6.2.2	Ultrasonic sensor	74
6.2.3	Laser	76
6.3	Peripheral components	77
7	Conclusions and future work	79
7.1	Conclusion	79
7.2	Future work	79
7.2.1	Depth measurement	79
7.2.2	Reduced camera resolution	80
7.2.3	Implementation with roe picking robot	80
7.2.4	Better roe centre estimation	80
	References	81
	Appendices	87
A	PCB schematic	89
B	Installation guide	90
C	Bill of materials	91

List of Figures

2.1	hatchery tub for arctic char in different life stages	6
2.2	Fungus growth in catfish roe	7
2.3	Matrix representation of a image	8
2.4	Example of erosion and dilation	9
2.5	Example of line and Hesse normal form	10
2.6	Example of points on line and corresponding intersecting sinus curves	11
2.7	Illustrations of activation functions	13
2.8	Single neuron and its output function	13
2.9	Graph of binary cross-entropy	15
2.10	Image convolution	16
2.11	Illustration of fully connected dense layers	17
2.12	Example of a flattening layer	17
2.13	Example of max pooling	18
2.14	Example of Snell's law	19
3.1	Connection of the hardware	22
3.2	Benchmarking of Raspberry Pi 3B	23
3.3	Gathering of images for dataset	26
3.4	Monochrome image of roe	27
3.5	Dead roe labelled in COCO Annotator	28
3.6	Examples of positive and negative images	28
3.7	Plot of random gathering of negative images from original image . .	29
3.8	Structure of VGGNet	30
3.9	Structure of reduced CNN model	30
3.10	Histogram of image	31
3.11	Hysteresis	32
3.13	Multiple detections of same roe	35
3.14	Reduced number of boundingboxes after GreedyNMS	36
4.1	Image chosen for testing the different models and algorithms	38
4.2	Histogram and thresholding of test image	39
4.3	Example of large and overlapping bounding boxes	40
4.4	Variables of two overlapping bounding boxes	41
4.6	Reduction of bounding boxes with close proximity	44
4.7	Program flow with two parallel predictors	45
4.8	Model speed by input image resolution, averaged across all models .	47

4.9	Histogram of models by amount of errors	48
4.10	Boxplot of errors by input image pixel size	49
4.11	Structure of chosen models	50
5.1	Cooking pot used to test distance measurement	52
5.2	Distance measuring characteristics (output)	53
5.3	Example of measurement of depth	54
5.4	Pinhole camera approach	55
5.5	Test-setup of laser-depth detection	56
5.6	Difference between the images shown in Figure 5.5	57
5.7	Detection of laser	58
5.8	DC-to-DC converter with bypass capacitors	59
5.9	Relay and power control for IR-light	60
5.10	Power control of 5V laser	60
5.11	Logic level conversion and powering of ultrasonic distance sensor	61
6.1	Predictions from original and chosen models	68
6.2	Prediction from original model with reduced input image resolution	69
6.3	Example of 10x10 pixel input image that contains a dead roe	70
6.4	Examples of finding roe centre	71
6.5	Results of detection and centre estimation on test image	71
6.6	Results from testing the infrared sensor	74
6.7	Boxplot over deviation in results from ultrasonic sensor test	75
6.8	Example of the test-setup of laser depth measurement	76
6.9	3D rendering of the PCB	78

List of Tables

3.1	Brief overview of Raspberry Pi 4 specifications	22
4.1	Profiling results of initial software	38
4.2	Results from tensorflow and tensorflow-lite versions of original model	46
4.3	Profiling results of the Tensorflow Lite model	46
6.1	Profiling results with changed contour-function	63
6.2	Profiling results with remaining overlapping bounding boxes removed	64
6.3	Profiling result after reduction of large bounding boxes	64
6.4	Profiling result after reduction of large bounding boxes on RPi . . .	65
6.5	Results of testing multiprocessing with 1 to 4 processes	66
6.6	Results and timing of models	67
6.7	Results from testing of ultrasonic sensor	75
6.8	Results from testing of laser depth measurement	76

Acronyms

- API** Application programming interface. 20, 25
- CNN** Convolutional Neural Network. v, 3, 15, 16, 44
- CPU** Central Processing Unit. 19, 20, 46, 65, 66, 72
- GPIO** General-purpose input/output. 58, 59, 76
- GPU** Graphics Processing Unit. 20
- IR** Infrared. v, 51, 52, 65, 73, 74, 78
- MOSFET** metal–oxide–semiconductor field-effect transistor. 60, 61
- NFF** Norwegian Fish Farms. 5
- NMS** Non Maximum Suppression. 35, 36, 40, 63, 64
- OS** Operating System. 38
- PCB** Printed Circuit Board. 77
- RAM** Random-access memory. 1
- ReLU** Rectified Linear Unit. 12, 13
- RPi** Raspberry Pi. v, xvii, 1, 22, 23, 37, 38, 44, 45, 47, 58–61, 65, 66, 72, 76, 78
- SSIM** Structural Similarity Index. 56
- TF-Lite** Tensorflow Lite. 20, 45, 46, 72
- USB** Universal Serial Bus. 59, 72

Chapter 1

Introduction

1.1 Motivation

By August 2019, Norway had already exported more than half a million tonnes of fish. The demand for both quantity and diversity of fish will probably continue to rise in the future [1]. If one were to supply this amount of product through commercial fishing, entire species of fish would likely be eradicated through over-fishing [2]. In order to supply this amount of product in a reliable and sustainable way, aquaculture industry is necessary. Automation and streamlining of the aquaculture industry is required to be able to scale up the production to match the increasing demand, while maintaining a good profitability.

1.2 Introduction of the project

SINTEF, in collaboration with Norwegian Fish Farms, seeks to automate parts of the production process in aquatic farming of fish, specifically arctic char. The focus is on the maturing roe, as this is the first step in the production chain and determines the amount of arctic char produced. Some of the roe dies before hatching, as the roe could be damaged when extracted from the char or die of natural causes. When roe dies, they provide a possibility for fungal spores to begin growth. A consequence of this is that the fungus can spread to healthy roe. Therefore, the dead roe affects the survivability of the surrounding roe. The current solution for this problem is manual removal of dead roe through the use of tweezers. This kind of work is repetitive and requires focus as the roe is delicate in its early stages. The desired solution to this problem is a robot that can detect and remove dead roe automatically.

During the project completed autumn 2019, the following work was done:

Hardware

A RPi 4 model B with 4GB of Random-access memory (RAM) with the necessary accessories was bought to be used as the hardware platform for the project. Four different cameras were evaluated, and it was decided that the colour camera Sony

IMX250 and the monochrome camera Ximea Mq013MG was to be tested. Different lenses were also considered before it was decided to use the Spacecom 12.5 mm lens.

Dataset

The images were gathered on October 25th 2019. A total of 596 monochrome images and 438 colour images were taken. After comparing the images, it was decided to use the monochrome images. The reasoning for this was that the roe were clearly visible in both types of images, whereas the monochrome images were of a smaller data size. From the monochrome images, 1173 dead roe were manually labelled by use of the COCO-annotator tool in order to create a dataset [3]. The labelled images were split into three sets of images; a training set (60%), a validation set (30%) and a test set (10%). The training set was augmented by use of rotation and flipping, which increased the amount from 696 to 5568 images.

Neural network

The images were processed by thresholding in order to identify areas where there might be dead roe. Afterwards, they were treated with a sliding window approach in order to do predictions with the neural network. Two neural networks were created. The first model achieved a accuracy of 97.4% on the test-set. The second model was created as a reduced form of the first model. It achieved a accuracy of 98.7%. While both of the model produced good results, the execution times were 203 seconds and 103 seconds respectively. This was deemed too slow for use in the application.

1.3 Objectives

The goal of this project can be summarised in the following three points:

- Optimise the program in order to reduce execution time without affecting accuracy.
- Further refine the neural network to reduce execution time without affecting accuracy.
- Test methods for determining the depth placement of the roe.

1.4 Outline

The report is organised as follows

Chapter 1 Introduction of project, summary of previous work and project aim.

Chapter 2 This chapter consists of the theory, which contain information about software optimisation, distance measurement and image manipulation.

Chapter 3 Description of previous work. It contains what camera and lens that was used, how the images was gathered and used to create a dataset, how the CNN was designed and what image operations that was done.

Chapter 4 In this chapter, the changes and optimisations of the software shown in the previous chapter is described.

Chapter 5 In this chapter, it is described how the different methods of distance measurement functions and how the tests are performed. This includes schematics for connecting and powering the equipment used.

Chapter 6 This chapter presents and discusses the results of the project.

Chapter 7 The conclusion of the results and a presentation of suggestions for future work.

Chapter 2

Theory

2.1 Aquaculture

Aquaculture, also known as aquafarming, refers to the farming of fish and other aquatic organisms [4]. The practise involves cultivating populations of fish under controlled conditions, in contrast to commercial fishing that relies on the natural populations.

2.1.1 Arctic char

Arctic char is a part of the Salmonidae family [5], which includes salmon and trout. The char has a body that resembles salmon. There are two variants of arctic char in Norway: a freshwater variant and an ocean variant. No other fish are found as far north as the freshwater variant. It thrives in cold, clear and oxygen-rich water, and has flesh-coloration in range from a pale pink to bright red. It has been described to have a taste between trout and salmon [6]. Arctic char usually weighs between 1 to 4.5 kilograms when sold commercially.

2.1.2 Aquacultural farming

The farming process begins with the production and maturing of char roe. Fertile char is gathered and either cut open to collect the roe or stroked to get the roe released without harming the fish. By gathering roe and semen from fish with desired traits, such as a faster growing speed and larger filet yields, it's possible to develop a subspecies of arctic char that is better suited for the aquaculture industry. After insemination, the roe is placed in hatching tubs with gently flowing water. Flowing water helps counteract fungus if the roe are rolled by the flow [7], but is mainly used to avoid stagnant water. Here they mature from roe, to alevin, and finally into fry. After this, the fish are moved between tubs of different sizes as they mature from fry to parr, to juveniles and finally to adults ready for harvest.

Norwegian Fish Farms (NFF) hatchery tub is constructed of plastic. It has different layers made out of metal, that can be removed during the different life stages. The metal layers has slits that allows the char to follow the instinctive need to burrow downward after hatching. A representation of the hatchery tub is shown in

Figure 2.1.

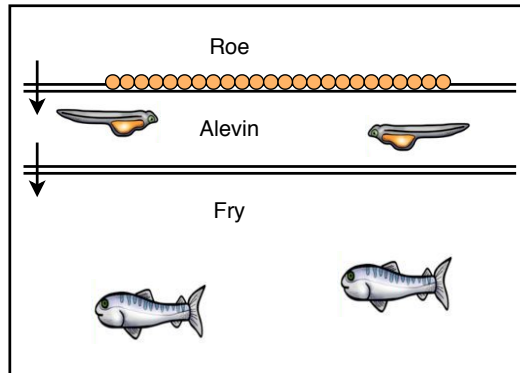


Figure 2.1: Simple representation of the hatchery tub for arctic char in different life stages

2.1.3 Roe mortality

When the roe have been inseminated there is a high mortality rate. Causes of death can be: roe that is defective, roe that was damaged during gathering and handling and lastly some roe may not have been properly inseminated. Some of the roe may have been dead or dying before the gathering began. These are mortality causes that are hard to reduce. Furthermore, the dead roe introduces pollution to the hatchery tub and provides a staging ground for fungus. The healthy roe are resistant to the fungal spores, but are vulnerable to the spread of the growing fungus. The fungus spread is therefore preventable by removing the dead roe before it affects the surrounding healthy roe. Stian Aspaas, aquaculture biologist and co-owner of Norwegian fish farms, has experienced a 40% mortality rate during the roe stage and estimates that it can be reduced to 20-10% if the dead roe are removed early enough [8].

2.1.4 Aquatic fungus

There exists many species of fungi, where Saprolegnia is the most common fungus responsible for significant infections in freshwater roe worldwide [9]. The fungus and its spores thrive in damp environments with decaying organic matter such as dead or unfertilised roe. As the fungus grows, it releases spores into the water that in turn infects any other dead roe. As seen in Figure 2.2, the infection is manifested as white mycelial growth on the surface of the roe.

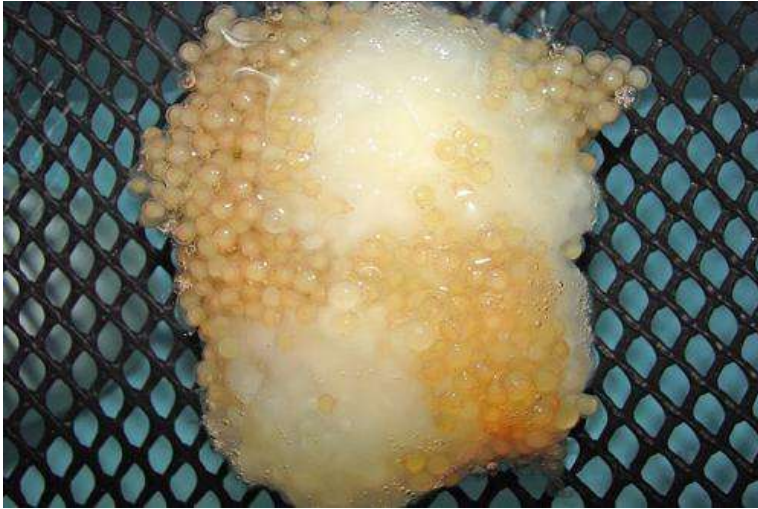


Figure 2.2: Fungus growth in catfish roe, image from phys [10]

2.2 Digital images

Digital images are represented as a matrix, where the values of the dimensions differ with how much information is stored. The matrix has three dimensions, consisting of length, height and colours as shown in Figure 2.3. The values usually range from 0-255, but other cameras can have higher range to capture the colour intensity better. This value signifies the strength of the light registered on this spot. By combining red, green and blue, it is possible to recreate the colours in the light spectrum visible to the human eye. Grayscale images can be represented as two-dimensional matrices, where the values represent the overall light intensity in one spot. As a result of this, grayscale images are only a third of the size compared to an equivalent colour image. In addition, cameras can capture light outside of the human visible light spectrum, such as infrared light.

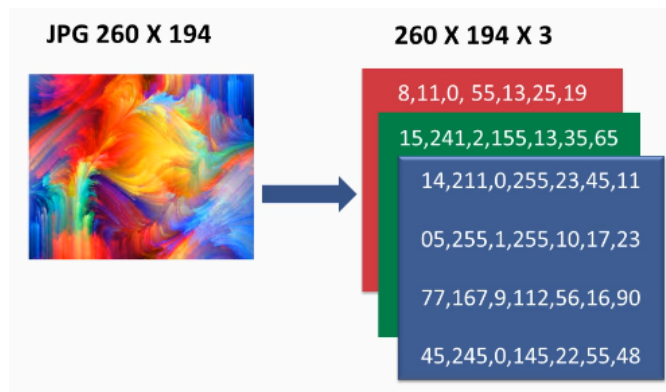


Figure 2.3: Matrix representation of a image, illustration from Packt [11]

2.3 Image processing and Computer vision

Image processing refers to the act of doing operations on an image in order to enhance it or to extract information. Computer vision is a scientific field that deals with how computers can be made to gain an understanding of the content in digital images. The goal is to be able to mimic the human vision system in a computational form, which can be used to create autonomous systems that can accomplish sophisticated tasks. Examples of computer vision tasks are face detection or autonomous inspection of windmills [12][13].

2.3.1 Erosion and dilation

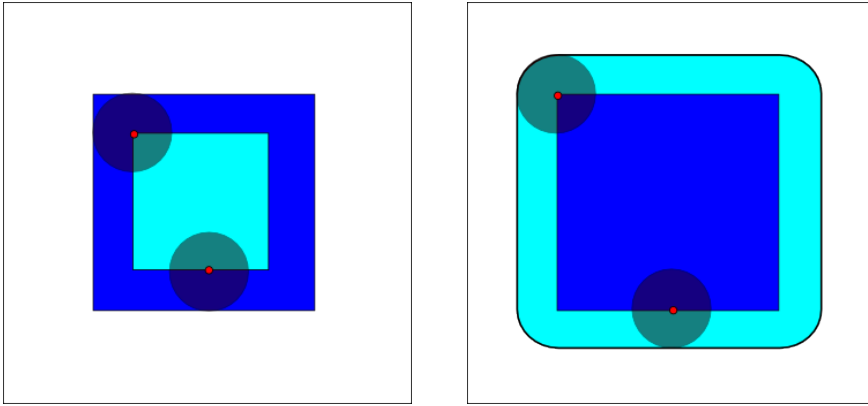
Erosion and dilation are the two fundamental operations in morphological image processing [14]. Both operations uses a predefined shape, often referred to as a filter, which is a two-dimensional array.

Erosion

When eroding, it is the neighbouring pixels that affects whether the considered pixel is retained or removed. Which neighbouring pixels that affects the decision is defined by the shape and size of the filter. Depending of the values of the surrounding pixels defined by the filter, the considered pixel is either removed or retained. An example of erosion is shown in Figure 2.4a.

Dilation

Dilation concerns the superimposing of the filter onto every pixel with a value above a value threshold. The values of the neighbouring pixels defined by the filter is changed, and the edges in the image swells. An example of dilation is shown in Figure 2.4b.



(a) Erosion, image from Wikipedia [14]

(b) Dilation, image from Wikipedia [15]

Figure 2.4: Erosion and dilation, dark-blue is original, light-blue is result, the red dot is the considered pixel and the dark shade is the filter

2.3.2 Otsu's thresholding method

Otsu's thresholding method is an adaptive method for thresholding an image. The optimal thresholding value is found by locating the minimal within-class variance [16]. The within-class variance is a weighted value sum of the variances of the two classes. The method is performed in multiple steps, where the first step is to generate a histogram of the image. Then the image is split in two classes, once for every threshold value. The within-class variance (V_w) is then found for each thresholding value using the following equations.

$$\sigma^2 = \frac{\sum (X_i - \mu_c)^2}{N_c} \quad (2.1)$$

$$W_C = \frac{N_c}{N_t} \quad (2.2)$$

$$V_w = \sum (W_c \cdot \sigma_c^2) \quad (2.3)$$

X_i is pixel value, μ_c is the mean within the class, N_c is the number of pixels within a

class, N_t is the total number of pixels and σ_c^2 is the variance of a class. By choosing the minimal V_w found, the optimal thresholding value is selected.

2.3.3 Hough line transform

The Hough transform is a feature extraction technique that can detect any shape that can be described in a mathematical form. It is also robust in the sense that it can detect shapes even if it is broken or distorted by a small degree.

The least complex shapes to mathematically describe are straight lines, represented in the form $y = ax + b$. However, vertical lines pose a problem, computationally, as unbounded values for a can occur. The lines are therefore described in Hesse normal form $r = x \cos(\theta) + y \sin(\theta)$ [17], where r is the shortest distance from the line to the origin, θ is the angle between the line and the x axis. An example of this is shown in Figure 2.5.

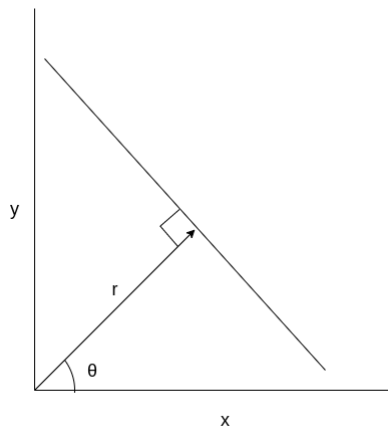


Figure 2.5: Example of line and Hesse normal form

A set of straight lines are drawn through every point of the image. These sets of lines corresponds to a unique sinus curve in the plane of r and θ . Any points that form a straight line will have sinus curves that crosses in the plane at the r, θ value for the line between the points. Thus, a point where many curves intersect will indicate a line as shown in Figure 2.6.

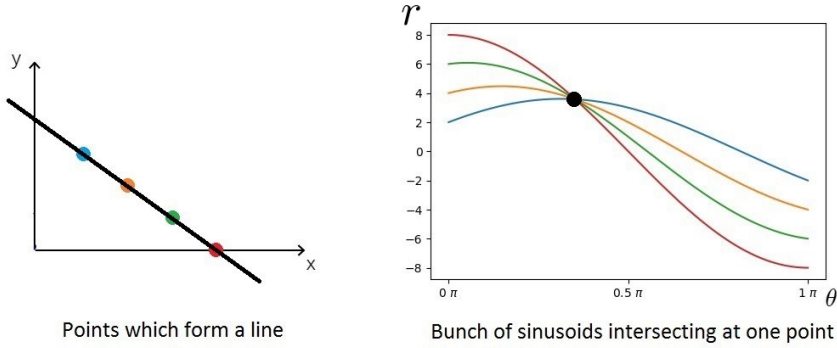


Figure 2.6: Example of points on line and corresponding intersecting sinus curves, illustration from Medium [18]

The Hough transform algorithm checks every point in the image and determines if there are lines between them, calculate r and θ and then increments the value in a two dimensional array where r and θ describes the placement in the array. Lines can then be found by finding local maximas or by use of a threshold value.

