Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Karoline Hokstad Barstein

Towards automated welfare monitoring of farmed salmon exploiting deep learning and computer vision

Master's thesis in Marine Cybernetics
Supervisor: Martin Ludvigsen
Co-supervisor: Christian Schellewald, Rune Volden
July 2021

NTNU
Norwegian University of
Science and Technology

Karoline Hokstad Barstein

Towards automated welfare monitoring of farmed salmon exploiting deep learning and computer vision

Master's thesis in Marine Cybernetics
Supervisor: Martin Ludvigsen
Co-supervisor: Christian Schellewald, Rune Volden
July 2021

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Norwegian University of
Science and Technology

Project Description

Working Title: Deep-Learning and Vision-Based Techniques for Monitoring Farmed Salmon Welfare

Work Description:
Monitoring farmed fish welfare is crucial from ethical and financial perspectives. Going towards a more autonomous fish farming industry, many aspects can be explored using computer vision-based techniques on camera data from farmed fish sea cages to monitor fish behavior. Deep-learning-based computer vision enables adaptable, scalable, and data-driven methods suitable for overcoming challenges of the dynamic underwater environment in sea cages. Examples of indicators of fish welfare that can be captured by vision are swimming velocity, lice, wounds/injuries, gill cover frequency, and fish orientation. This master's thesis will investigate vision-based techniques on stereo camera data from farmed salmon sea cages to monitor the welfare of farmed salmon.

The thesis is written in collaboration with FiiZK, a supplier of software, closed cage, and technical tarpaulins in the aquaculture industry, and SINTEF, a research organization that conducts contract research and development projects. FiiZK provides the necessary equipment for running machine learning models, and SINTEF provides stereo camera data from salmon sea cages. Investigation of data availability, evaluation of the limitation of the data, and constructing the scope is a part of the work in the thesis.

Workflow:

1. Data acquisition and decision of main focus of the thesis: Investigate available camera data from salmon sea cages and record new data if needed. The data availability and limitations will be evaluated to formulate the scope of the thesis.
2. Perform a background and literature review on:
    - Fish welfare indicators based on fish behavior.
    - Previous studies on monitoring fish behavior.
    - Relevant traditional computer vision methods.

- Relevant (deep-learning) vision-based techniques.

3. Choose fish welfare indicators to look closer at, by:
    - Classification of indicators by visual and temporal detectability.
    - Investigation of the quality and size of the dataset.
4. Implement computer vision and deep learning methods for fish welfare monitoring:
    - Prepare datasets.
    - Investigate how to combine vision-based techniques to gain insight about farmed salmon welfare.
    - Perform tasks using chosen vision-based techniques on the prepared dataset.
5. Evaluate the results of the individual methods and the final experimental pipeline.
6. Conclude the thesis and make a recommendation for applications, improvements, and further work.

Summary

Preserving fish welfare is an extensive issue in the fish farming industry. An industrial salmon sea cage can contain as many as 200 000 fish, and monitoring the welfare of this amount of individual animals is challenging. Manual observations and measurements of behavior, environmental factors, and fish health have been predominant. However, farmed salmon production is still expanding, and there is a need for more thorough and efficient methods. One problem is the inefficiency of manual operations, another is the lack of a complete spatial and temporal overview of the sea cage. Most sea cages today have underwater cameras installed used for manual observations, and several indicators of fish welfare can be monitored visually. This can be exploited by computer vision technology. Underwater images are subject to complex challenges, but the evolution in computer-vision and deep-learning technology shows promise for applying such methods for practical purposes to better and more efficiently monitor fish welfare. This thesis proposes a proof-of-concept for estimating farmed salmon swimming velocity on a dataset of stereo video from industrial sea cages by combining deep-learning object detection, IOU-based object tracking, and image scene depth estimation by semi-global block matching.

A number of salmon welfare indicators were evaluated and classified based on their visual and temporal detectability. *Swimming activity*, or more specifically *swimming velocity*, was considered as the most feasible indicator to study, given the dataset. Scattering, low contrast, color distortions, and light reflections represent additional challenges for vision-based techniques underwater, compared to terrestrial applications. Swimming velocity was to be estimated by tracking each fish by measuring the position of a fixed point on the fish body. The middle point of object detection bounding boxes served as this fixed point. Fish anatomy was studied to decide which object to track, together with considerations of the limitations of the dataset itself, depth estimation, and object detection. The caudal fin appeared to be the best choice.

For object detection, the YOLOv4 model was chosen due to its GPU parallel computing capabilities and its superior performance in terms of speed and accuracy. Image frames were extracted from the videos and annotated. Each training image was augmented four times, resulting in a training set with five versions of each image. The best-performing

network weights through the training were saved and used for the final model. To obtain 3D positions for the salmon, stereo matching by semi-global block matching was implemented. From the resulting disparity maps, the 3D position of caudal fins could be estimated by using the average disparity value of a predefined inner middle area of the bounding box and the center point of the bounding box. Detections with an average disparity value in this inner area above a certain threshold, indicating a non-smooth depth estimation of the fin surface, were discarded. The proposed object tracking method was based on overlap between detection boxes, meaning no image or motion information was required. The algorithm was modified to allow "gaps" between detections in consecutive frames. The individual components were combined to form a final experimental pipeline to estimate salmon swimming velocity. The 3D caudal fin positions were tracked in two video experiments. For each 3D track, the mean velocity was calculated using the distance between consecutive points, the number of frames between each certain point, and the frame rate of the videos. Further, the mean salmon swimming velocity and standard deviation were calculated for each experiment.

The detection model was tested on speed and performance. The model performed at 87.49% mAP@0.50 after training on 208 original images. For the stereo 3D reconstruction part, the total reprojection error of the camera calibration was 0.2234 pixels. Further, the disparity maps showed improvement after applying a weighted least squares filter. Speckles were removed, and surfaces appeared smoother. Measurements of known sizes in reconstructed calibration images corresponded with the ground truth sizes. The tracking algorithm was tested on the YOLOv4 detections. By using a moderate IOU overlap threshold, we achieved reliable tracks. The maximum and average track length increased when allowing gaps and the number of total tracks was reduced. The final tracking algorithm was subject to some issues caused by the detection results, e.g., tracks constructed of false positives and split tracks caused by false negatives.

Visualizations and 3D plots of tracks from the final pipeline experiments, together with measurements of mean velocity and standard deviation over all tracks in each video, were analyzed. The mean salmon swimming velocities were measured to 0.5479 ms^{-1} and 0.6561 ms^{-1}, and the standard deviations to 0.3175 ms^{-1} and 0.3149 ms^{-1}, respectively.

Motion blur was the main contributor to errors in the final pipeline of swimming velocity estimation. The object detection model performed sufficiently on detecting caudal fins in real-time. The modification of the tracking algorithm to allow gaps induced an improvement in the length of tracks and decreased the number of tracks, indicating that tracks split due to gaps in the original algorithm were merged in the modified algorithm. For the tracking algorithm, camera egomotion affecting IOU overlap appeared as the largest issue. A next step might be to include a motion model for the egomotion or use an algorithm that considers image information.

Overall, by removing insufficient detections and disparity estimations from the tracks, we obtained an experimental pipeline that was able to estimate salmon swimming velocities in the two experiments conducted. 3D plots of tracks corresponded with the observed motion of the fish relative to the camera. The mean velocities computed from the tracks were reasonable for the swimming velocity of farmed salmon. As a proof-of-concept, the system shows that there is large potential in applying and combining well-known vision-based techniques for monitoring fish welfare in industrial sea cages and that image quality is the main limitation as of today.