2.3.4 Structural Similarity Index

The Structural Similarity Index is a method of comparing different images, or more specifically a method of measuring the perceptual difference between two images [19]. The results are represented as a value between -1 and 1, with 1 being perfectly identical images and 0 indicating no structural similarity. A negative value indicates that the local image structure is inverted [20].

The method is a multiplication of three terms, the luminance term $l(x, y)$, the contrast term $c(x, y)$ and the structural term $s(x, y)$ as shown in (2.4) [21].

$$SSIM(x, y) = [l(x, y)]^\alpha \cdot [c(x, y)]^\beta \cdot [s(x, y)]^\gamma \quad (2.4)$$

Where l, c , and s are calculated as

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \quad (2.5a)$$

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \quad (2.5b)$$

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \quad (2.5c)$$

Where μ_x is the average of x , μ_y is the average of y , σ_x^2 is the variance of x , σ_y^2 is the variance of y , σ_{xy} is the covariance of x and y . The values C_1 , C_2 and C_3 are calculated as

$$C_1 = (0.01 \cdot L)^2 \quad (2.6a)$$

$$C_2 = (0.03 \cdot L)^2 \quad (2.6b)$$

$$C_3 = \frac{C_2}{2} \quad (2.6c)$$

L is the maximum range of the pixel value, usually 255 for the 0-255 range or 1 for the 0-1 range. With the default values $\alpha = \beta = \gamma = 1$, the SSIM function can be simplified as follows.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (2.7)$$

2.4 Neural Network

Neural networks are a set of algorithms that are designed to recognise numerical patterns. By translating real-world data into vectors it is possible to generate a machine perception of the data, whether it is text, images or sound.

2.4.1 Neurons

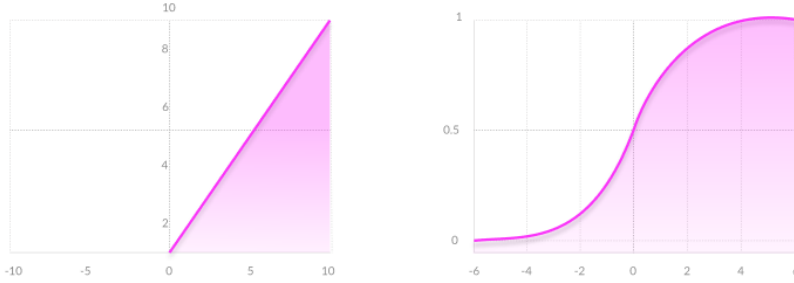
The basic building block of a neural network is the neuron, which takes inspiration from the human brain [22]. Neurons take inputs from other nodes or external inputs. The inputs have individual weights that are applied to the input value. Lastly, the sum of all the inputs and a bias is taken to create the output. The weight of an input indicates the importance of the input, compared to the other inputs.

Activation functions are a fixed function applied to the input, in order to have an output with a given characteristic. Two common activation functions are Rectified Linear Unit (ReLU) and Sigmoid. ReLU is a threshold at zero, such that it ensures positive values. The function is as follows

$$f(x) = \max(x, 0)$$

Sigmoid squashes the input value between 0 and 1, and the function is as follows

$$f(x) = \frac{1}{1 + e^{-x}}$$

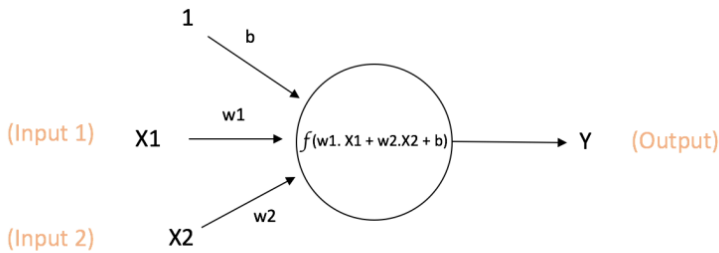


(a) Illustration of ReLu

(b) Illustration of Sigmoid

Figure 2.7: Illustrations of activation functions, from Missinglink [23]

A visualisation of a neuron and its output function can be seen in Figure 2.8, where inputs are x , weights are w , bias is b and the output Y is the result from the activation function f .



$$\text{Output of neuron} = Y = f(w1.X1 + w2.X2 + b)$$

Figure 2.8: Single neuron and its output function, illustration from ujjwalkarn [24]

2.4.2 Layers

Neural networks commonly described to be divided into three types of layers, the input layer, the output layer and the hidden layer. The input layer is where the

user inputs data to the network. In the case of images, it is the pixel values that go into the network, which means the input layer needs to have the same dimensions as the image.

The output layer is where the predictions are made, and the structure is dependent of how the network is used. An example of this is a classification task, where the output layer will often have a number of nodes that matches with the number of possible classes.

The hidden layer is composed of one or multiple actual layers. The naming stems from the fact that the layer only interacts with the output and input layer, without any input from the user. Most, if not all, differences in network architecture is found in the hidden layer.

2.4.3 Loss function

Finding the loss of a network is an important part in the training process as it measures how far from the correct solution a prediction from the network is. The objective of the training is to minimise this value. This is also referred to as minimising the loss function.

One example of a loss function is binary cross-entropy. This loss function increases rapidly the further off the prediction is from the correct answer. With y as the class indicator and p as the prediction, the function is given as

$$-(y\log(p) + (1 - y)\log(1 - p))$$

which gives a loss value as shown in Figure 2.9.

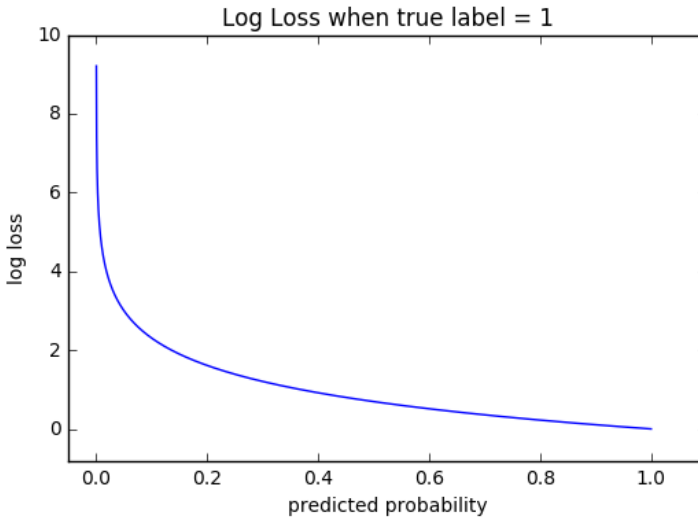


Figure 2.9: Graph of binary cross-entropy, visualisation from Docs [25]

2.4.4 Backpropagation

Backpropagation, or backward propagation of errors, is the process of tuning the weights of the neurons based on the loss. This works in the opposite direction compared to when the neural net is used to make a prediction, by moving from the output layer towards the input layer.

2.4.5 Gradient descent

Gradient descent is a method for finding the minimum of a function, which in this case is the loss function. There are variations of algorithms that solve this problem, but common for most is that they iteratively find the steepest direction with the assumption that this will lead to the global minimum.

2.4.6 Convolutional Neural Networks

It is possible to feed each individual pixel into a neural network and learn some features pertaining to the pixels. However, this does not take the information about how pixels relate to each other into account, which is important when it comes to images. This is where the CNN is used.

The CNN gets its name from the convolutional layer that is often used in this kind of network. The network takes information about how pixels that are spatially close to one-another relate. This happens through the use of a kernel, which is usually a small 3x3 matrix. The values that the kernel contains is referred to as a filter. The kernel with a sharpening filter is shown in Figure 2.10.

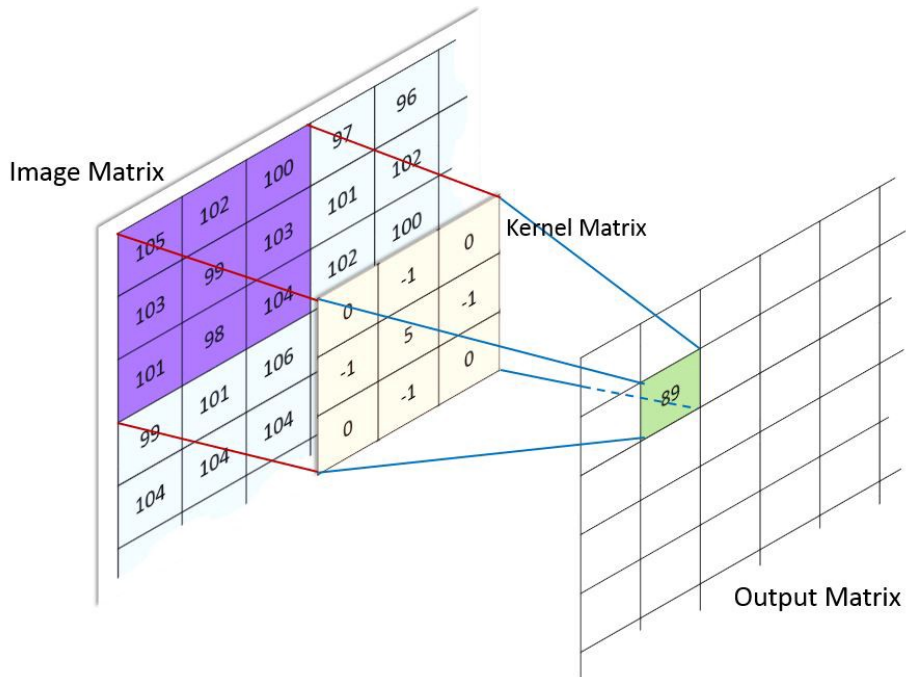


Figure 2.10: Image convolution, illustration from Computer Science Wiki [26]

Beyond this, other common layers used in a CNN are: Dense layers, flattening layer and max pooling layers. These are described below.

Dense layer

Dense layers are layers where every neuron in the previous layer is connected to every neuron in the dense layer. For this reason, the dense layer is also often referred to as fully connected layers. An example of this can be seen in Figure 2.11.

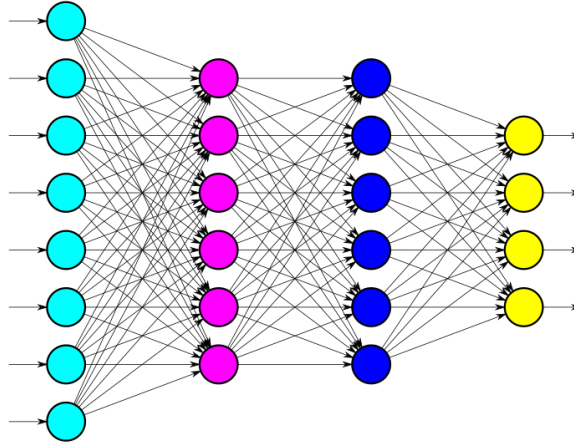


Figure 2.11: Illustration of fully connected dense layers with different colours per layer, image from Computer Science Wiki [27]

Flattening layer

The flattening layer transforms a multidimensional matrix of features into a vector, such that it can be inserted into a dense layer. This is shown in Figure 2.12, where a two dimensional matrix is transformed into a vector.

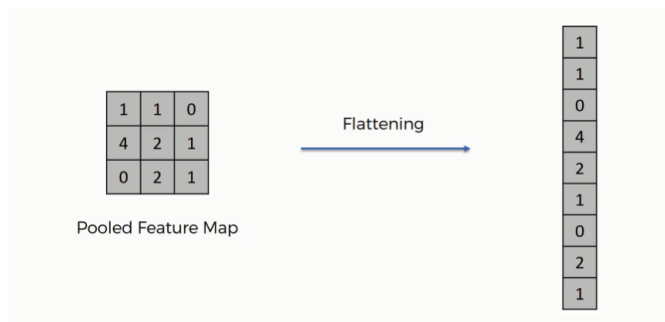


Figure 2.12: Example of a flattening layer, image from Computer Science Wiki [28]

Max pooling layer

Max pooling layers are used to down-sample the input data, in such a way that the most present features are preserved. It also has the advantage of reducing the total computational cost of the network. An example of a matrix going through a 2×2 max pooling operation can be seen in Figure 2.13, where a 4×4 matrix is reduced to a 2×2 matrix by only preserving the highest values in 2×2 windows.

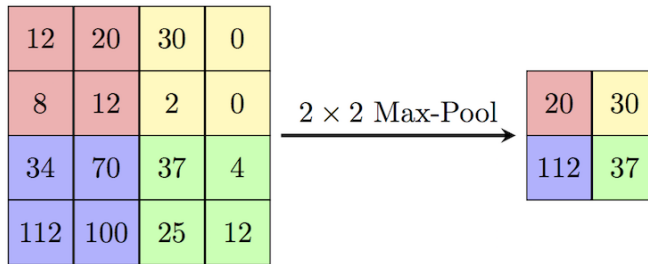


Figure 2.13: Example of max pooling, illustration from Wiki [29]

2.5 Snell's law

When light changes the medium it travels through, a refraction occurs. Snell's law describes the relationship between the refraction and the angle at which the light enters the new medium, known as the angle of incidence. The formula is given in (2.9), with an example shown in Figure 2.14. n_1 is the incident index, n_2 is the refracted index. These are found by taking the speed of light in vacuum c and dividing it by the speed of light in the new medium [30].

$$n = \frac{c}{v} \quad (2.8)$$

θ_1 is the incident angle, θ_2 is the refracted angle and v is the speed of light in the given medium. The refractive index of water and air is approximately 1.33 and 1 respectively [31].

$$n_1 \sin \theta_1 = n_2 \sin \theta_2 \quad (2.9)$$

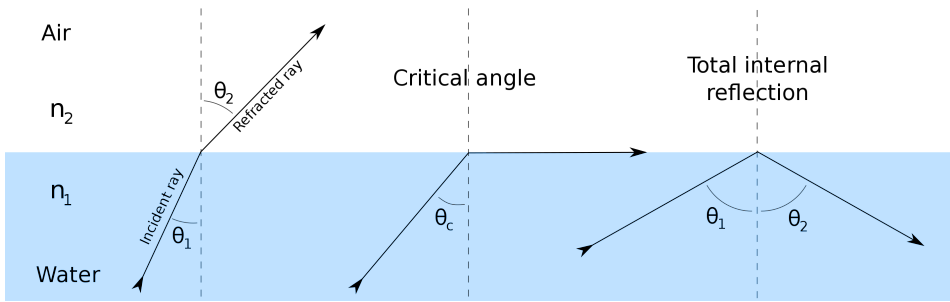


Figure 2.14: Example of Snell's law, illustration from Wikipedia [32]

2.6 Software optimisation

A significant part of this project was optimising the speed of software without negatively affecting the results. This section contains theory that are relevant for optimising software.

2.6.1 Multiprocessing

Computer programs often run sequentially, which means that all actions are taken in the order they are listed. The length of the execution time for the actions is dependant on the speed of the processor executing them, assuming all actions are internal computations. Multiprocessing is the ability to use multiple processors in the same system. This covers both a computers ability to support two or more Central Processing Units (CPUs) and a programs utilisation of multiple CPUs. This enables computers to execute instructions in parallel, or in other words, do more work in the same amount of time.

2.6.2 Profiling

Program profiling is a form of analysis often used to aid in program optimisation. This allows for measuring of memory usage, finding the time complexity of the program, measuring of time used, counting the usage of instructions or function calls and the frequency and duration of these calls. This gives a better understanding of the program flow and highlights which parts of the program that are computationally heavy or uses the largest parts of the execution time.

2.6.3 Time complexity

Time complexity is a way to describe the amount of time an algorithm takes to run. The most common way to do this is by describing the best or worst case execution times in a mathematical form. The variable used is often the amount of elementary operations used by the program, but can also be the amount of data that is the input to the program. Big O, Θ and Ω notation is commonly used to describe the growth rates of functions. $O(\cdot)$ is used to describe the worst case, or maximum theoretical time usage of the algorithm which often is the most interesting to know, while $\Omega(\cdot)$ is used to describe the best case, or the minimum theoretical time usage of the algorithm [33]. $\Theta(\cdot)$ is used to describe the average time usage.

2.7 Tensorflow

Tensorflow is an open-source library originally developed for use in Googles Brain Team for creating and using machine learning models, but it is now widely used [34]. Tensorflow has a architecture that allows for computing on a wide range of devices such as servers, desktop computers and mobile devices. It can be configured to use both CPUs and Graphics Processing Units (GPUs), which allows for rapid training of complex machine learning networks. It functions as a Python Application programming interface (API) for high-performance C++ code.

2.7.1 Tensorflow Lite

Tensorflow Lite (TF-Lite) is specifically designed for use on mobile and embedded devices. The TF-Lite Converter takes a trained Tensorflow network and converts it to a binary representation of the original model. This is based on FlatBuffer, which is a way to effectively serialize data into a small binary file while also having quick access to that data [35]. TF-Lite can therefore bypass a lot of file parsing and unparsing that can be computationally expensive. TF-Lite also implements various other optimisations, some that can reach all the way down to the hardware.

Chapter 3

Previous work

3.1 Requirements analysis

The first task that was completed was a specification of the system requirements. As the focus of the project was to find a method of identifying dead roe, the image requirements were the main focus. As it becomes hard to recognise features when the image is of less than 100 pixels, a requirement was made that each roe should have a resolution of at least 10x10 pixels when collecting images for the dataset. The reasoning for this was that the images could be down-sampled later in order to test with lower resolution images.

The requirement for the hardware was split into a processing system and a camera. The processing system should be able to handle the processing demands from the vision system algorithms within a reasonable time limit. The system must also be able to use the camera to gather images. The camera must have a sufficient resolution to meet the image requirements.

3.2 Hardware

The first part of the project was to determine what hardware to test for suitability. The main hardware components needed in the project was a camera and a computer. The computer needed to be able to make use of the camera and enable the processing of the computer vision system. The camera had to be able to capture images with sufficient quality to recognise individual roe, which was specified to be 10x10 pixels per roe. The Figure 3.1 shows how the entirety of the final system will be connected, where the project was focused on the processing system and camera.

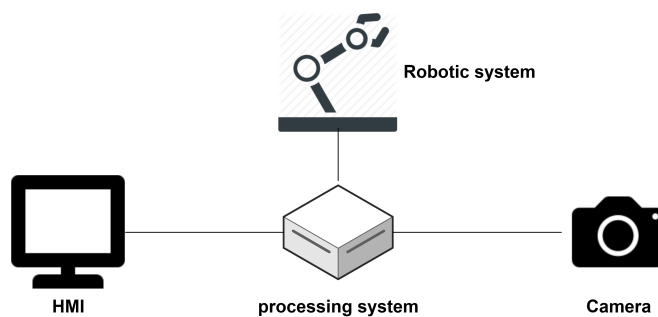


Figure 3.1: Connection of the hardware

Raspberry pi 4

It was decided to use a RPi 4 as the hardware platform for this project. The RPi is a popular system on a chip, which is an integrated circuit with all the components of a computer. Its main selling points are its small size and general usability. It supports Ubuntu, which is a widely used Linux operating system. Some specifications are listed in Table 3.1.

<i>Component</i>	<i>Capability</i>
Processor	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC 1.5GHz
Ram	4GB LPDDR4-3200 SDRAM
USB	2 USB 3.0 ports; 2 USB 2.0 ports.
Memory	microSD-card
GPIO	40 pin GPIO header
Environment Operating temperature	0 C to +50 C
Size	8.5x5.5x2 cm

Table 3.1: Brief overview of Raspberry Pi 4 specifications [36]

The price of a RPi 4 is about 730NOK, without a case, microSD or a power supply [37]. As the RPi offers 40 GPIO pins, it is a suitable platform for development of the robotic parts of the roe extraction robot. For use in the processing of a vision system, it is likely too slow to provide real-time capabilities as the benchmarks of the RPi 3B+ show in Figure 3.2. It is important to note that the RPi 4 has upgraded hardware and will perform better than the model 3, the benchmarks are just used to give an indication of performance.

Board	MobileNet v1 (ms)	MobileNet v2 (ms)
Coral Dev Board	15.7	20.9
Coral USB Accelerator	49.3	58.1
NVIDIA Jetson Nano (TF)	276.0	309.3
NVIDIA Jetson Nano (TF-TRT)	61.6	72.3
Movidius NCS	115.7	204.5
Intel NCS2	87.2	118.6
MacBook Pro	33.0	71.0
Raspberry Pi (TF Lite)	271.5	379.6
Raspberry Pi (TF)	480.3	654.0

Figure 3.2: Benchmarking of Raspberry Pi 3B, from Hackster [38]

The goal was to create an optimised algorithm that can be run on the RPi, but since it was difficult to predict how much processing power the final vision-algorithm needed, possible solutions to supplement the RPi were explored.

The first option is to offload the processing from the RPi to a more powerful processing unit. This requires a distributed system, where the RPi is responsible for gathering images, while another local computer or a cloud system is responsible for processing the images. The negative aspects of this method is that it has more points of failure as more systems are used. An example of an added point of failure is the internet connectivity necessary for the communication.

The second option is to extend the image computing power with a Coral USB accelerator. The accelerator uses an application-specific integrated circuit designed by Google to give high performance for machine learning algorithms on low-power devices [39]. As it only supports Tensorflow Lite, it requires that the model that is used is converted before use. This may require additional work, but is a consideration as the accelerator can give 9-11 times the performance of the RPi as shown in Figure 3.2.