Sammendrag

Bevaring av fiskevelferd er et omfattende tema i oppdrettsnæringen. En industriell laksemerd kan inneholde så mange som 200 000 fisk, og det er utfordrende å overvåke velferden til denne mengden enkeltdyr. Manuelle observasjoner og målinger av atferd, miljøfaktorer og fiskehelse har vært dominerende. Oppdrettslaksproduksjonen utvides imidlertid fortsatt, og det er behov for grundigere og mer effektive metoder. Et problem er ineffektiviteten ved manuell operasjon, et annet er mangelen på en fullstendig romlig og tidsmessig oversikt over merden. De fleste laksemerder i dag har undervannskameraer installert brukt til manuelle observasjoner, og flere indikatorer for fiskevelferd kan overvåkes visuelt. Dette kan utnyttes av datasynteknologi. Undervannsbilder er underlagt komplekse utfordringer, men utviklingen innen datasyn og dyplæringsteknologi viser muligheter til å bruke slike metoder for praktiske formål, for bedre og mer effektiv overvåkning av fiskevelferd. Denne oppgaven foreslår et proof-of-concept for estimering av svømmehastighet for oppdrettslaks på et datasett med stereovideo fra industrielle laksemerder. Dette ved å kombinere objektdeteksjon ved dyp læring, IOU-basert objektsporing og dybdeestimering av bildescener ved semi-global blokkmatching.

En rekke laksevelferdsindikatorer ble evaluert og klassifisert ut fra deres visuelle og tidsmessige detekterbarhet. *Svømmeaktivitet*, eller nærmere bestemt *svømmehastighet*, ble ansett som den mest gjennomførbare indikatoren å studere, gitt datasettet. Spredning, lav kontrast, fargeforvrengninger og lysrefleksjoner representerer ytterligere utfordringer for synsbaserte teknikker under vann, sammenlignet med terrestriske applikasjoner. Svømmehastigheten ble beregnet ved å spore hver fisk ved å måle posisjonen til et fast punkt på fiskekroppen. Midtpunktet for deteksjonsboksene fungerte som dette faste punktet. Fiskeanatomi ble studert for å bestemme hvilket objekt som skulle spores, sammen med hensyn til begrensningene i selve datasettet, dybdeestimering og gjenstandsdeteksjon. Kaudalfinnen så ut til å være det beste valget.

For objektgjenkjenning ble modellen YOLOv4 valgt på grunn av parallelle databehandlingsmuligheter på GPU og overlegen ytelse når det gjelder hastighet og nøyaktighet. Bilder ble hentet fra videoene og annotert. Hvert treningsbilde ble augmentert fire ganger, noe som resulterte i et treningssett med fem versjoner av hvert bilde. De beste

nettverksvektene fra treningen ble lagret og brukt til den endelige modellen. For å oppnå 3D-posisjoner for laksen ble stereomatching ved semi-global blokkmatching implementert. Fra de resulterende dybdeestimatene kunne 3D-posisjonen til kaudalfinner estimeres ved å bruke det gjennomsnittlige dybdeestimatet til et forhåndsdefinert indre midtområde av deteksjonsboksen og midtpunktet for deteksjonsboksen. Deteksjoner med et gjennomsnittlig dybdeestimat i dette indre området over en viss terskel, som indikerer en ikke-jevn dybdeestimering av finneflaten, ble forkastet. Den foreslåtte objektsporingsmetoden var basert på overlapp mellom deteksjonsbokser, noe som betyr at ingen bilde- eller bevegelsesinformasjon var nødvendig. Algoritmen ble modifisert til å tillate "hull" mellom deteksjoner i påfølgende bilder. De enkelte komponentene ble kombinert for å danne en endelig eksperimentell pipeline for å estimere laksens svømmehastighet. Finneposisjoner i 3D ble sporet i to videoeksperimenter. For hvert 3D-spor ble gjennomsnittshastigheten beregnet ved hjelp av avstanden mellom påfølgende punkter, antall bilder mellom hvert bestemte punkt og bildefrekvensen til videoene. Videre ble den gjennomsnittlige lakseshastigheten og standardavviket beregnet for hvert eksperiment.