3.3 Camera and lens

NTNU and Sintef had multiple cameras and lenses available for consideration and testing. It was decided to use the Ximea Mq013MG 1.3 MP 60 fps monochrome camera, in combination with a Spacecom lens with a focal length of 12.5 mm and 1.4 inch optical format.

The positive traits of the USB3.0 bus interface of the Ximea camera is that it supplies power and has no CPU usage. The negative traits of the USB3.0 bus are the short maximum cable distance and difficulty of connecting multiple cameras, but these negated as there is only one camera used and the cable length are under 5 meters.

The first step was to find the size of the area captured in the image. This is found from the focal length, length from the object and sensor size. The formula is

$$w = L/f * C_w$$

$$h = W * C_h$$

Were L is distance from object, f is focal length and C is a constant given from the sensor size. Since the camera sensor size is 2/3 inch, the constants is $C_w = 8.8$ and $C_h = 0.75$ [40].

During the excursion to Tydal, it was found that a good width for the images was around 40 cm, which results in a distance of 60 cm above the tub. This gave a overview of half of the hatchery tub, with an area of 42.2x31.6 cm ($w \cdot h$) in the images. However, when the vision system is implemented with the robotic system, a shorter distance from the tub may be wanted. A lens with 8 mm focal length will give a 33x24.75 cm ($w \cdot h$) image at a 30 cm distance, such as the Tamron 8mm 1/1.8inch C mount Lens [41].

3.4 Software

Throughout this project, different software was needed for the cameras, as well as for tasks such as labelling of images. The programming language chosen for this project was Python3.6, as it is a popular programming language that has many good libraries used in the field of vision systems. In addition, Python allows for fast prototyping and developing.

Software used for the camera was XIMEA Camtool, which were available from the vendors website [42]. For labelling of images, COCO Annotator was used [3].

The following libraries for python were used:

Numpy: Scientific computing

math: Mathematical functions

Matplotlib: Plotting library

json: JSON encoding or decoding

cv2: Open-source Computer Vision Library

pickle: Object serialization

tensorflow: Open-source machine learning

tensorflow lite: Open-source machine learning for integrated devices

Keras: Open-source python API for TensorFlow

scikit: Open-source image processing and machine learning

spidev: SPI interfacing

ximea: Ximea camera driver

multiprocessing: Multiprocessing

cProfile: Profiling

3.5 Image gathering

25.October 2019, a trip to Norwegian fish farms facility in Tydal was organised. The purpose of the trip was to gather images of roe for use in the creation of a dataset. In addition, a guided trip through the entire facility was given, included a demonstration of how the char roe was handled in its maturation process in order to give a better insight into the workflow.

The original plan that had the camera mounted to a board that were fastened in two points at each end of the maturation tub, was quickly proved to be non-feasible as the tubs were positioned to close to each other. As a result, the image-rig had to be built with only one point fastened to the tub. The original rig was constructed of simple parts to allow for configurability, which made the new construction of the rig possible.

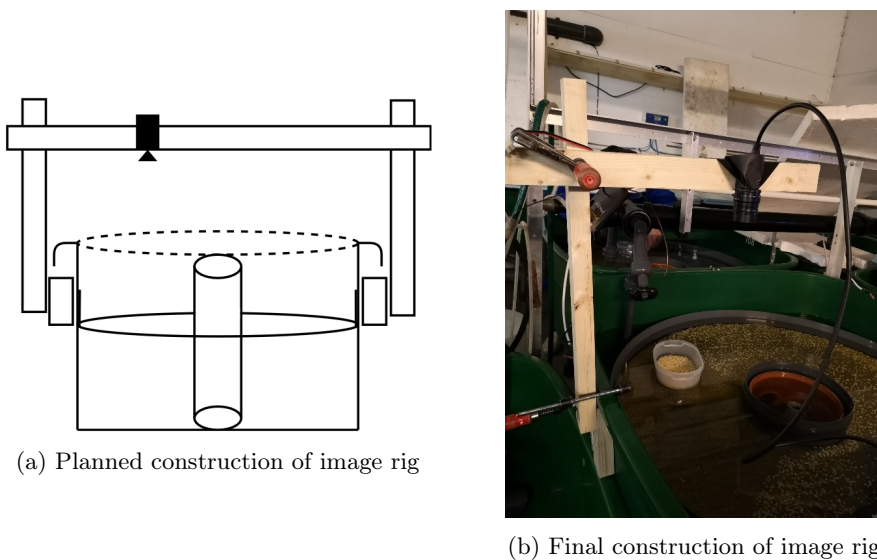


Figure 3.3: Gathering of images for dataset

The cameras were mounted 60 cm from the roe when the images were taken. Before the mounting of the camera, the lens was adjusted on a checker-board pattern in order to get it semi-calibrated. The last of the calibration was done on the mounted camera. The calibration was done by adjusting the aperture to maximum allowed light intensity, before adjusting the focus. As a result, it was easy to see when the camera was in the correct focus. After this, the aperture was adjusted to the correct level.

The area had mounted powerful overhead lights that were used to provide good lighting conditions for the workers at the facility. When images was gathered with the monochrome camera, these overhead lights were turned off and the tubs were illuminated by the IR-light instead. As seen in Figure 3.4, the dead roe is quite recognisable and has a pronounced contrast in comparison to the healthy roe.

The reasoning for using a monochrome camera for the vision system was as follows: Firstly, some lightning conditions is more beneficial to the roe than others. In a meeting with Norwegian fish farms, Stian mentioned that IR-light could be better as normal lightning exposed the roe for stress. Lastly, the memory size of the images does impact the speed of the image processing. The monochrome images are smaller, 0.69 MB compared to 5.37 MB of the colour images. The difference in size comes from the fact that the colour camera has a higher resolution, but also from that the colour images are three times the size of an equivalent monochrome image.

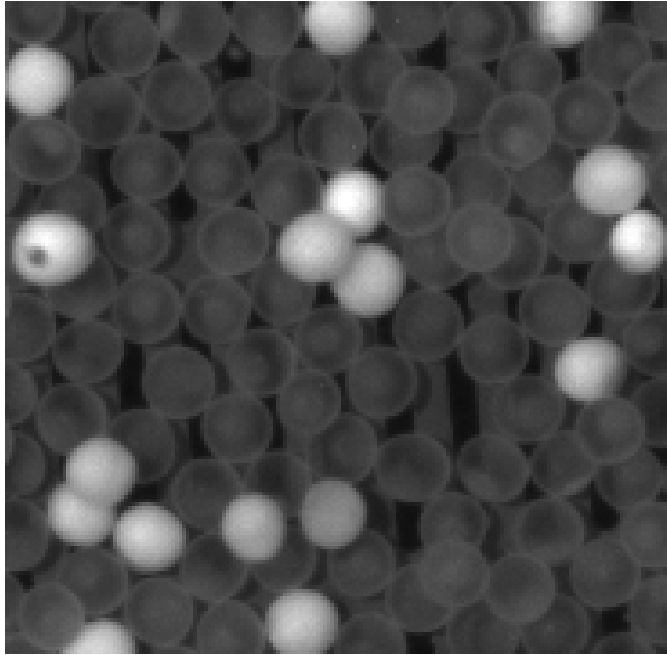


Figure 3.4: Monochrome image of roe

3.5.1 Labelling of dataset

The first step in the creation of a dataset is labelling of the images. The work consists of defining where the dead roe is located in the images by defining bounding boxes. As this is a classification test, the labelled dead roe is defined as the positive images. Furthermore, the images that don't contain dead roe will be considered negative images. The COCO-annotator was used to label the eggs manually [3]. This program was chosen because of multiple reasons that are listed under.

- The annotator has a good interface, which offer multiple ways of defining regions and modifying these. Interface is shown in Figure 3.5
- The labels come in the COCO-format which is widely used and contains a lot of information.
- Is built as a server and a web-application, so it can be configured to be reachable from any computer. This allows more flexibility in where to work from.
- It is possible to integrate a trained network to help in the labelling process, to reduce the amount of work needed.

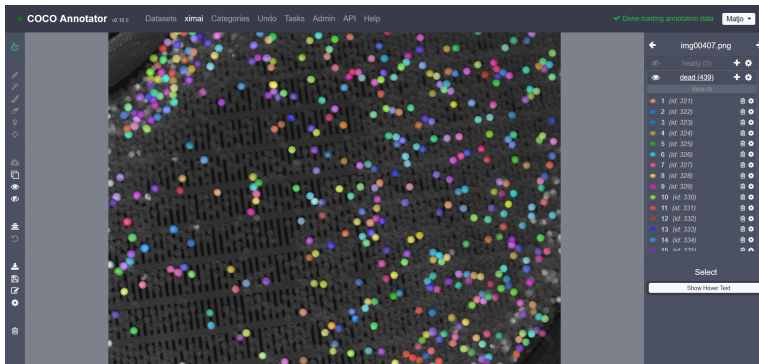


Figure 3.5: Dead roe labelled in COCO Annotator

After some images were labelled, the defined areas needed to be extracted into smaller cut-outs and were divided into a training-set, a validation-set and a test-set. The images were split with 30% in the validation-set, 60% in the training-set and 10% in the test-set. The reason for splitting the images in this way was in how the images were used. The training-set and validation-set was used in the training of the network. The training-set was used to tune the parameters, and the model was then tested on the validation set. This gave a bias for the validation-set, so some images had to be reserved to a testing-set that was used after the network model had been trained. This image-set gives more accurate results over the models performance.

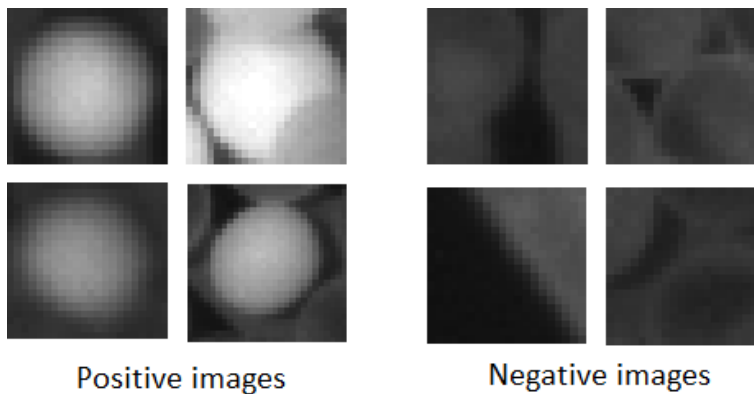


Figure 3.6: Examples of positive and negative images

The negative images were taken at random from the images, with a check that none of the negative areas overlapped the labelled areas of the dead roe. The spread of the negative images is shown in Figure 3.7.

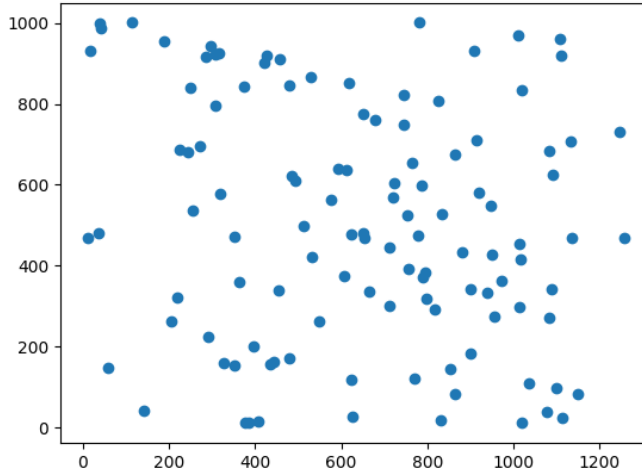


Figure 3.7: Plot of random gathering of negative images from original image

3.6 Data augmentation

The quantity and diversity of the dataset is of great importance for the training of a neural network. A simple rule of thumb is that larger quantity and more diversity gives better results. Data augmentation can significantly increase the diversity of data that can be used in training, without going through the process of gathering more real data.

For data augmentation, it was decided to go for the commonly used techniques of rotating and flipping. These are considered safe transformations as the methods needed to preserve the characteristics of the dead roe, and are also simple to implement [43]. Although the roe are round, they are not perfectly so and the augmentation methods will increase the diversity of the dataset. The images were augmented by vertical flipping and rotation to 90, 180 and 270 degrees.

3.7 Model

The model structure that was used was inspired by a model developed by Simonyan and Zisserman, commonly known as VGGNet [44]. VGGNet has a uniform structure, as it consists of repetitive convolution layers followed by a pooling layer, before

ending in fully connected layers to produce the outputs.

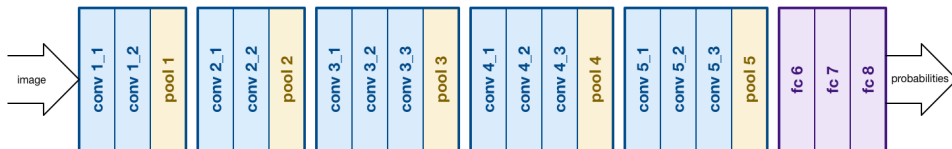


Figure 3.8: Structure of VGGNet, illustration from Medium [45]

The model created follows the same structure as VGGNet, with a smaller depth. The model structure is shown in Figure 3.9.

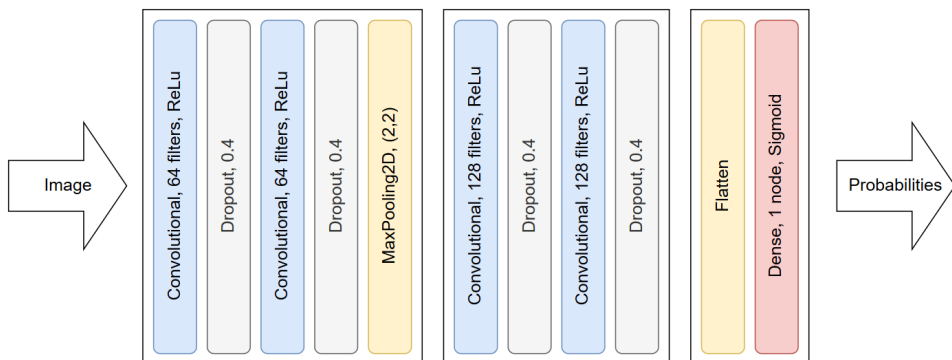


Figure 3.9: Structure of reduced CNN model

3.7.1 Training

During training, dropout was added after each convolutional layer in order to reduce overfitting. Dropout randomly ignores some of the nodes in the model, which makes the training process noisy. This hinders the network layers from co-adapting to correct mistakes from previous layers, which makes the model more robust [46].

The training of the model was set to stop early if the loss began to increase on the validation set. It was configured to allow two epochs of increasing validation loss before it reverted to the lowest loss and saved that model. As a result, the model could train until it got a good result without the added work of manually setting the amount of training epochs.

3.8 Image input and preprocessing

As the dead roe has clear characteristics, it was decided to do pre-processing of the images in order to find and suppress areas that are known to not contain roe. By doing this, the process becomes more specific as the neural network only needs to do predictions on areas that were suspected to contain dead roe.

3.8.1 Thresholding

In order to reduce the area that is fed into the CNN, simple thresholding was applied. This is a pointwise operation applied to pixels that do not match with a given limit. In this case, all pixels under the given limit were reduced to zero. This filters out the dead roe, as these have a higher pixel intensity as seen in Figure 3.4. In order to determine the thresholding limit, a histogram was generated from one of the images. The histogram is a plot over the pixel intensity in a given image, and is shown in Figure 3.10.

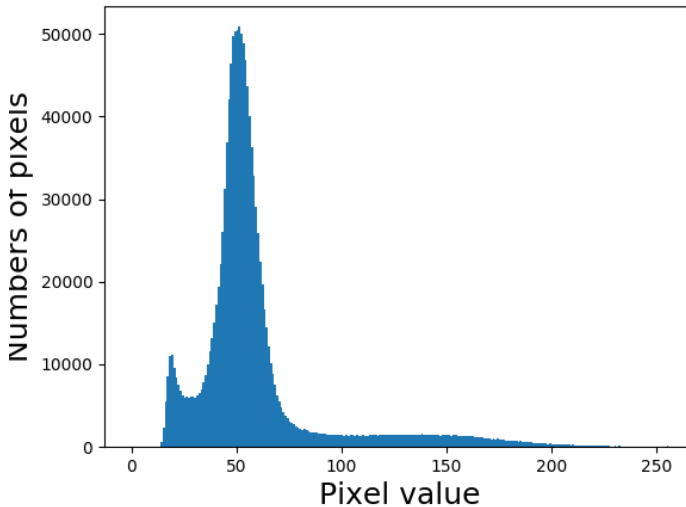


Figure 3.10: Histogram of image

By setting the thresholding value to 100, the dead roe is filtered out as seen in Figure 3.12a and 3.12b.

3.8.2 Edge detection and contouring

After the image is thresholded, the areas that are remaining could contain dead roe. The next task was to determine where the bounding boxes are around these areas, in order to have smaller images that are used with the CNN. The areas are defined by using canny edge detection [47]. This algorithm has four steps, listed below.

1. Noise Reduction

As edge detection is sensitive to noise in the image, a 5x5 Gaussian filter is used to smooth the image. The result of this can be seen in Figure 3.12c.

2. Finding Intensity Gradient of the Image

A Sobel kernel is used to find the derivatives in both horizontal and vertical direction. This is used to find the edge gradient and direction for each pixel.

3. Non-maximum Suppression

Non-maximum suppression removes pixels that are not a part of the edge by finding the local maximum. Any pixel that does not pass this check is suppressed.

4. Hysteresis Thresholding

The last step is to determine what are true edges by thresholding and connectivity. Any edge with gradient above the max-threshold is considered to be an edge, and those under min-threshold is discarded. Any gradient between the thresholds must be connected to edges above max-threshold to be considered a true edge. The thresholds and edges are illustrated in Figure 3.11.

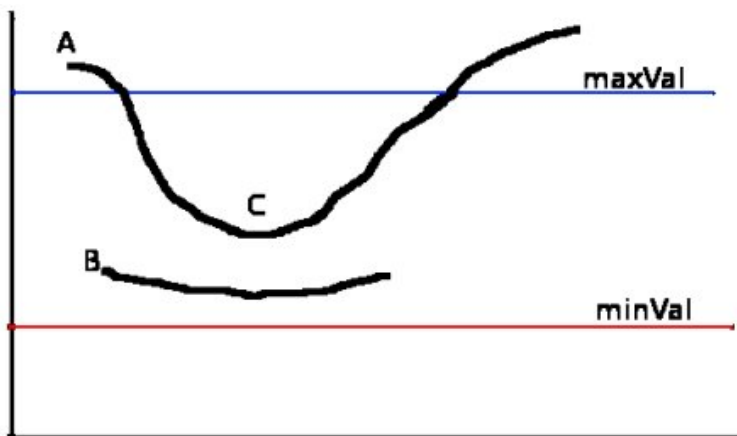
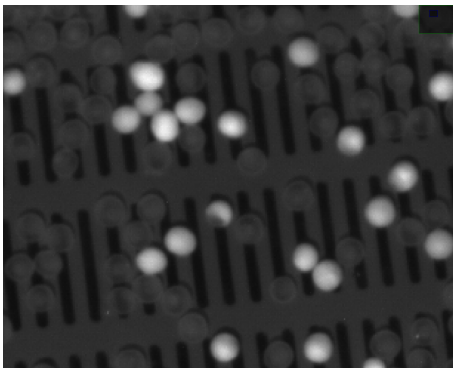
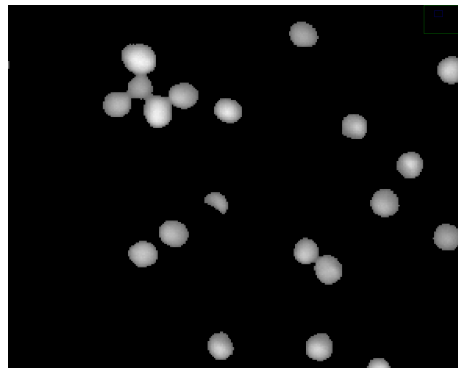


Figure 3.11: Hysteresis, illustration from Docs [48]

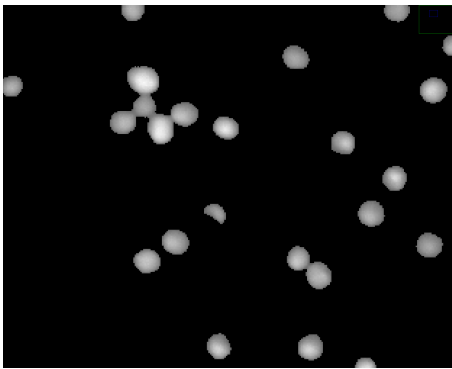
The results from the Canny edge detection is shown in Figure 3.12d. The bounding boxes was defined by finding the minimum and maximum values of x and y for each edged area, the results is shown in Figure 3.12e.



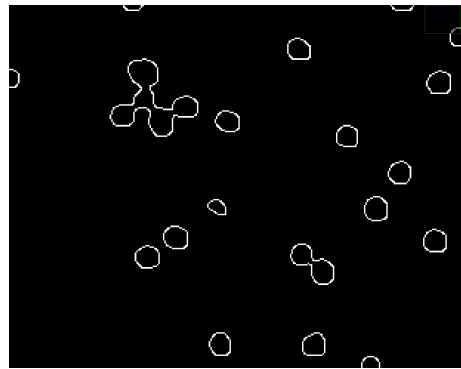
(a) Original image



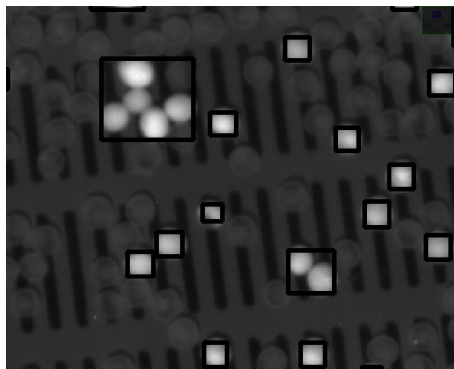
(b) Thresholded image



(c) Smoothed image



(d) Edge-detected image



(e) Suggested areas

Figure 3.12: Image operations in preprocessing of image

3.9 Reduction of hitboxes

After running the detection model, the output is given as a list that contains the coordinates, height, width and the probability given by the model. As a consequence of using the sliding window, hundreds of proposals are generated as the model can give multiple hits on the same roe. An example of this is seen in Figure 3.13, where single detection's is shown in the lower right corner and overlapping detection's are shown in the top left corner. In order to provide a better localisation, the overlapping predictions needed to be removed.

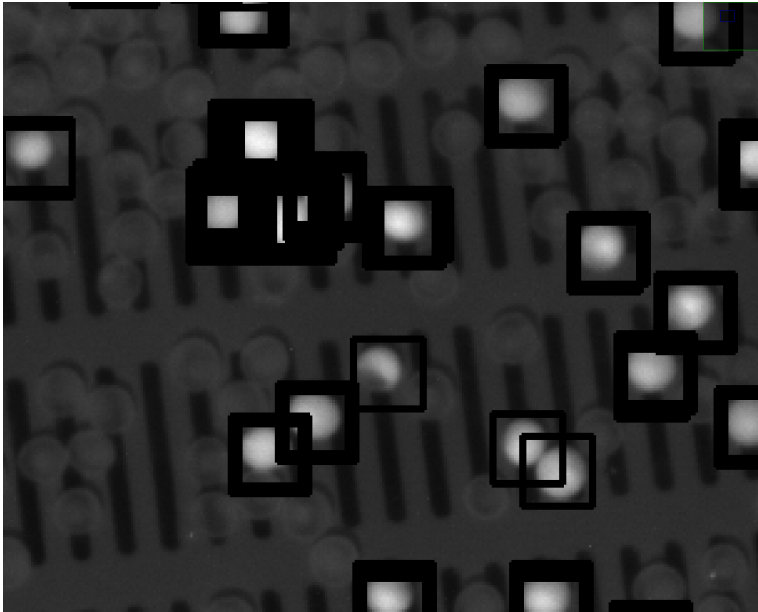


Figure 3.13: Multiple detections of same roe

3.9.1 Non-maximum suppression

A solution to the overlapping prediction problem was to use Non Maximum Suppression (NMS), which is a key post-processing step in many computer vision applications [49]. The implementation used were the felzenszwalb method, a iterative, greedy NMS [50].

The process starts with the iteration through the list of predictions, where the predictions that overlap in a certain degree are put into buckets. From these buckets, the prediction with the highest probability score is selected and the rest is removed from the prediction list. The results are shown in Figure 3.14, where the

overlapping predictions were reduced significantly.

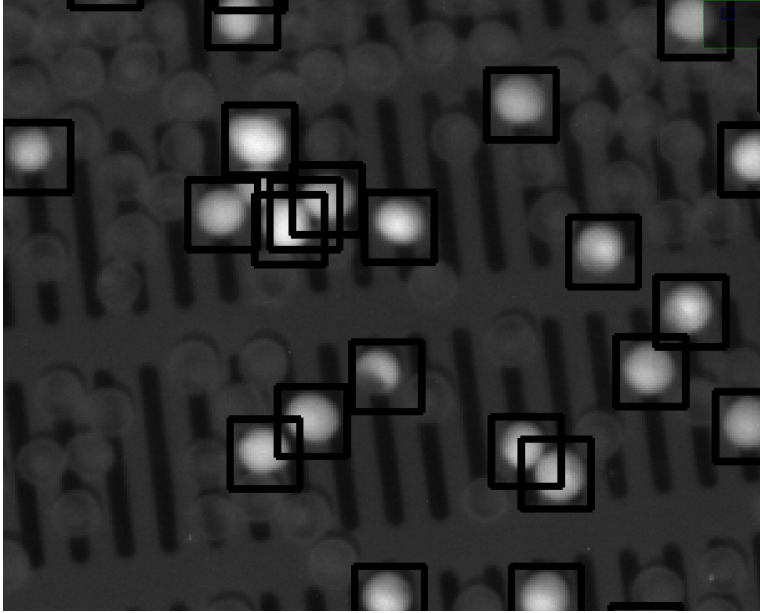


Figure 3.14: Reduced number of boundingboxes after GreedyNMS

The known downsides with a greedy implementation of NMS is threefold. Firstly, the highest predictions with the highest probability may not be the prediction that is best centred on the roe. Secondly, the NMS can remove the predictions of roe that is too close to each other, as the bounding boxes may be too overlapping. Thirdly, the NMS does not remove false positives that may occur.

In the first case, the negative impact is reduced as the CNN is used to confirm the presence of a dead roe in the centre of the image, and the bounding boxes is configured after the largest size of the roe. As a result of this, the bounding boxes covers the detected roe in its entirety.

The second case is the most problematic of the downsides. Specifically in the first round of the roe removal, there can be dead roe close to each other because of the large amount. This is in some form reduced by allowing a large amount of overlap, but not entirely prevented.

The third case is not considered a problem, as the CNN is the responsible part for confirmation of dead roe, and this is thus not the responsibility of the NMS.

Chapter 4

Optimisation and refinement of previous work

4.1 Test setup

In order to measure the computation time and gather information about the distribution of computing time within the program, a profiling library was used. The profiling library chosen was cProfile. Three different hardware systems were used in this project, a desktop computer, a laptop and the RPi 4. The desktop computer was used for computationally heavy tasks such as training the models, while the laptop was used for code development and algorithm testing. The RPi 4 was used as it is the intended hardware platform for the final product.

To be able to compare the changes done in the software, a single test image was chosen to be used consequently throughout the project. This allows for the comparison of results without having any of the changing variables the use of different images will bring. The test image is shown in Figure 4.1 and has a resolution of 1280x1024 pixels. The image has a file-size of 791,1 Kb and contains 186 dead roe.

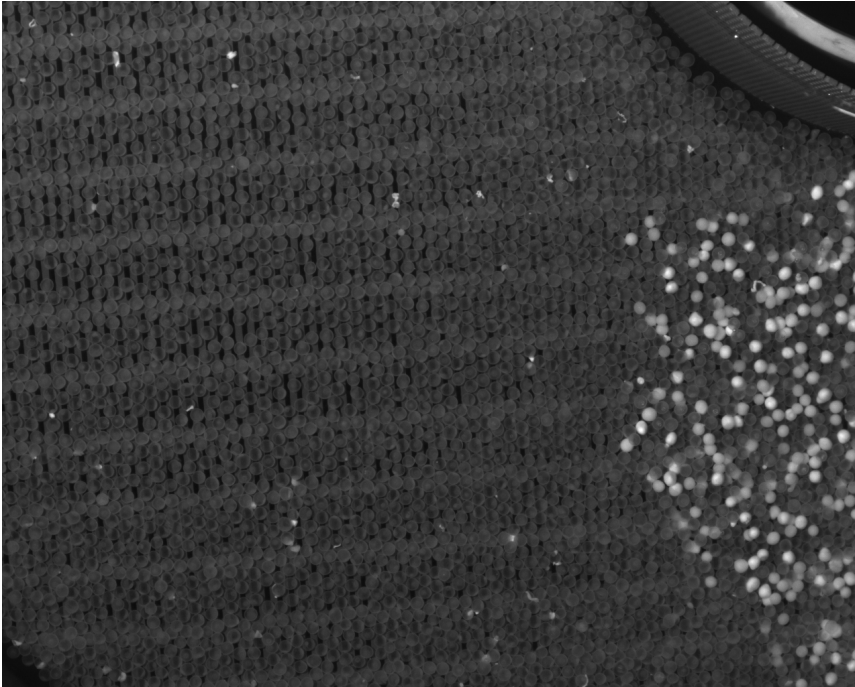


Figure 4.1: Image chosen for testing the different models and algorithms

4.1.1 Profiling of the software designed in previous work

The initial program makes 46 million function calls in 38.5 s, when executed on the laptop. Table 4.1 shows that 93% of the time is spent by using the neural net to perform predictions.

ncalls	cumulative time (s)	per call (s)	function
1	38.497	38.497	full program
1	36.736	36.736	detection program
9532	36.049	0.004	predictor

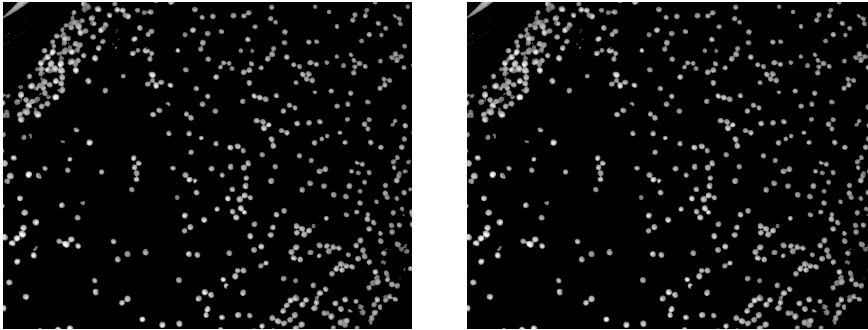
Table 4.1: Profiling results of initial software

When executed on the RPi, it runs 27 million function calls in 173.7 s. This is 4.5 times slower than on the laptop, and shows that the RPi is more limited by its processor speed while being more efficient with its function calls. The difference in the amount of function calls is likely due to the difference in system architecture and Operating System (OS), and is not suitable for cross-platform comparison.

4.2 Dynamic thresholding

The existing program used the method of static thresholding, where the value was found by making a histogram over the pixel values in an image and testing different thresholding values from the histogram. This was fine for development. However, the environment that the vision system will be used in is unknown and the light level may be changing while the robot operates.

In order to have a dynamic and thus a more robust thresholding, Otsu's thresholding was used [16]. This takes the changing light levels into account. On the test image, Otsu's method found a threshold value of 96, which is functionally equal to the static value of 100 found manually in the previous work. This indicates that the dynamic thresholding chooses reasonable thresholding values. The results from static thresholding and Otsu's thresholding on a randomly selected image is shown in Figure 4.2.



(a) Static thresholding

(b) Dynamic thresholding

Figure 4.2: Histogram and thresholding of test image

4.3 Removing overlap

A problem with the generated bounding boxes is that there is a degree of complete overlap, an example of this can be seen in Figure 4.3b. The detected roe in the lower left corner is overlapped by the bounding box containing the entire figure. This is unwanted as it leads to the neural network making predictions on the same areas of the image multiple times, and therefore using more time for no benefit.

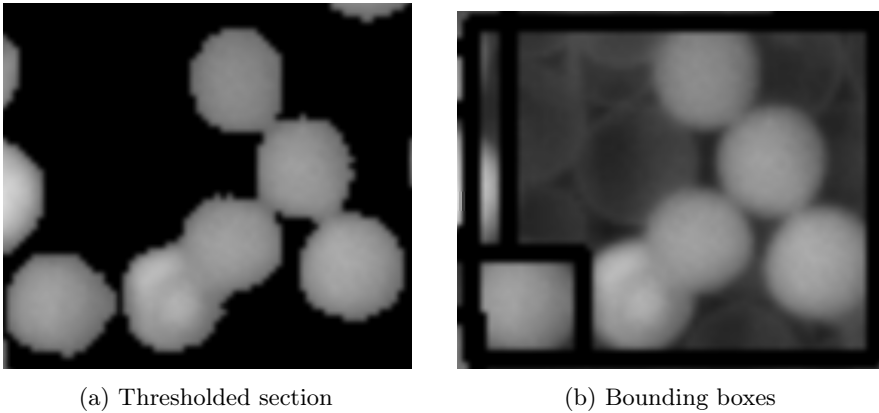


Figure 4.3: Example of large and overlapping bounding boxes

In order to remove the overlapping bounding-boxes, the `RETR_TREE` contour function call was replaced with `RETR_EXTERNAL`. As described in the documentation [51], this only returns the outermost contours that were found in the contour hierarchy tree. This reduces the amount of overlapping bounding boxes, but some still remain.

In order to remove the remaining overlapping bounding boxes, the boxes have to be compared. The original method worked by sorting the bounding boxes by size, and then comparing the biggest box with all the other bounding boxes. The bounding boxes located inside the biggest box were removed, and the process continued with the next bounding box in the list. The program structure was written as shown in Listing 4.1.

```
Sort list by area size
while list > 0:
    for box in smaller_boxes:
        if box is within the largest bounding box:
            remove box
```

Listing 4.1: Psuedocode, removal of contained bounding boxes

As this is in practice a nested for loop, the time complexity is $O(n^2)$. This is too computationally expensive as this operation will be performed on a large amount of bounding boxes.

It was found that the problem of removal of the remaining overlapping bounding boxes are similar to the NMS, only filtered on total overlap instead of a partial overlap. Thus Tomasz Malisiewicz's implementation of NMS (fastNMS) was studied and modified [52]. The major change is that fastNMS removes the inner for loop by

use of numpy's maximum and minimum functions. This allows for truth checks on numpy arrays, which is significantly faster than individual if checks. The code is shown in Listing 4.2

```
Sort list by area size
while list > 0:
    keep largest box and remove it from list
    Do check and remove all boxes that are completely
        overlapped by the last kept box
```

Listing 4.2: Psuedocode, faster removal of contained bounding boxes

The check is performed by doing four comparisons. As an example, two bounding boxes are shown in Figure 4.4 with the variables used.

1. Find maximum of 0 and $X1-X2$
2. Find maximum of 0 and $Y1-Y2$
3. Find maximum of 0 and $X2+w2-(X1+w1)$
4. Find maximum of 0 and $Y2+h2-(Y1+h1)$
5. Multiply the found values together and remove from list if result is larger than 0.

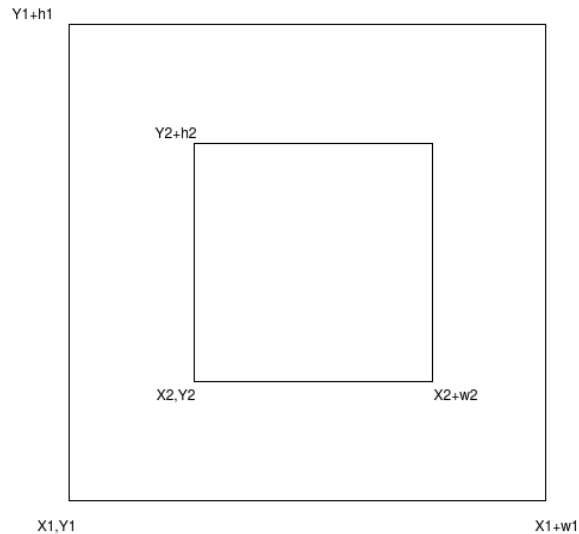


Figure 4.4: Variables of two overlapping bounding boxes

4.4 Reduction of large bounding boxes

The bounding boxes shown in Figure 4.3 are still larger than preferred. When large bounding boxes are passed to the detection program, they are processed by use of a sliding window. This results in a large computing cost for large bounding boxes. By counting the pixels in Figure 4.3a, it was found that 55.2% was empty space. By further processing, the large bounding boxes can be divided into smaller bounding boxes, and thus reduce the execution time needed.

This was done by iteratively eroding the thresholded area defined by the large bounding boxes, to reduce the edges of the dead roe until the entire roe is eroded away. By doing edge detection on the eroded images in reverse order, the centre of the overlapping roe may be found as is should be the last area to be eroded away.

The following process was performed on the chosen section of the thresholded image.

1. Opening, eroding followed by dilation to reduce noise.
2. Distance transform, find the distance from the remaining pixels to the closest 0 value.
3. Save the largest found distance.
4. Increase the erosion of the image with 10% of the largest found distance for each iteration
5. Edge detect and keep results that has a certain centre distance from other kept results and a size below a chosen value.
6. Repeat from step 4 until iteration 10.

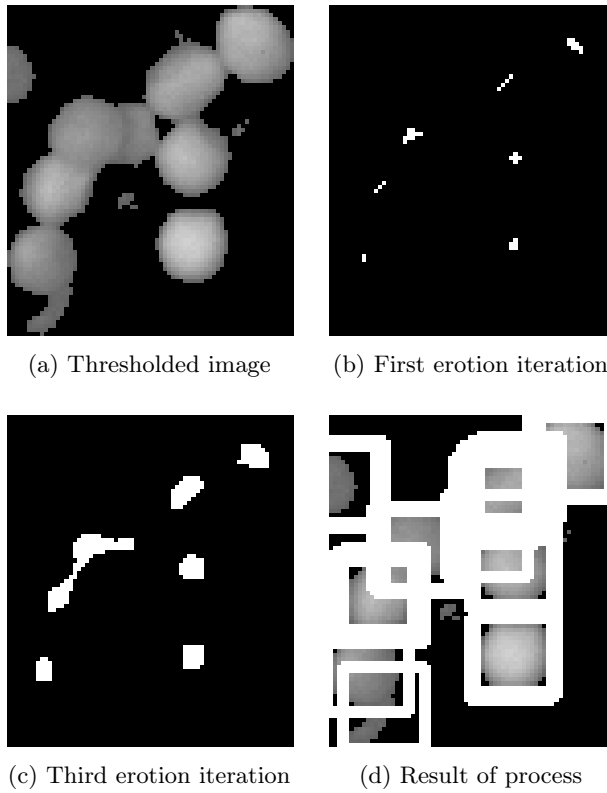


Figure 4.5: Example of erosion process on large bounding box

From Figure 4.5d it can be seen that multiple close proximity bounding boxes that have been identified over the same roe. These are reduced by doing another sorting where bounding boxes are greedily removed if the X and Y coordinates are closer than 3 pixels. The results from this is shown in Figure 4.6, and a reduction in bounding boxes can be seen, without any significant reduction in the coverage of the dead roe.

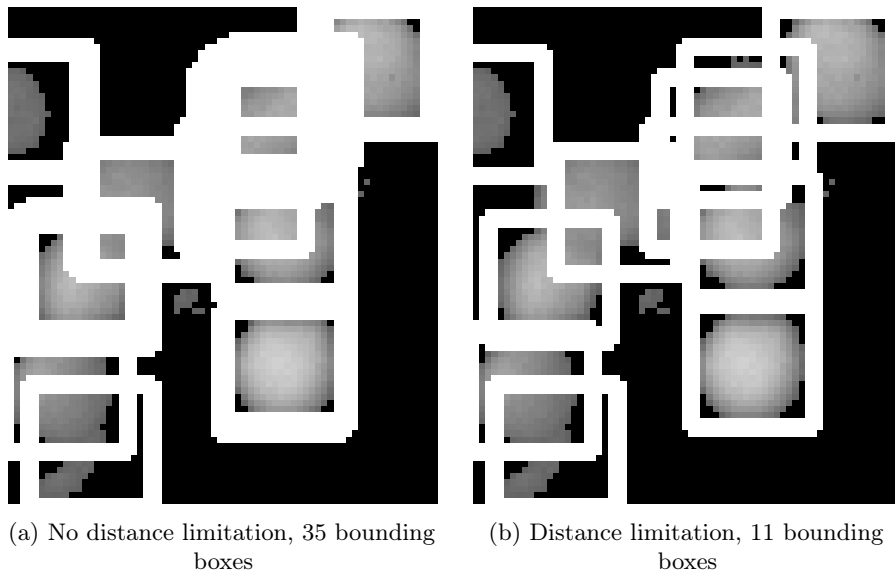


Figure 4.6: Reduction of bounding boxes with close proximity

4.5 Finding centre of roe

It is wanted to define a centre of the roe, as this is beneficial when removing the roe. A simple approach would be to use the centre of the bounding box that was predicted to contain dead roe. However, the roe may not be centred in the box, which would give less than ideal results. To identify the centre of the roe, the image contained by the bounding box is iteratively eroded in the same fashion as the method used in Section 4.4. As the CNN was used to make predictions, the bounding box is confirmed to contain at least 1 roe as there may be overlapping roe.

4.6 Increased processor utilisation with multiprocessing

When running the program on the RPi, it was discovered that full processor utilisation was not achieved. The view of the RPi's processes, found by the use of the Linux top command, showed that about 53% of the processor capacity was used [53]. As the profiling in Table 6.4 shows, 97.7% of the work in the detector program is done when the model is working. The timing of the program was performed with

the time function, as cProfile interacted poorly with the multiprocessing library.

The multiprocessing library in python uses subprocesses instead of threads. These subprocesses are isolated from each other, which allows for the loading of multiple instances of the same model. The part of the program that runs in parallel is the predictor, as shown in Figure 4.7 with 2 parallel predictors.

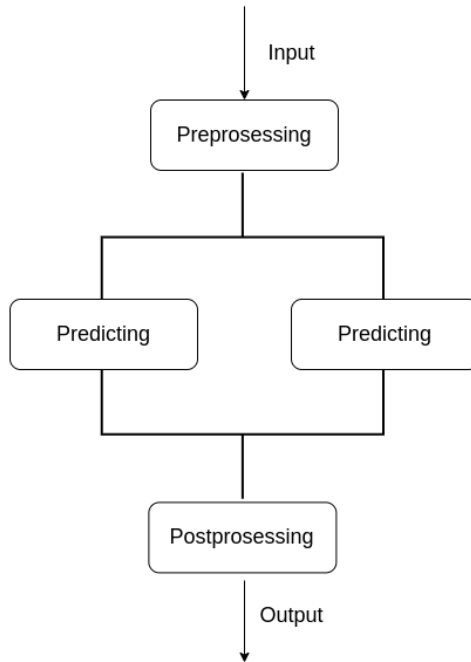


Figure 4.7: Program flow with two parallel predictors

4.7 Tensorflow lite

By converting the neural network into the TF-Lite format, the network is optimised for use on embedded devices and will in theory get a shorter execution time without any effect on the accuracy. The neural network was converted to a TF-Lite model by the use of the converter class in the Tensorflow Python API as described in the documentation [54]. The model used in testing is the original model that was tested in Section 4.1. The tests were performed on the RPi. The results of using the converted model is shown in Table 4.2.

Model	Time (s)	Calls	Nr of predicted roe	CPU usage
Original	75.155	12058034	176	48%
TF-Lite	36.988	1548835	176	70%

Table 4.2: Results from tensorflow and tensorflow-lite versions of original model

To ensure that the accuracy of the converted TF-Lite model was unchanged, the output-lists generated from both models were compared and found to be identical. The time spent was reduced to half, or a 50.7% reduction without affecting accuracy.

As TF-Lite requires the input image to be provided as an array, some processing had to be done before the prediction. It was therefore of interest to check if higher performance could be achieved by doing this work as preprocessing. The results shown in Table 4.3 shows that the predictor still uses 93.7% of the time in the detection program, while the CPU utilisation remained at 70%. It was therefore decided to test multiprocessing of the TF-Lite converted model as well, in the same manner as shown in Figure 4.7.

ncalls	cumulative time (s)	per call (s)	function
1	37.138	37.138	full program
1	30.670	30.670	detection program
3816	28.741	0.008	predictor

Table 4.3: Profiling results of the Tensorflow Lite model

By use of the time module, the execution time was found to be 31.7 s with 2 processes and 30.2 s with 3 processes. The CPU usage was 90% and 96% respectively. It was therefore decided to use three processes, as this gives a 59.9% reduction in time compared to the original, single-process model.

4.8 Analysis of neural network

In the previous work, the model that was used in the neural network was based on the VGGNet [44]. Smaller images and simpler models execute faster since it requires less computing operations. It was therefore decided to see how much it was possible to simplify the model and reduce the image quality, before the negative impact on the accuracy of the predictions outweighs the increase in speed.

In order to do this, 144 model-structures were created with the original model as inspiration. The new model-structures ranged from 1→6 convolutional layers, 0→2 max pooling layers, 1→2 dense layers and a $2^{0\rightarrow3}$ multiplier on the layer size. The model-structures were each trained on images with resolution in the even numbers

range of 2x2 to 20x20 pixels. The images was created by scaling down the original image sets. This resulted in 10 trained models per model-structure, or 1440 total models. The models was trained in the same way as described in Section 3.7.1, with dropout layers after each convolutional layer and with the models training until the loss began to increase on the validation set.

The trained models were tested on a validation image set, with the results saved in order to create confusion matrices. The models were also timed on the same test image, in order to evaluate the speed and accuracy of the model. The testing and timing were all performed on the same desktop computer, so the speed will differ from when the models are used on the RPi. The results show that the speed of the models increase with smaller input images as shown in Figure 4.8.

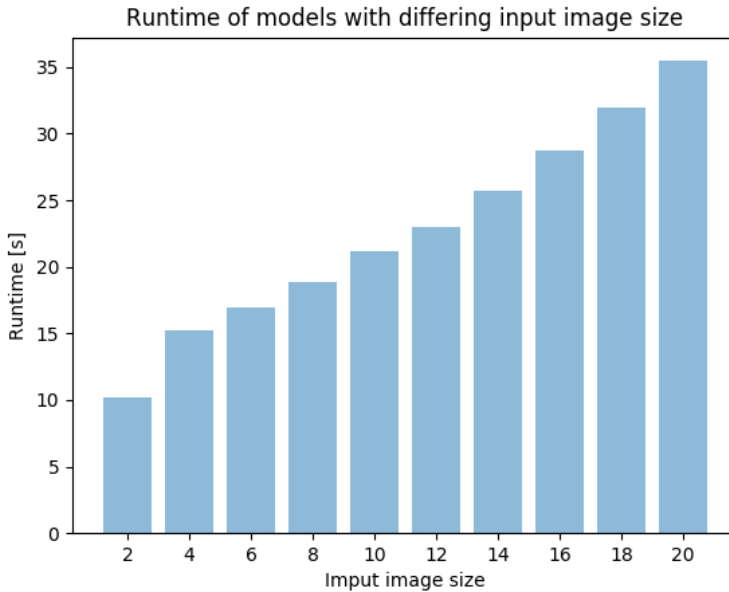


Figure 4.8: Model speed by input image resolution, averaged across all models

By combining the amount of false positives and false negatives, the total amount of errors is found. A histogram was plotted and is shown in Figure 4.9. This shows that a significant amount of the models has one or more errors, more specifically 1082 of the 1440 trained models.

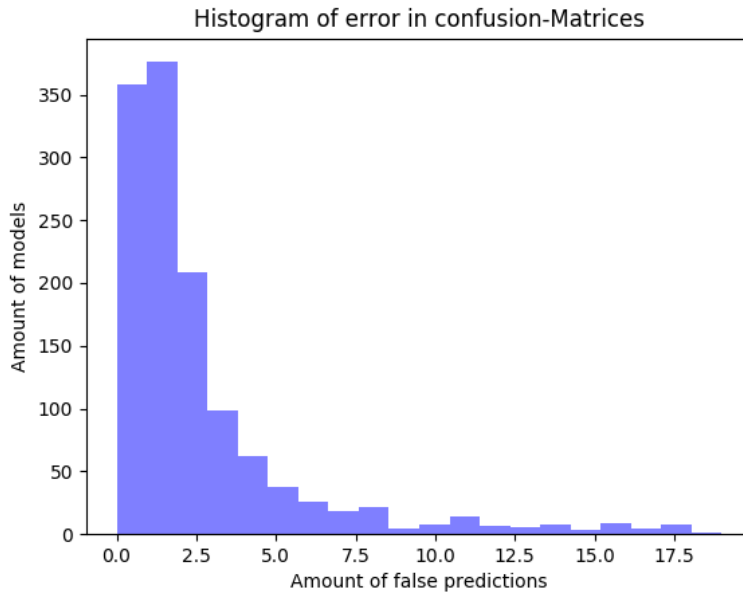


Figure 4.9: Histogram of models by amount of errors

The first step is to remove the model-structures where the 10 trained versions all had 1 or more prediction errors on the validation set. This removed 57 of the model-structures. The accuracy of the remaining models can be shown by the use of a boxplot, with the models grouped by the input image resolution. This is plotted in Figure 4.10. The plot for models with an input image resolution of 2x2 pixels is omitted as models had a high error rate that made the rest of the models hard to compare in the boxplot. Models with input image resolution of 2x2 pixels will not be considered further as all models gave some amount of false predictions. The boxplot shows that there are varying amounts of errors in the outliers with a lot of the models-structures scoring 0 wrong predictions. As it is the models that have 0 wrong predictions that will be considered, another method is needed to compare the remaining 87 model-structures and 9 input image resolutions.

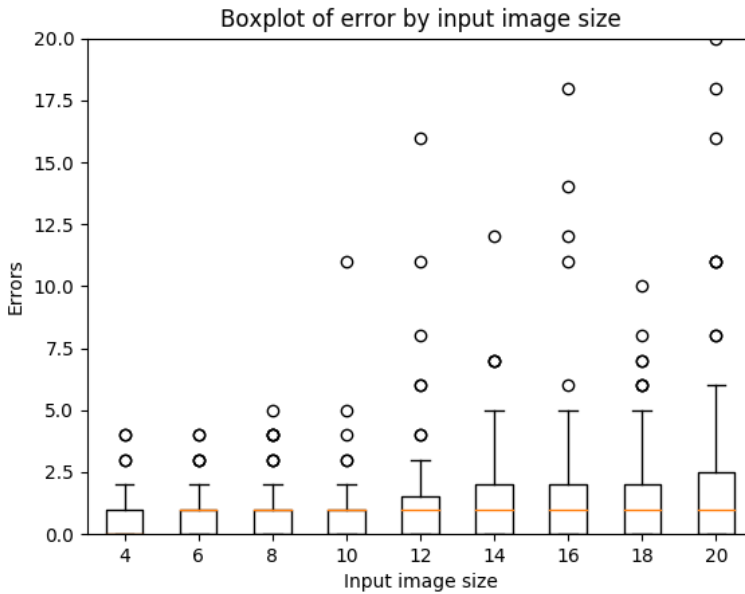


Figure 4.10: Boxplot of errors by input image pixel size

A new image was manually labelled with the use of the COCO-annotator tool [3]. The predictions from the models and the labels were compared to generate confusion matrices with pixel values. It was then possible to create weight functions in order to compare models with different traits.

Three different weight functions were used to pick three models, with the functions shown in the list under. The first only weighted the true positives, but the second and third functions also weighted the wrong predictions. The second function weight false positives and false negatives equally, while the third function weighted the false positives more than the false negatives as the removal of healthy roe is unwanted.

1. truePositive
2. truePositive - falsePositive - falseNegative
3. truePositive - 0.5 · falsePositive - 0.3 · falseNegative

The first weight function returned model-structure 16 with a input image resolution of 8x8. The second function returned model-structure 96 with a input image resolution of 10x10, with the third function returning model-structure 96 again, but with input image resolution 12x12. The model-structures are shown in Figure 4.11.

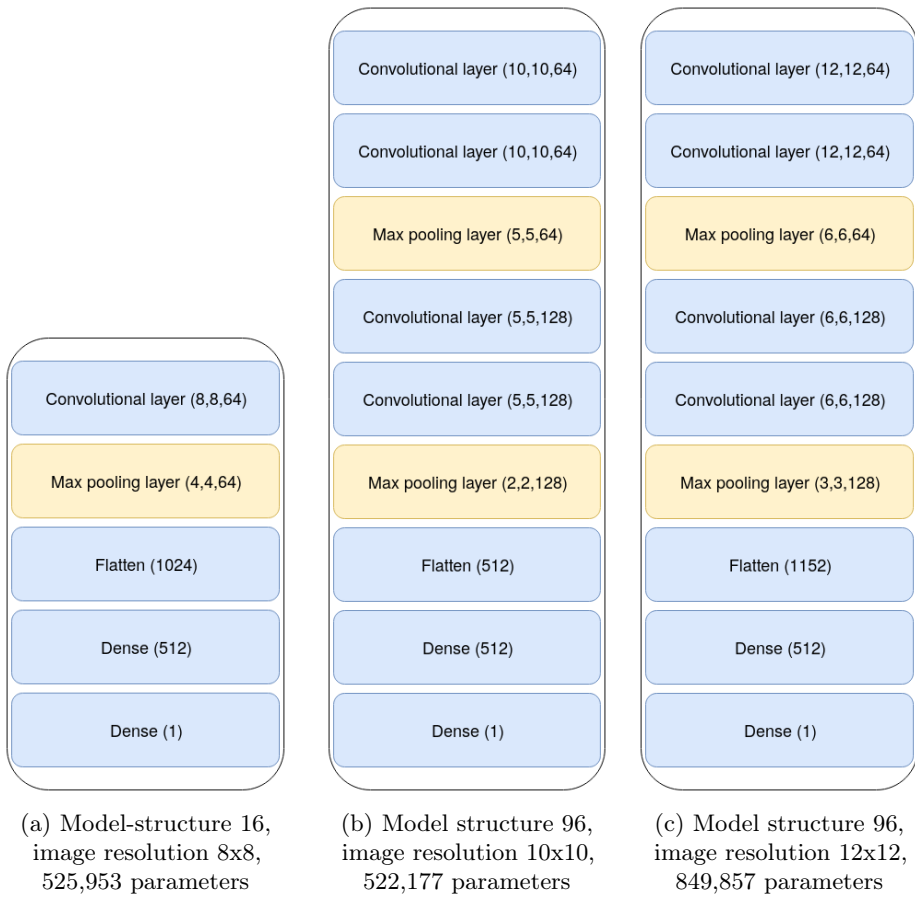


Figure 4.11: Structure of chosen models

Chapter 5

Methods

5.1 Distance measurement

The identification of the roe is done with a single overhead-mounted camera. A consequence of the single camera setup is the inability to identify the depth placement of the roe. As the depth is important for the removal process, methods for finding the depth will be investigated. The first method will be an IR-sensor, while the second method will be a combination of an ultrasonic sensor and a laser.

5.1.1 Test setup

A cooking pot with a depth of 25 cm and a diameter of 30 cm was used to imitate the metal plates that are used as platforms within the hatchery tub. The tub has plastic walls, the pot differs from this as it is completely made of metal, but is deemed acceptable as it is the metal bottom that is important for testing. The water height used in the tests ranged from 6 to 10 cm, as this is within the range the hatchery tubs water depth varies [8]. The cooking pot used is shown in Figure 5.1.



Figure 5.1: Cooking pot used to test distance measurement

5.1.2 Infrared sensor

The IR-sensor chosen was a Parallax analogue optical sensor with a range from 10-80cm [55]. It works by measuring the time of flight of the infrared light emitted from the sensor. It is important to note that the speed of light differs in air and water, so this needs to be taken into consideration. The sensor uses a supply voltage of 5V and returns an analogue signal in the range 0-3.2V [56]. The analogue signal corresponds with distance as shown in Figure 5.2.

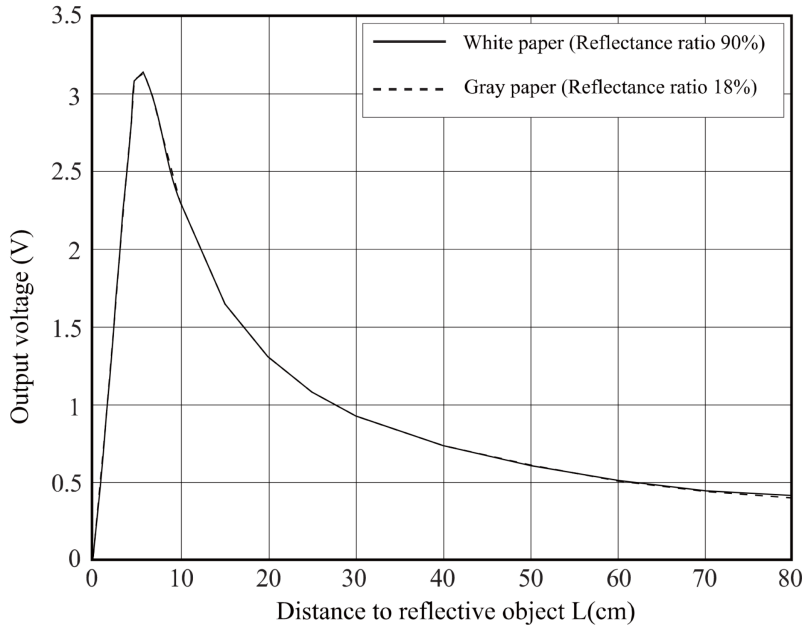


Figure 5.2: Distance measuring characteristics (output), from Sharp [56]

5.1.3 Ultrasonic sensor

The HC-SR04 ultrasonic sensor has an operating voltage of 5V [57]. It works by sending a burst of ultrasound at 40kHz and registering when the sound returns after it has bounced off an obstacle. By multiplying the measured time with the speed of sound, the travel distance of the ultrasonic sound is found. This is then divided by two, as it is the distance to the obstacle that is of interest.

5.1.4 Laser

The laser is a 1mW red line laser, and is used in conjunction with the camera [58]. A laser that is mounted with a known angle v a distance L from the centre of the camera can be used to determine the depth of the water, if used in conjunction with Snell's law and a known height above the water surface.

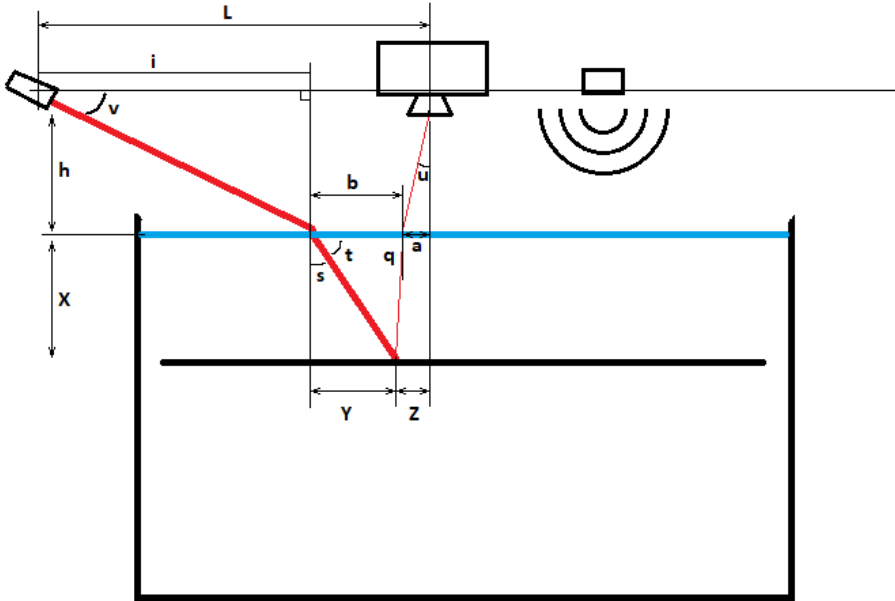


Figure 5.3: Example of measurement of depth

By using the angle v and height h found by the use of the ultrasonic sensor, the length i can be found as shown in (5.1). The angle v is also used in combination with Snell's law to find the angle s and subsequently the angle t .

$$i = h \cdot \tan(v) \quad (5.1)$$

$$s = \arcsin\left(\frac{n_{\text{air}}}{n_{\text{water}}} \cdot \sin(90 - v)\right) \quad (5.2)$$

$$t = 90 - s \quad (5.3)$$

The Ximea Mq013MG 1.3MP camera used has a pixel resolution of $5.3\mu\text{m}$. By using the amount of pixels found in the distance from the centre of the image to the reflected laser, in other words the distance a , we get the physical distance on the camera's image sensor. This can be used in combination with the focal length of the lens f to find the angle u . This is done by finding the triangle formed between the lens and the sensor, as shown in Figure 5.4 and (5.5).

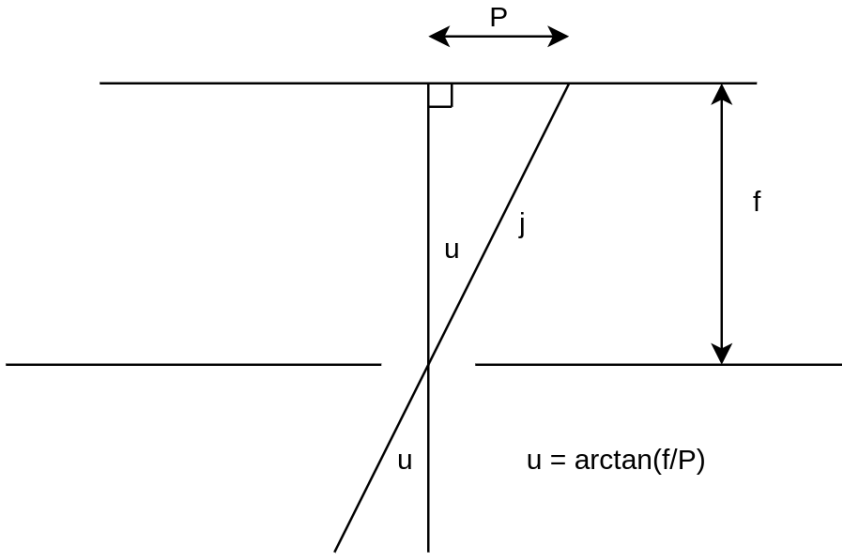


Figure 5.4: Pinhole camera approach

$$j = \sqrt{P^2 + f^2} \quad (5.4)$$

$$u = \arccos\left(\frac{f}{j}\right) \quad (5.5)$$

As the triangles formed by P, f and a, h is formal triangles, the length a can be found as shown in (5.6).

$$a = \frac{P \cdot h}{f} \quad (5.6)$$

By applying Snell's law, the angle q can be found as shown in (5.7).

$$n_{air} \cdot \sin(u) = n_{water} \cdot \sin(90 - q) \quad (5.7)$$

$$q = 90 - \arcsin\left(\frac{n_{air}}{n_{water}} \cdot \sin(u)\right) \quad (5.8)$$

With i, L and a as known lengths, b is found. Then the length of the underwater

beam, called ub , can be found using the sine formula as shown in (5.9).

$$ub = \frac{b \cdot \sin(q)}{\sin(180 - t - q)} \quad (5.9)$$

Finally, the depth X can be found by simple trigonometry as shown in (5.10).

$$X = ub \cdot \cos(s) \quad (5.10)$$

Detecting laser in image

The camera was mounted 103 cm above the ground by duct-taping it to a table. The laser was mounted by the same means a distance of 45 cm from the camera with the same height, at an approximately 30 degrees angle. Two images were taken, one with and the other without the laser activated. The images are shown in Figure 5.5.

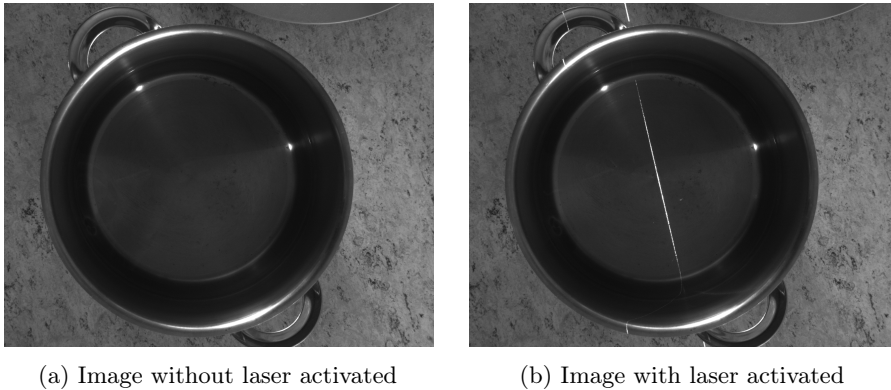


Figure 5.5: Test-setup of laser-depth detection

The images were compared using Structural Similarity Index (SSIM) to determine the exact discrepancies between the images [59]. This gives a score of the similarity between the pictures as well as an image that shows the difference with disparity highlighted in black. The similarity score is given as a value between -1 and 1, and was found to be 0.951 in the two images. The difference image is shown in Figure 5.6.

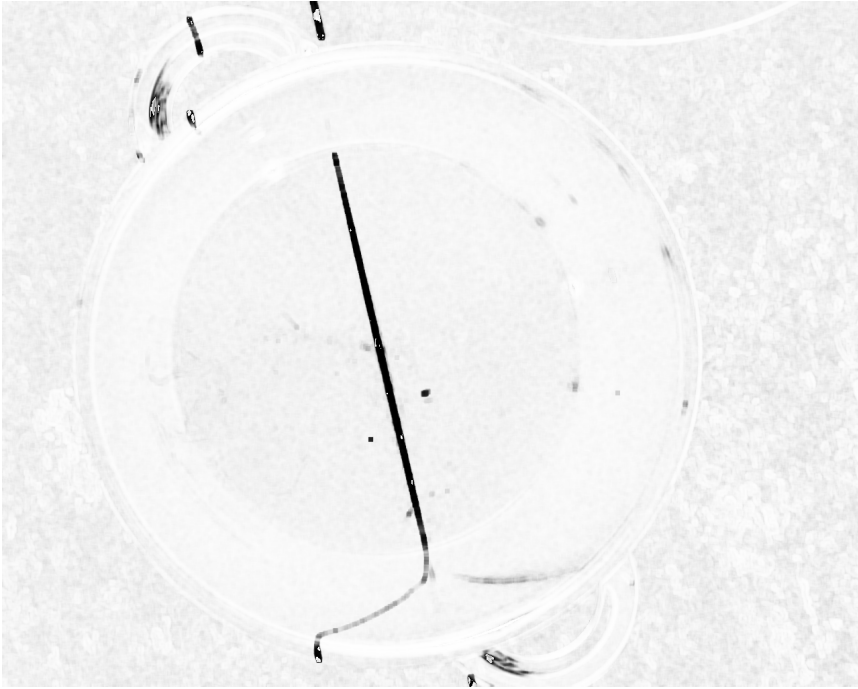


Figure 5.6: Difference between the images shown in Figure 5.5

In order to separate out the light noise that occurred, the image is thresholded with the use of Otsu's method. Edge detection was performed with the Canny edge detection algorithm, before the lines were found by Hough line transformation. This resulted in the detection of the two edges of the laser line, with the centre found as the average of the two lines. The code for edge detection and line transformation is shown in Listing 5.1. The results of the edge detection and line transform is shown in Figure 5.7.

```
edges = cv2.Canny(thresh, 75, 150)
lines = cv2.HoughLinesP(edges, 0.5, np.pi/180, 30, maxLineGap
    =250)
```

Listing 5.1: Detection of laserline

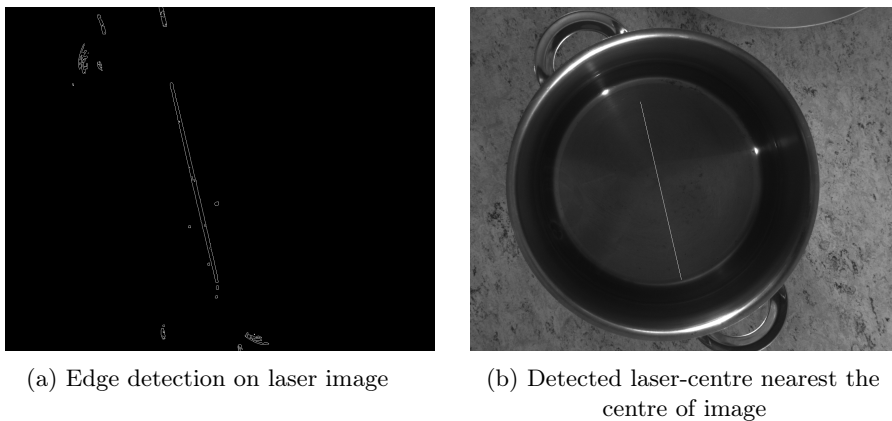


Figure 5.7: Detection of laser

The distance between the laser line and the centre of the image in the x -axis is found by a polynomial curve fitting of the line, which gives the function for x in the form $x = a \cdot y + b$. This finds the x -location of the laser line in the centre of the y -axis, and the distance from the centre of the image to the laser line is found to be 80 pixels. The code is shown in listing 5.2.

```

coefficients = np.polyfit([y1,y2], [x1,x2], 1)
a = coefficients[0]
b = coefficients[1]
#x = a\cdot y + b
distance = a*int(nolaser.shape[0]/2)+b-int(nolaser.shape[1]/2)
distance = np.round(np.absolute(distance))

```

Listing 5.2: Finding pixel distance from center of image

5.2 Peripheral components

The RPi and its peripherals need to be connected together, but the peripherals and the RPi will not necessarily be mounted in close proximity to each other. The General-purpose input/output (GPIO) on the RPi has a operational voltage of 3.3V with the peripherals having operating voltages of 12V, 5V and 3.3V. In addition to the dedicated power supply for the RPi, a 24V power source is available for use from the robotic system.

In order for the RPi to function with the peripherals, the following must be achieved.

1. Voltage level conversion

2. Logic level conversion
3. Enable RPi to control the peripherals

5.2.1 Power regulation

The available power is a 24V source, that needs to be regulated to 12V. This is achieved by using a Recom DC-to-DC converter that lowers the 24V voltage down to 12V [60]. The regulator can supply 1 A, and are supplied with bypass capacitors to remove eventual high frequency noise.

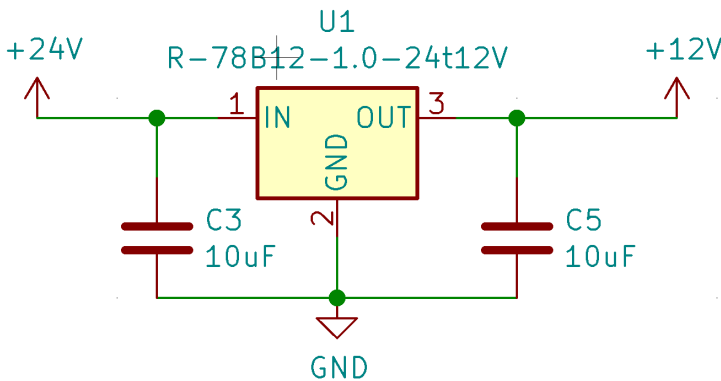


Figure 5.8: DC-to-DC converter with bypass capacitors

The 5V power is supplied from the RPi. The schematics show that the 5V output pins of the RPi are directly connected to the RPi's dedicated power supply, and is therefore not limited to the 12mA that the rest of the GPIO pins are rated for [61]. The RPi documentation states a power draw of 1.25A at load, which gives a 1.75A limit for the RPi peripherals [62]. The power draw of 15mA from the ultrasonic distance sensor and the 25mA from the laser is well within this limit, even with the maximum 0.9A Universal Serial Bus (USB)3.0 power draw. However, it is not likely that the USB power draw will be that high as the camera used only has a 0.18A power draw [63].

5.2.2 IR-light control

The IR-light source operates at 12V with a 0.3A current. To control the light, a relay is used. It is rated to handle 2A and up to 60VDC, which is higher than the the needs of the lights source. As the relay requires a 30mA and a 5V voltage, the

control signal is first raised to 5V through a metal–oxide–semiconductor field-effect transistor (MOSFET), which in turn drives another MOSFET as a switch. The relay is supplemented with a flywheel diode to extend the lifespan of the relay, as the relay coils can create high voltages when the relay is closed without the diode [64]. The schematic is shown in Figure 5.9 with a bill of materials in Appendix C.

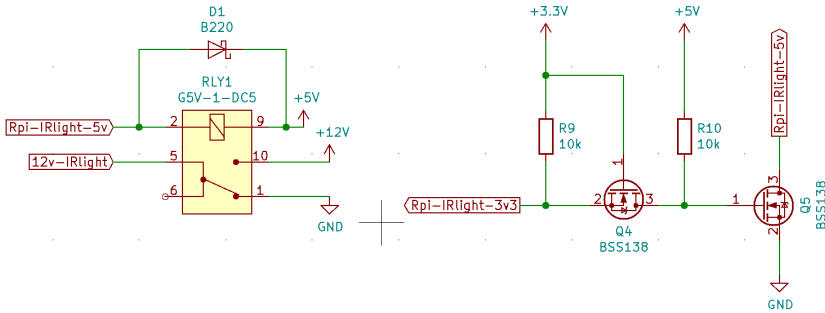


Figure 5.9: Relay and power control for IR-light

5.2.3 Laser

The 5V laser needs to be controlled by the RPi, as it is unwanted to expose the roe to the laser beam any longer than necessary. As the laser current is maximum 25mA with a average of 20mA, it is sufficient to control the power connection by a simple MOSFET switch circuit as shown in Figure 5.10.

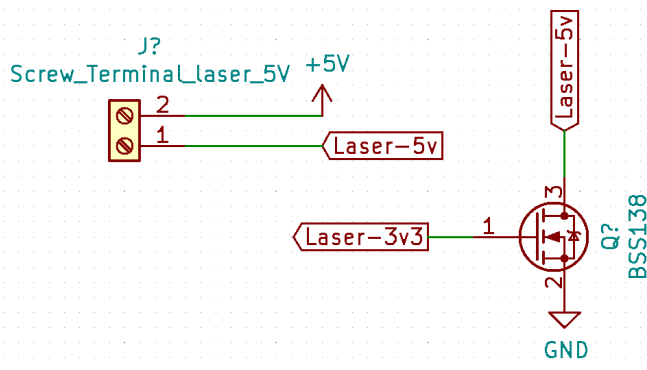


Figure 5.10: Power control of 5V laser

5.2.4 Ultrasonic distance sensor

The ultrasonic distance sensor needs to be supplied with 5V [57]. The input and output signals also operates at 5V, and therefore needs a logic level conversion in order to function together with the 3.3V RPi. The 5V output signal is converted to a 3.3V signal by use of a voltage divider, consisting of 10k Ω and 20k Ω resistors. In order to raise the input signal from 3.3V to 5V, a MOSFET is used. This is shown in Figure 5.11.

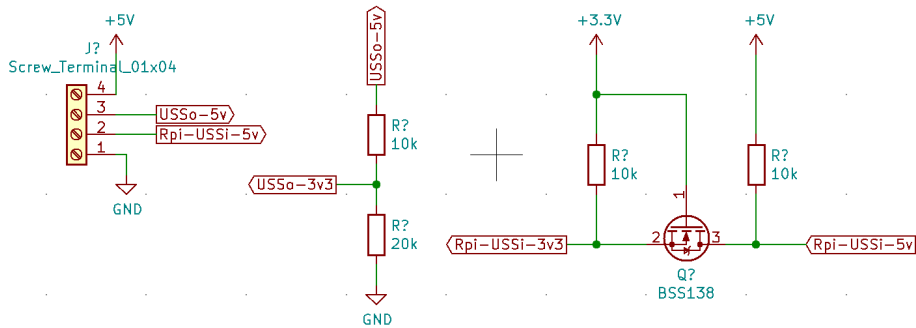


Figure 5.11: Logic level conversion and powering of the 5V ultrasonic distance sensor

Chapter 6

Results and discussions

6.1 Software

Initially, the software had an execution time of 173.7 s on the test image shown in Figure 4.1. After the changes done to optimise the execution time, this was reduced to 13.6 s. Because cProfile wasn't able to profile the software while the multiprocessing library was in use, it is challenging to find the exact time used to initialise the software. When profiling with a single process as shown in Table 4.3, the initialising time is 6.5 s, so it is fair to assume that the initialisation for the multiprocessing software to be at least the same. This means that the active processing time on the test image is 7.1 s.

6.1.1 Removing overlap

The results from changing the `RETR_TREE` to `RETR_EXTERNAL` is a reduction from 46 million to 40 million function calls, which ran in 38.5 s and 32.5 s respectively. This change reduced the execution time with 15.6% compared to the results in Table 4.1. As seen in Table 6.1, the calls to the neural net is reduced from 9532 to 8134 calls.

ncalls	cumulative time (s)	per call (s)	function
1	32.488	32.488	full program
1	30.746	30.746	detection program
8134	30.195	0.004	predictor

Table 6.1: Profiling results with changed contour-function

Removing the remaining overlapping bounding boxes was done with a modified fastNMS algorithm, with the results shown in 6.2.

ncalls	cumulative time (s)	per call (s)	function
1	31.488	31.488	full program
1	30.060	30.060	detection program
7862	29.448	0.004	predictor

Table 6.2: Profiling results with remaining overlapping bounding boxes removed

Discussion of the results

The change from the RETR_TREE to RETR_EXTERNAL is essentially free, in the meaning that the work done by the contouring algorithm is the same while the output is changed from all contours to only the outermost contours. This gives a direct decrease in execution time, without having to do any work to obtain it.

The use of the modified fastNMS also gave a decrease in execution time, but not to the degree the change to the contouring algorithm did. This was expected as the most of the overlapping bounding boxes already was removed. The modified fastNMS was however found to be an efficient algorithm for this kind of problem, and gave a lower execution time.

6.1.2 Reduction of large bounding boxes

The profiling results after reducing the large bounding boxes are shown in Table 6.3. The program executed 35 million function calls in 28.7 s, which is a reduction of 11.6% from the results shown in Table 6.1 for a total time reduction of 25.4% when compared to the original program.

ncalls	cumulative time (s)	per call (s)	function
1	28.698	28.698	full program
1	26.947	26.947	detection program
7112	26.383	0.004	predictor

Table 6.3: Profiling result after reduction of large bounding boxes

As a result of restricting the proximity of the bounding boxes, the profiling results from this change showed a reduction to 19,4 million function calls in 16.5 s, corresponding to a 42% reduction from the results shown in Table 6.3, and a total time reduction of 57% compared to the original program.

The removal of overlapping bounding boxes in addition to the reduction of large bounding boxes reduced the amount of predictions from 7930 to 3816. This corresponds to a 51.8% work reduction for the neural network.

When profiled on the RPi, the program executes in 12 million function calls in 74.9 s.

ncalls	cumulative time (s)	per call (s)	function
1	74.880	74.880	full program
1	66.687	66.687	detection program
3816	65.159	0.017	predictor

Table 6.4: Profiling result after reduction of large bounding boxes on RPi

Discussion of the results

These changes removed the amount of work done on areas without dead roe by a significant amount. It also relies on an accurate thresholding of the image, as if roe is thresholded away, the roe wont be detected. However, this was considered to not be a problem as the IR-light causes the dead roe to be the brightest points in the image.

6.1.3 Increased processor utilisation with multiprocessing

Testing was performed with up to 4 parallel processes on the RPi, with the execution time, CPU usage and memory usage recorded. This is shown in Table 6.5. It was not tested with more processes, as the CPU usage reached 100% utilisation with 4 processes. The results were a 24.5%, 31.8% and a 33.6% reduction in execution time for 2, 3 and 4 processes respectively. This shows that there was a time benefit from using multiprocessing, but it comes with an overhead and diminishing returns.

Process	Total time (s)	CPU usage	Memory usage
1/1	110	52.7%	3.3%
1/2	83	37.8%	3.3%
2/2	83	37.7%	3.3%
1/3	75	30.6%	3.3%
2/3	75	29.6%	3.3%
3/3	75	29.3%	3.3%
1/4	73	24.3%	3.3%
2/4	73	23.8%	3.3%
3/4	73	23.8%	3.3%
4/4	73	23.8%	3.4%

Table 6.5: Results of testing multiprocessing with 1 to 4 processes

Discussion of the results

However, despite the overhead and diminishing returns, it is still worth using multiprocessing until full processor utilisation is achieved. A counterpoint to using four processes is that it is not a big increase from the performance of three processes, while it causes the highest amount of heat generation at maximum processor utilisation. If the RPi exceeds a certain temperature threshold, it responds with throttling the CPU core which reduces the performance [65]. It is therefore important that the RPi has a sufficient cooling system, preferably active cooling.

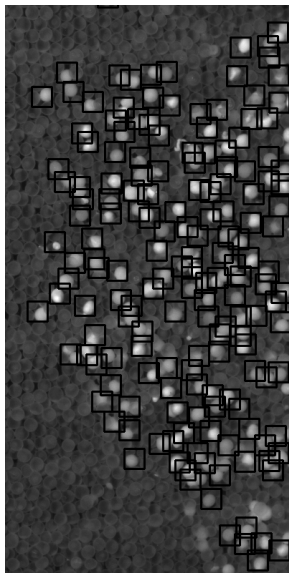
6.1.4 Analysis of neural network

Model-structure 16 had an minimal structure, while model-structure 96 was of a larger size. The models were timed on the test-image, with the result shown in Table 6.6.

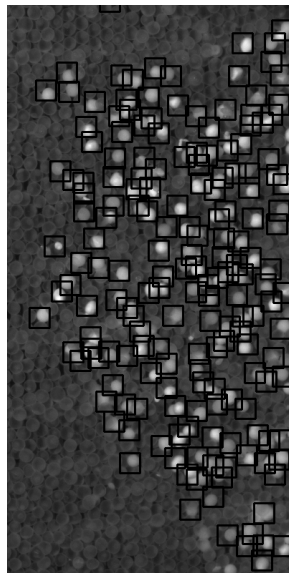
Model-structure	Image resolution	Time (s)	Calls	Nr predicted roe
Original	20x20	18.917	19486198	176
16	8x8	11.443	13946532	182
Original	8x8	13.567	19486089	181
96	10x10	15.401	20883998	165
Original	10x10	13.814	19487102	196
96	12x12	17.166	20882292	131
Original	12x12	14.608	19486123	186

Table 6.6: Results and timing of models

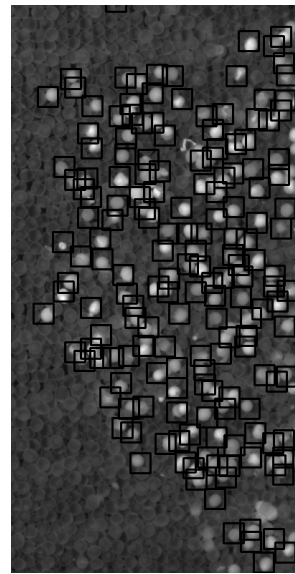
The original model-structure was also timed on the same test-image in order to do a comparison. The predictions are shown in Figure 6.1, with the images cut to show the relevant lower right part of the test-image.



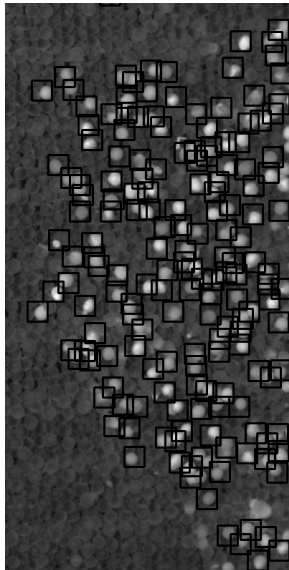
(a) Results from original model-structure, input image resolution 8



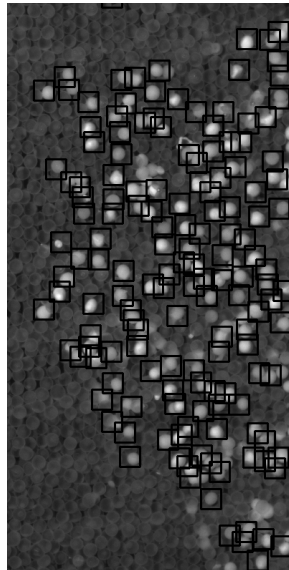
(b) Results from original model-structure, input image resolution 10



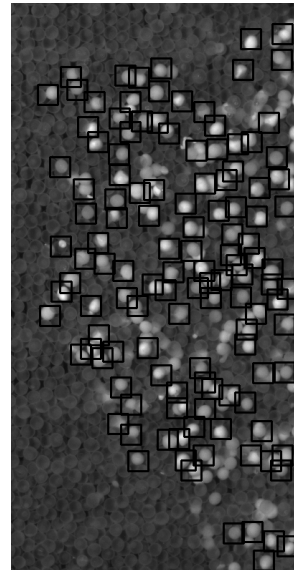
(c) Results from original model-structure, input image resolution 12



(d) Results from model-structure 16, input image resolution 8



(e) Results from model-structure 96, input image resolution 10



(f) Results from model-structure 96, input image resolution 12

Figure 6.1: Predictions from original and chosen models with same input image resolution

From Figure 6.1, it can be seen that model-structure 96 with input image resolution of 12x12 pixels misses a lot of the roe. The same is true to a lesser degree when the input image resolution is 10x10 pixels. The most accurate of the models is the original model-structure with an input image resolution of 10x10 pixels. As the accuracy of the model is an important factor, it was decided to continue with the original model-structure, but with a reduced input image resolution of 10. A test on a image with a high amount of dead is shown in Figure 6.2, where the roe is detected with a high accuracy.

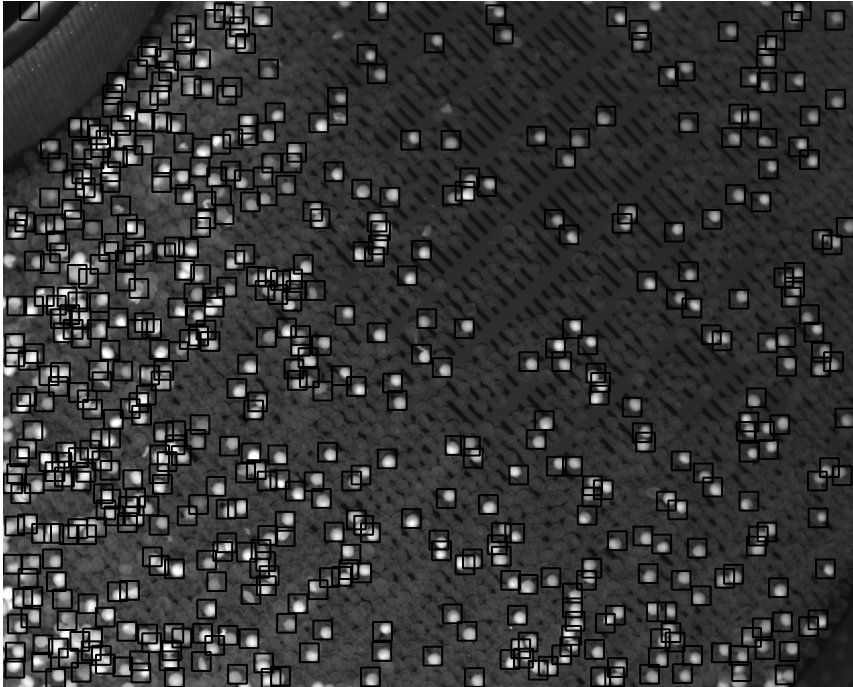


Figure 6.2: Prediction from original model-structure with input image resolution of 10x10 pixels

Discussion of the results

By testing a large amount of models with varying input image resolution, it was possible to see how the model complexity and input image resolution affected the detection of the roe. The simpler models mostly had an issue with identifying the walls of the tub as dead roe if it was sufficiently bright. The more complex models were harder to separate on other properties than speed, as they mostly performed with a high degree of accuracy. The reason for the small difference in the accuracy of the higher complexity model-structures was likely due to the low complexity in the identification task. In simple terms, the neural networks only needed to identify

that a object was round and that it has a sufficient brightness. The difficulty of manually evaluating the models led to the method of comparing the models with the use of weight functions, where the speed was not accounted for was tried. This was to find the models that were considered to be the most accurate to compare against the original model-structure, with the plan of considering the execution time afterwards. It was discovered that the original model-structure performed better than the found models, with input image resolution of 10x10 pixels being a point where the accuracy of the original model still was high. The 10x10 pixel input image resolution is a large enough size that roe can be recognised, even if the roe does not perfectly fill the image. An example of a 10x10 pixel image resolution with a dead roe is shown in Figure 6.3. As a consequence of the reduced model input image resolution could be that a camera with a smaller resolution could be used, which could decrease cost.

Another consideration was that the placement of nearby roe will likely change when dead roe is removed. As a consequence, any problems with detection of overlapping roe will disappear. This could have allowed for the use of a simpler neural network, but as this was an assumption that has to be tested and may be conditional it was not considered during this project.

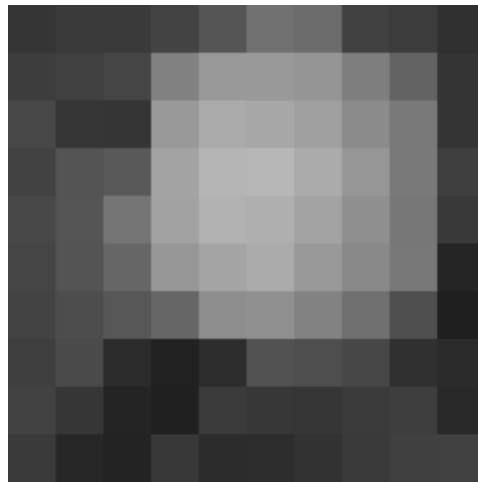


Figure 6.3: Example of 10x10 pixel input image that contains a dead roe

6.1.5 Finding centre of roe

This method for finding the centre of the dead roe worked well when the bounding box only contained one dead roe. However, when the bounding box contained two dead roe, it selected the roe with highest pixel intensity.

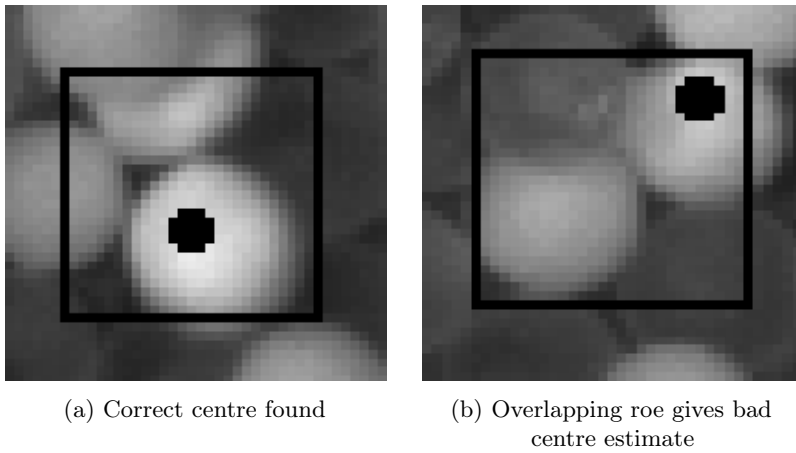


Figure 6.4: Examples of finding roe centre

When the algorithm is used on a larger image, it can be seen that the problem of overlapping roe is significant as some roe is marked multiple times as seen in Figure 6.5.

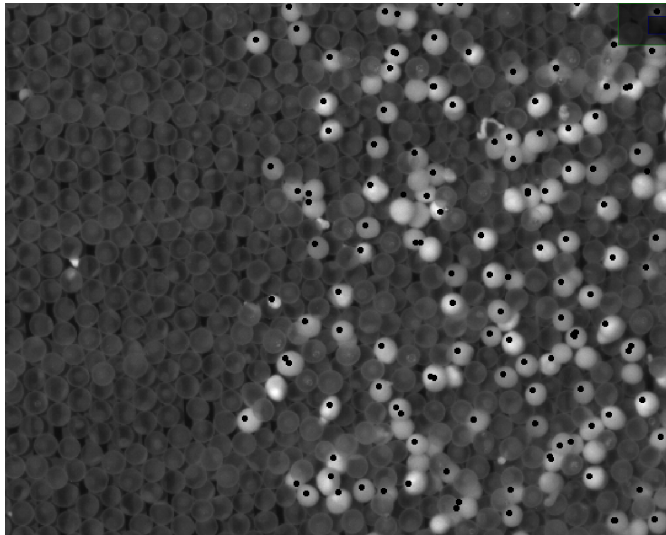


Figure 6.5: Results of detection and centre estimation on test image

Discussion of the results

The chosen method of erosion has the problem of selecting the roe with the highest pixel intensity when applied to bounding boxes containing overlapping roe. However, this problem is self-rectifying as when the roe that are marked multiple times are removed, the remaining roe will be marked correctly. This makes the solution functional, but it will likely affect the removal time of the roe if the removal strategy relies on pathfinding algorithms for the optimal removal order.

6.1.6 TensorFlow lite and multiprocessing

By converting the model chosen in Section 6.1.4, and using it in the 3-processes configuration, the timing module returns a timing of 13.6 s when timed on the test image. This is a reduction of 92.2% from the initial software. When timed on the image shown in Figure 6.2, the program used 25.0 s.

Discussion of the results

The use of TF-Lite gave a significant performance boost without affecting the accuracy of the model in a detectable degree during the tests. It is combined with the multiprocessing library to utilise the CPU fully, as the utilisation was about 70% with a single process. The reason for not achieving full processor utilisation with one process was hard to pinpoint and is still unknown. It was suspected that it was the preparation and loading of the data to the predictor that was the bottleneck. However, when the preparation was done beforehand, it did not affect the execution time in a noticeable way. If the processor utilisation could be increased with a single process, this would likely give a higher performance increase than multiple processes.

6.1.7 Discussion of the performance on the Raspberry Pi 4

A discussion point is whether the RPi 4 is a platform with sufficient computing power or if other alternatives such as cloud computing, a more powerful platform or a USB accelerator is necessary.

The software performs with a decent speed when timed on images with a significant amount of dead roe present. A 25 second execution time on a image is nowhere near a real time vision system, but is deemed sufficient for this application. Predictions on full images should only be used to gain an overview of the dead roe in an area, while only the parts close to the picking robot will be predicted during the removal process as this requires accurate and up to date placement information.

Another consideration is the possibility of using a camera with a lower resolution. This will lead to smaller pictures, and thus a shorter execution time. How big this reduction in time will be depends on how much the resolution is lowered. However, as the current speed is considered sufficient, any amount of decrease in camera resolution will also result in an accepted execution time.

6.2 Depth measurement

Two different methods of finding the depth placement of the roe without having to install equipment in the tub itself was tested. The first method is a time of flight sensor that relies on IR-light. The second method is a combination of a ultrasonic distance sensor and a laser to find the distance to the water surface and the water depth respectively.

6.2.1 Infrared sensor

The testing with the cooking pot was performed with two different distances, 66 cm and 32 cm, to the metal bottom (M). On each distance, measurements were taken with no water (W), 6 cm of water and 10 cm of water. Testing was also performed on a standard A4 sheet of paper (P) with the distance of 35 cm and 66 cm to ensure that the sensor was working properly. The results are shown in Figure 6.6. The results show that the sensor gave some spikes in the measurement, and also gave some variation in the measurements when water was present. It also had a significant error when measuring the distance to the metal pot, regardless if water was present.

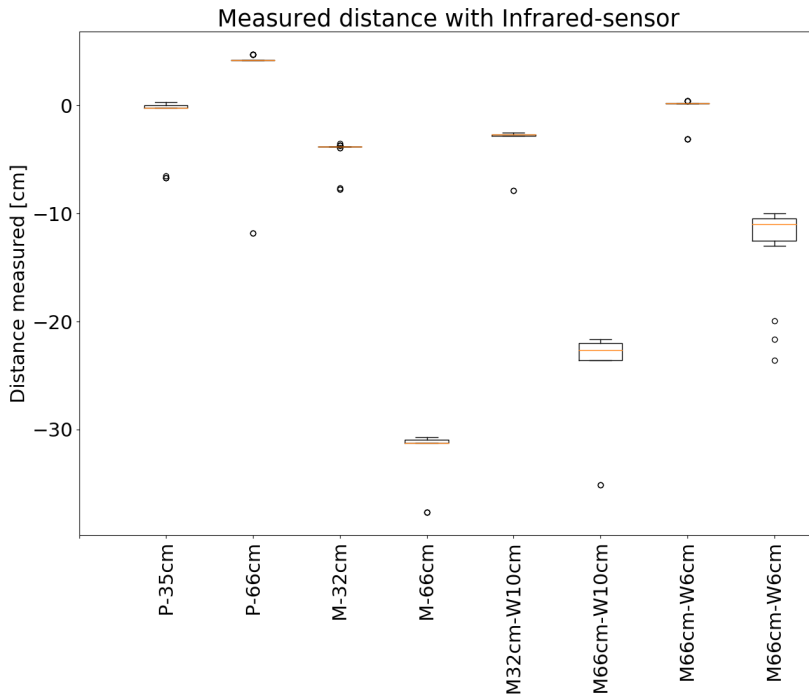


Figure 6.6: Results from testing the infrared sensor. P is paper, M is the metal surface, W is the water depth.

Discussion of the results

The IR-sensor interacted poorly with the metal surface, likely due to the reflecting properties of the metal. The assumption was taken that it is unwanted to paint or change the metal bottom of the hatchery tub, as there likely exists restrictions on what is allowed to do with equipment that handles food production. Due to this, it was deemed unsuited for use.

6.2.2 Ultrasonic sensor

The ultrasonic sensor was tested with differing heights and water levels. The results from this testing is shown in Table 6.7 and Figure 6.7.

Material	Distance	Water-level	Measured distance	Difference
Paper	112cm	-	111.41cm	0.59cm
Paper	69.5cm	-	69.48cm	0.02cm
Paper	36.5cm	-	36.95cm	0.45cm
Metal	34cm	6.5cm	29.02cm	1.7cm
Metal	68cm	6.5cm	61.65cm	0.15cm
Metal	68cm	10cm	58.91cm	0.91cm
Metal	35cm	10cm	25.39cm	0.39cm

Table 6.7: Results from testing of ultrasonic sensor

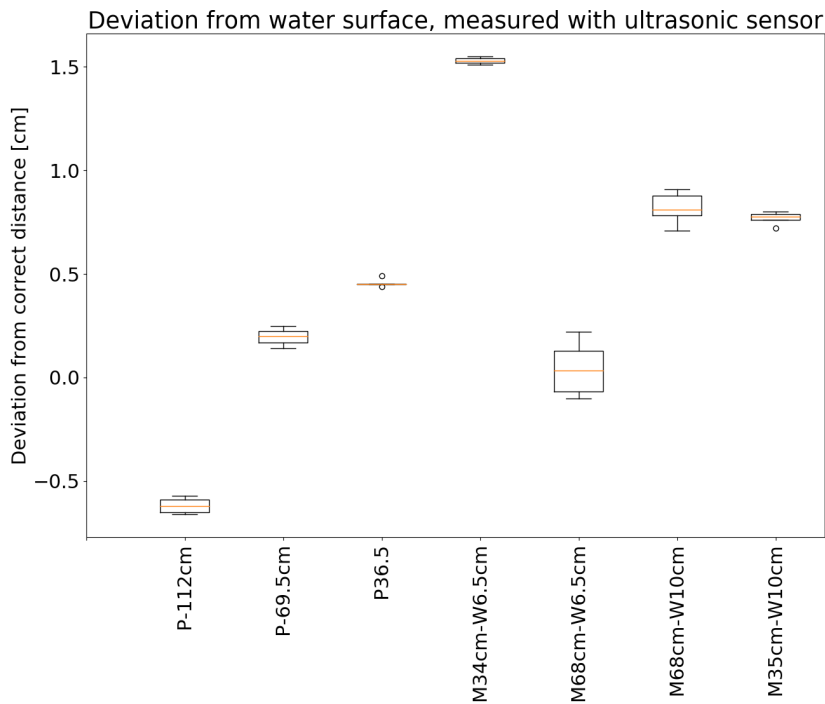


Figure 6.7: Boxplot over deviation in results from ultrasonic sensor test. P is paper, M is the metal surface, W is the water depth.

Discussion of the results

The testing shows that the ultrasonic sensor is capable of detecting the water surface. The error in the measurement is likely due to the crude test-setup as the sensor is

reported to have an accuracy of up to 3 mm [66]. However, the sensor accuracy is dependent on the RPi's GPIO read speed. Therefore it is preferred to use a library that has the GPIO implemented in C, in order to get a high accuracy.

6.2.3 Laser

Testing was set up with the laser mounted a distance of 33 cm from the camera, with an angle of 29.7° downwards from the horizontal line. The camera was mounted 45 cm above the bottom of the cooking pot, while the pot was filled with 9 cm of water. The results of the test is shown in Table 6.8



Figure 6.8: Example of the test-setup of laser depth measurement

Variable	Calculated value	Measured value	Deviance
a	2.8 cm	2.7 cm	0.1 cm
b	5.2 cm	4.3 cm	0.9 cm
i	25 cm	26 cm	1 cm
X	5.7 cm	9 cm	3.3 cm

Table 6.8: Results from testing of laser depth measurement

Discussion of the results

The calculated depth value deviated from the actual value with 3.3 cm, corresponding to a 36.6% error. This is likely due to the crude test-setup as this method requires high accuracy in its construction. The test setup should ideally be on an even surface, with the components mounted at as exact known distances as possible. The test setup used was mounted with tape, without any precision measuring equipment. As a consequence, the distances and angles found had a high degree of uncertainty. This made the setup unsuited for anything other than a proof of concept and to identify major problems. In addition to the inaccuracies of the test-setup, the laser and camera might have affected the results. A consequence of the laser line thickness was that it had no clearly defined centre. However, this was handled by detecting the edges and using the calculated centre. A laser with a more precise and thinner line thickness could increase the accuracy. Furthermore, the camera was likely not levelled, which would have a direct impact on the results as the calculations are based on a level camera.

This test showed that the laser measuring method could be functional given a high accuracy in its construction. However, it is likely impractical to both construct and maintain such an accuracy as the intended use for the system is to be moved between different hatchery tubs. Therefore, this method is not recommended.

6.3 Peripheral components

A Printed Circuit Board (PCB) was designed, but not produced and is therefore shown as an 3D rendering in Figure 6.9. However, the functionality of the PCB was tested using a breadboard and similar components. No problems were detected with this setup while testing.

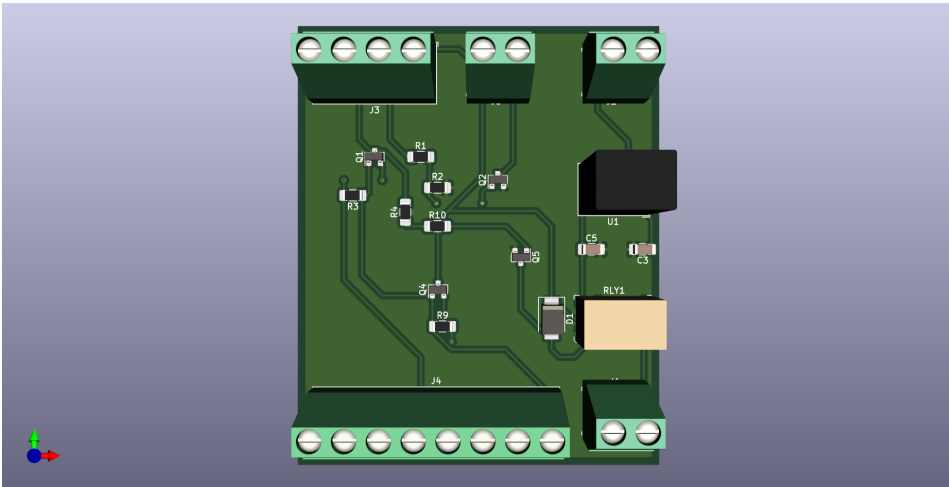


Figure 6.9: 3D rendering of the PCB

Discussion of the results

The circuit board is a simple board of minimal size, and was made to extend the RPi's capabilities. The screw terminals were chosen to allow for easy mounting of cables, as flexibility is often important when testing prototypes. The power regulator is rated to supply 1A, which covers the 0.3A load well. It was chosen to use a relay for controlling the IR-light, as a relay is a robust way to control the relatively high current that the IR-light needs.

The circuit should be redesigned with mounting holes to allow for easy mounting, or alternatively, a clip-in cover box can be designed and 3D printed. The schematic and list of components for the circuit can be seen in Appendix A and Appendix C respectively.

Chapter 7

Conclusions and future work

7.1 Conclusion

The optimisations shown in this rapport reduced the execution time of the software with 92.2%. This is seen as an acceptable reduction. It is important to notice that the software execution time differs depending on the amount of roe in the picture taken. The intended use is therefore to do a full prediction of an initial image in order to generate an approximate location for dead roe. New images should then be taken over the roe that will be extracted, with a new prediction done on a smaller area under the extractor to pinpoint the location of the chosen roe. Due to this, the Raspberry Pi is deemed a sufficiently powerful platform for running the software.

Of the depth measurement methods tested in this project, it was the combination of the ultrasonic sensor and laser that gave the best results. However, the calculated water depth deviated from the actual depth by a relatively large degree when measured with the laser. It is therefore recommended to investigate other methods for finding the depth before the vision system is integrated with the robotic system. Alternatively, the water depth can be manually measured.

In conclusion, the software has been significantly improved upon but the tested methods of determining the depth placement of the roe did not give a satisfactory accuracy.

7.2 Future work

Based on the results this project have found, future points of research and areas of improvements are suggested.

7.2.1 Depth measurement

The laser depth measurement can be tested with a more accurate test-setup. However, it is recommended that different methods for finding the depth of the roe is investigated. An example could be a pressure sensor that can be mounted on the side of the first of the stacked metal bottoms. An example sensor could be

the 10m ultra high pressure sensor from bluerobotics, that has a accuracy of 0.16 mm [67]. Alternatively, a MS5837-30BA pressure and temperature sensor can be implemented, that gives the possibility of tracking temperature in addition to the 2mm depth measurement accuracy [68].

GPIO libraries implemented in C

If the ultrasonic distance sensor is used in the depth measurement, the library used to read data from the sensor should be switched over to a library that is implemented in C for better performance. An example of such a library is the pigpio library [69].

7.2.2 Reduced camera resolution

As the model chosen has an input image resolution of 10x10 pixels, the camera originally used could be changed to a camera with a smaller resolution. This could reduce the cost of the final product, as high resolution cameras generically is expensive.

A suggestion for a setup to test minimal image resolution is the Blackfly S USB3 BFS-U3-04S2M-CS 0.4 MP camera in combination with a Tamron 8mm 1/1.8 inch C mount Lens.

7.2.3 Implementation with roe picking robot

Parallel with this project, another project that focused on designing a robot for removal of roe was done. The work in this thesis needs to be implemented into this robot.

7.2.4 Better roe centre estimation

The current centre estimation has significant problems with overlapping roe. It may be refined by the use of a better segmentation method.

References

- [1] Exported Salmon 2019 Statistics Norway. URL: <https://www.ssb.no/statbank/sq/10025725> (visited on 08/25/2019).
- [2] *The EU fish market 2016 edition*. eng. 2016 Edition. EU fish market 2016. Brussels]: [European Commission], 2016. URL: <http://bookshop.europa.eu/uri?target=EUB:NOTICE:KLAP16001:EN:HTML> (visited on 10/20/2019).
- [3] Justin Brooks. *COCO Annotator*. <https://github.com/jsbroks/coco-annotator/>. 2019.
- [4] FAO. *FAO Fisheries & Aquaculture - Fishery Statistical Collections - Global Aquaculture Production*. URL: <http://www.fao.org/fishery/statistics/global-aquaculture-production/en> (visited on 05/27/2020).
- [5] Red List. *Salvelinus alpinus (Arctic Char)*. URL: <https://www.iucnredlist.org/species/19877/9102572> (visited on 05/27/2020).
- [6] Chefs Resources. *Artic Char Culinary Profile*. URL: <https://www.chefs-resources.com/seafood/finfish/arctic-char/> (visited on 12/09/2019).
- [7] *The progressive fish-culturist*. eng. Washington, D.C., 1934.
- [8] Stian Aspaas. private communication. 2020.
- [9] Mwansa Mathilda Songe. *Pathogenicity and infectivity of Saprolegina species in Atlantic salmon (Salmo salar L.) and their eggs*. eng. Oslo, 2015. (Visited on 11/20/2019).
- [10] phys. *Killing fish egg fungus with a disinfectant*. URL: <https://phys.org/news/2015-02-fish-egg-fungus-disinfectant.html> (visited on 12/03/2019).
- [11] Packt. *Convolution on RGB images*. URL: https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781789613964/2/ch021v11sec21/convolution-on-rgb-images (visited on 09/14/2019).
- [12] Sondre Høglund. *Autonomous Inspection of Wind Turbines and Buildings using an UAV*. eng. 2014. URL: <http://hdl.handle.net/11250/261286> (visited on 12/01/2019).
- [13] Achim Sanjay Manikarnika. *A General Face Recognition System*. eng. 2006. URL: <http://hdl.handle.net/11250/258577> (visited on 12/01/2019).

-
- [14] Wikipedia. *Wikipedia, Erosion (morphology)*. URL: [https://en.wikipedia.org/wiki/Erosion_\(morphology\)](https://en.wikipedia.org/wiki/Erosion_(morphology)) (visited on 05/09/2020).
- [15] Wikipedia. *Wikipedia, Dilation (morphology)*. URL: [https://en.wikipedia.org/wiki/Dilation_\(morphology\)](https://en.wikipedia.org/wiki/Dilation_(morphology)) (visited on 05/09/2020).
- [16] Nobuyuki Otsu. “A Threshold Selection Method from Gray-Level Histograms.” eng. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66.
- [17] Richard Duda and Peter Hart. “Use of the Hough transformation to detect lines and curves in pictures.” eng. In: *Communications of the ACM* 15.1 (1972), pp. 11–15.
- [18] Medium. *Hough Lines Transform Explained*. URL: <https://medium.com/@tomasz.kacmajor/hough-lines-transform-explained-645feda072ab> (visited on 05/09/2020).
- [19] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity.” eng. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612.
- [20] A.C Brooks, Xiaonan Zhao, and T.N Pappas. “Structural Similarity Quality Metrics in a Coding Context: Exploring the Space of Realistic Distortions.” eng. In: *IEEE Transactions on Image Processing* 17.8 (2008), pp. 1261–1273.
- [21] MathWorks. *Structural similarity (SSIM) index for measuring image quality - MATLAB ssim - MathWorks Nordic*. URL: <https://se.mathworks.com/help/images/ref/ssim.html> (visited on 05/09/2020).
- [22] J J Hopfield. “Neural networks and physical systems with emergent collective computational abilities.” eng. In: *Proceedings of the National Academy of Sciences of the United States of America* 79.8 (1982), pp. 2554–2558. URL: <http://search.proquest.com/docview/74120818/> (visited on 12/01/2019).
- [23] Missinglink. *7 Types of Activation Functions in Neural Networks: How to Choose?* URL: <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/#commonnonlinear> (visited on 05/27/2020).
- [24] ujjwalkarn. *A Quick Introduction to Neural Networks*. August 2016. URL: <https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/> (visited on 12/01/2019).

- [25] Read The Docs. *Loss Functions*. URL: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html (visited on 12/06/2019).
- [26] Machinelearninguru. *Image convolution*. URL: http://machinelearninguru.com/computer_vision/basics/convolution/image_convolution_1.html (visited on 12/06/2019).
- [27] Computer Science Wiki. *Layers in a Neural Network explained - deeplizard*. URL: <https://deeplizard.com/learn/video/FK77zZxaBoI> (visited on 05/24/2020).
- [28] Computer Science Wiki. *Convolutional Neural Networks (CNN): Step 3 - Flattening*. URL: <https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening> (visited on 05/24/2020).
- [29] Computer Science Wiki. *Max-pooling / Pooling - Computer Science Wiki*. URL: https://computersciencewiki.org/index.php/Max-pooling/_Pooling (visited on 05/24/2020).
- [30] Pellissippi State Community College. *Refraction of Light (Snell's Law)*. URL: <http://www.pstcc.edu/nbs/WebPhysics/Exp%202006.htm> (visited on 05/21/2020).
- [31] Gregory Paul Baxter, Laurie Lorne Burgess, and Herbert Wilkens Daudt. "THE REFRACTIVE INDEX OF WATER." eng. In: *Journal of the American Chemical Society* 33.6 (1911), pp. 893–901.
- [32] Wikipedia. *Snell's Law*. URL: https://en.wikipedia.org/wiki/Snell%27s_law#/media/File:Snells_law2.svg (visited on 04/18/2020).
- [33] *Introduction to algorithms*. eng. Cambridge, Mass, 2009.
- [34] Tensorflow. *About Tensorflow*. URL: <https://www.tensorflow.org/> (visited on 05/08/2020).
- [35] Google. *FlatBuffers: FlatBuffers*. URL: <https://google.github.io/flatbuffers/> (visited on 05/29/2020).
- [36] Raspberry Pi Foundation. *Raspberry pi 4 hardware specifications*. URL: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/> (visited on 12/06/2019).
- [37] Komplet. *Raspberry pi 4, 4GB*. URL: <https://www.komplett.no/product/1133779/datautstyr/pc-komponenter/hovedkort/integrert-cpu/raspberry-pi-4-model-b-4gb-ram> (visited on 09/20/2019).

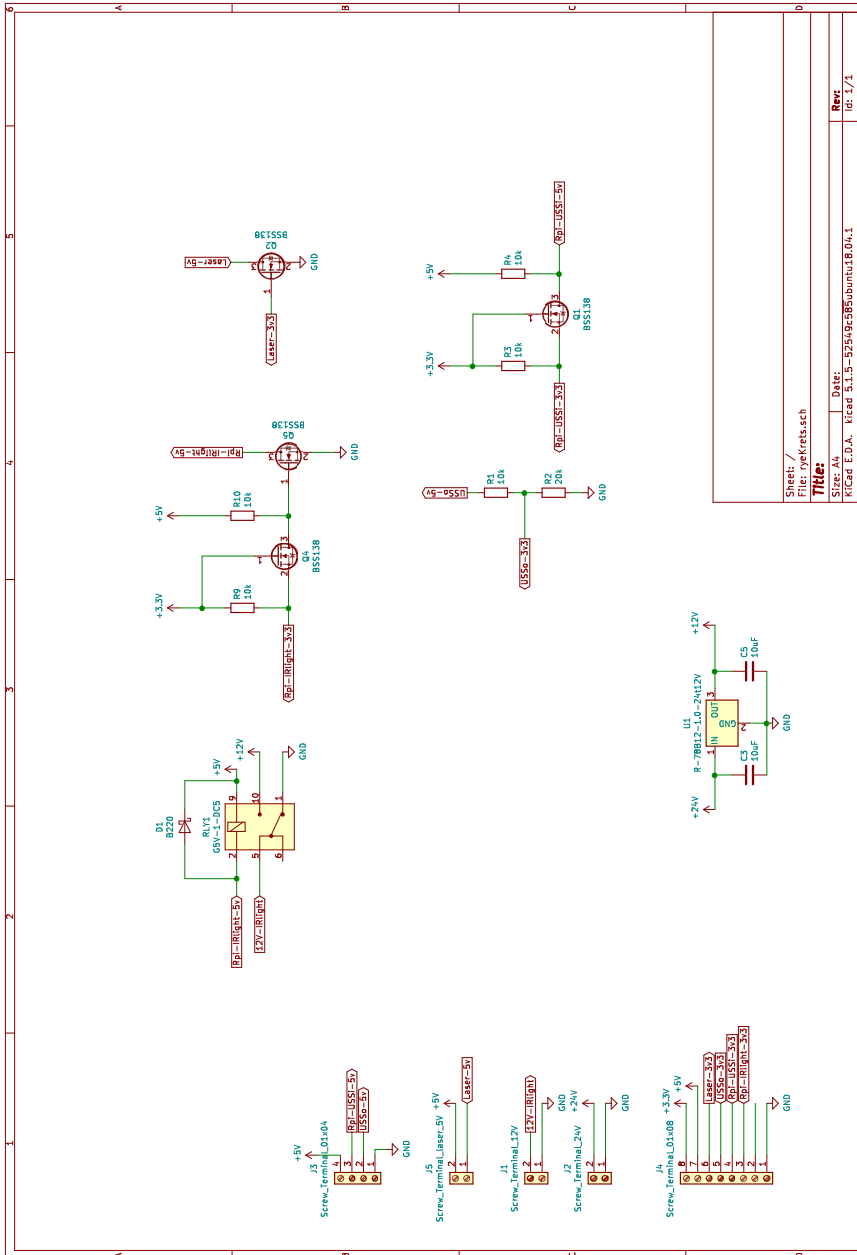
- [38] Hackster. *Benchmarking TensorFlow and TensorFlow Lite on the Raspberry Pi*. May 2019. URL: <https://www.hackster.io/news/benchmarking-tensorflow-and-tensorflow-lite-on-the-raspberry-pi-43f51b796796> (visited on 09/27/2019).
- [39] Coral. *USB Accelerator*. URL: <https://coral.withgoogle.com/products/accelerator/> (visited on 09/16/2019).
- [40] FLIR Systems. *Lens calculator*. URL: <https://www.flir.com/iis/machine-vision/lens-calculator/> (visited on 11/03/2019).
- [41] Flir. *Tamron 8mm 1/1.8inch C mount Lens | FLIR Systems*. URL: <https://www.flir.com/products/tamron-8mm-11.8inch-c-mount-lens/> (visited on 05/29/2020).
- [42] Ximea. *Ximea Camtool*. URL: https://www.ximea.com/support/wiki/allprod/ximea_camtool (visited on 10/06/2019).
- [43] Connor Shorten and Taghi Khoshgoftaar. “A survey on Image Data Augmentation for Deep Learning.” eng. In: *Journal of Big Data* 6.1 (2019), pp. 1–48. (Visited on 10/13/2019).
- [44] Karen Simonyan and Andrew Zisserman. “Very Deep Convolutional Networks for Large-Scale Image Recognition.” eng. In: *arXiv.org* (2015). URL: <http://search.proquest.com/docview/2081521649/> (visited on 10/17/2019).
- [45] Medium. *CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more*. November 2017. URL: <https://medium.com/analytics-vidhya/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5> (visited on 10/17/2019).
- [46] “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” In: *Journal of Machine Learning Research* 15 (2014), pp. 1929–1958. URL: <http://jmlr.org/papers/v15/srivastava14a.html> (visited on 11/14/2019).
- [47] John Canny. “A computational approach to edge detection.” eng. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* pami-8 (1986), pp. 679–698. URL: <http://search.proquest.com/docview/23996017/> (visited on 10/17/2019).
- [48] Read the Docs. *CV2 Hysteresis*. URL: https://opencv-python-tutroals.readthedocs.io/en/latest/_images/hysteresis.jpg (visited on 10/31/2019).

- [49] Jan Hosang, Rodrigo Benenson, and Bernt Schiele. “Learning non-maximum suppression.” In: (2017). (Visited on 10/07/2019).
- [50] Github. *Greedy Non-Maximum suppression, Felzenszwalb method*, Github. URL: <https://github.com/rbgirshick/voc-dpm/blob/master/test/nms.m> (visited on 05/29/2020).
- [51] OpenCV. *OpenCV: Contours Hierarchy*. URL: https://docs.opencv.org/3.4/d9/d8b/tutorial_py_contours_hierarchy.html (visited on 03/12/2020).
- [52] Tomasz Malisiewicz. *blazing fast nms.m (from exemplar-svm library)*. August 2011. URL: <http://www.computervisionblog.com/2011/08/blazing-fast-nmsm-from-exemplar-svm.html> (visited on 11/03/2019).
- [53] Michael Kerrisk. *top command - Linux manual page*. URL: <http://man7.org/linux/man-pages/man1/top.1.html> (visited on 04/24/2020).
- [54] Tensorflow. *Converter Python API guide*. URL: https://www.tensorflow.org/lite/convert/python_api (visited on 05/06/2020).
- [55] DigiKey. *Digikey - IR distance sensor*. URL: <https://www.digikey.com/product-detail/en/parallax-inc/28995/28995-ND/3523692> (visited on 04/24/2020).
- [56] Sharp. *Digikey - IR distance sensor datasheet*. URL: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a21yk_e.pdf (visited on 04/24/2020).
- [57] Adafruit. *HC-SR04 Ultrasonic Sonar Distance Sensor user manual*. URL: https://media.digikey.com/pdf/Data%20Sheets/Adafruit%20PDFs/3942_Web.pdf (visited on 04/26/2020).
- [58] Watterott. *Laser Module Emitter - Red Line (1mW)*. URL: <https://shop.watterott.com/Laser-Module-Emitter-Red-Line-1mW> (visited on 04/28/2020).
- [59] Zhou Wang et al. “Image quality assessment: from error visibility to structural similarity.” eng. In: *IEEE Transactions on Image Processing* 13.4 (2004), pp. 600–612.
- [60] *R-78B-1.0(L) Recom DC/DC Converter*. URL: <https://recom-power.com/pdf/Innoline/R-78B-1.0.pdf> (visited on 05/03/2020).

-
- [61] Raspberry Pi foundation. *Raspberry Pi 4 Model B schematics*. URL: https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_4b_4p0_reduced.pdf (visited on 05/16/2020).
- [62] Raspberry Pi foundation. *Raspberry Pi 4 Model B power-requirements*. URL: <https://www.raspberrypi.org/documentation/faqs/#pi-power> (visited on 05/16/2020).
- [63] XIMEA. *XIMEA - USB3 Vision Standard cameras with USB 3.0 interface based on Onsemi, CMOSIS and e2V*. URL: <https://www.ximea.com/en/products/usb3-vision-cameras-xiq-line/mq013mg-e2> (visited on 05/20/2020).
- [64] HW group. *How (not) to destroy a relay | HW-group.com*. URL: <https://www.hw-group.com/cs/podpora/how-not-to-destroy-a-relay> (visited on 05/29/2020).
- [65] Raspberry Pi foundation. *Thermal testing Raspberry Pi 4*. URL: <https://www.raspberrypi.org/blog/thermal-testing-raspberry-pi-4/> (visited on 05/23/2020).
- [66] Sparkfun. *Sparkfun Ultrasonic Distance Sensor*. URL: <https://www.sparkfun.com/products/15569> (visited on 05/13/2020).
- [67] *Ultra High Resolution 10m Depth/Pressure Sensor*. URL: <https://bluerobotics.com/store/sensors-sonars-cameras/sensors/bar02-sensor-r1-rp/> (visited on 05/21/2020).
- [68] *0-30 BAR DIGITAL PRESSURE SENSOR (TE-connectivity)*. URL: <https://www.te.com/usa-en/product-CAT-BLPS0017.html> (visited on 05/21/2020).
- [69] pigpio library. *Pigpio library*. URL: <http://abyz.me.uk/rpi/pigpio/index.html> (visited on 05/21/2020).

Appendices

A PCB schematic



B Installation guide

This is a installation guide for the Raspberry Pi. It is assumed that the Raspberry Pi is set up with a working image of Ubuntu or a similar Linux-based operating system.

1. `sudo apt-get update && sudo apt-get upgrade`
2. `sudo apt-get install build-essential cmake pkg-config`
3. `sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng-dev`
4. `sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev`
5. `sudo apt-get install libxvidcore-dev libx264-dev`
6. `sudo apt-get install libfontconfig1-dev libcairo2-dev`
7. `sudo apt-get install libgdk-pixbuf2.0-dev libpango1.0-dev`
8. `sudo apt-get install libgtk2.0-dev libgtk-3-dev`
9. `sudo apt-get install libatlas-base-dev gfortran`
10. `sudo apt-get install libhdf5-dev libhdf5-serial-dev libhdf5-103`
11. `sudo apt-get install libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5`
12. `wget https://bootstrap.pypa.io/get-pip.py`
13. `sudo python3 get-pip.py`
14. `pip3 install opencv-contrib-python==4.1.0.25`
15. `pip3 install numpy`
16. `pip3 install tensorflow`
17. `pip3 install keras`
18. `pip3 install h5py`
19. `pip3 install matplotlib`
20. `pip3 install Pillow`
21. `wget https://www.ximea.com/downloads/recent/XIMEA_Linux_SP.tgz`
22. `tar xzf XIMEA_Linux_SP.tgz`
23. `cd package`
24. `./install`
25. `sudo tee /sys/module/usbcore/parameters/usbfs_memory_mb >/dev/null`
`«<0`

C Bill of materials

Reference	Value	Datasheet	
J1	Screw_Terminal_12V	~	Connector
J2	Screw_Terminal_24V	~	Connector
U1	R-78B12-1.0-24t12V	https://www.recom-power.com/pdf/Innoline/R-78Bxx-1.0.pdf	Regulator_Switching
C3	10uF	~	Device
C5	10uF	~	Device
RLY1	G5V-1-DC5	https://omronfs.omron.com/en_US/ecb/products/pdf/en-g5v_1.pdf	dk_Signal-Relays-Up-to-1-Amps
Q4	BSS138	https://www.fairchildsemi.com/datasheets/BS/BSS138.pdf	Transistor_FET
R9	10k	~	Device
R10	10k	~	Device
Q5	BSS138	https://www.fairchildsemi.com/datasheets/BS/BSS138.pdf	Transistor_FET
J4	Screw_Terminal_01x08	~	Connector
D1	B220	http://www.jameco.com/Jameco/Products/ProdDS/1538777.pdf	Diode
J5	Screw_Terminal_laser_5V	~	Connector
Q2	BSS138	https://www.fairchildsemi.com/datasheets/BS/BSS138.pdf	Transistor_FET
J3	Screw_Terminal_01x04	~	Connector
R1	10k	~	Device
R2	20k	~	Device
Q1	BSS138	https://www.fairchildsemi.com/datasheets/BS/BSS138.pdf	Transistor_FET
R3	10k	~	Device
R4	10k	~	Device