Deteksjonsmodellen ble testet på hastighet og ytelse. Modellen hadde en ytelse på 87.49% mAP@0.50 etter trening på 208 originale bilder. For 3D-rekonstruksjonsdelen var den totale reprojeksjonsfeilen for kamerakalibreringen 0,2234 piksler. Videre viste dybdeestimeringene forbedring etter påføring av et vektet minste kvadrat-filter. Hull i dybdeestimeringene ble fjernet, og overflatene ble jevnere. Målinger av kjente lengder i rekonstruerte kalibreringsbilder samsvarte med de sanne størrelsene. Sporingsalgoritmen ble testet på YOLOv4-deteksjonene. Ved å bruke en moderat grense for IOU-overlapp oppnådde vi pålitelige spor. Maksimal og gjennomsnittlig sporlengde økte når det ble tillatt hull, og antall spor ble redusert. Den endelige sporingsalgoritmen var gjenstand for noen problemer forårsaket av deteksjonsresultatene, for eksempel spor konstruert av falske positive deteksjoner og splittede spor forårsaket av falske negative deteksjoner.

Visualiseringer og 3D-plott av spor fra eksperimentene med den endelige pipelinen, sammen med målinger av gjennomsnittshastighet og standardavvik over alle spor i hver video, ble analysert. Gjennomsnittlig laksesvømmehastighet ble målt til henholdsvis 0,5479 ms^{-1} og 0,6561 ms^{-1}, og standardavvikene til 0,3175 ms^{-1} og 0,3149 ms^{-1}.

Uskarphet var den viktigste bidragsyteren til feil i den endelige pipelinen for estimering av svømmehastighet. Objektdeteksjonsmodellen detekterte kaudalfinner i sanntid. Modifiseringen av sporingsalgoritmen induserte en forbedring i sporlengden og reduserte antall spor, noe som indikerer at sporene som ble splittet på grunn av hull i den opprinnelige algoritmen, ble slått sammen i den modifiserte algoritmen. For sporingsalgoritmen var kameraets egenbevegelse, som påvirker IOU-overlapp, som det største problemet. Et neste steg kan være å inkludere en bevegelsesmodell for egenbevegelse eller bruke en algoritme som tar hensyn til bildeinformasjon.

Samlet sett oppnådde vi en eksperimentell pipeline som var i stand til å estimere laksens svømmehastighet i de to eksperimentene, ved å fjerne utilstrekkelige deteksjoner og dybdeestimeringer fra sporene. 3D-plott av spor samsvarte med den observerte bevegelsen til fisken i forhold til kameraet. Gjennomsnittlige hastigheter beregnet fra sporene var rimelige for svømmehastigheten til oppdrettslaks. Som et proof-of-concept viser systemet at det er stort potensial i å anvende og kombinere velkjente visjonsbaserte teknikker for å overvåke fiskevelferd i industrielle laksemerder, og at bildekvalitet er den viktigste begrensningen per i dag.

Preface

This master's thesis is carried out during the spring/summer of 2021. It is submitted as the final thesis for the Master of Science (M.Sc.) degree at the Norwegian University of Science and Technology (NTNU). This report accounts for 100% of the final grade in *TMR4930 - Marine Technology, Master's Thesis*.

The work has been performed at the Department of Marine Technology (IMT) at NTNU in collaboration with FiiZK and SINTEF Ocean, under the supervision of Professor Martin Ludvigsen (NTNU) and co-supervisors Rune Volden (FiiZK) and Christian Schellewald (SINTEF Ocean). The outline for the project was formed in collaboration with them. The presented work contributes to the knowledge building for the characterization and stereo camera-based metric measurements of salmon within ongoing projects at SINTEF Ocean (i.e. INDISAL (NFR 282423), OWITOOLS (FHF 901594)). SINTEF Ocean provided the data originating from the FHF project LAKSIT (FHF 901184). FiiZK provided the necessary equipment.

Through the two module courses the autumn semester of 2020, *TMR06 - Autonomous Marine Systems* and *TTK25 - Computer Vision for Control*, I have gained more knowledge about computer vision and deep learning and especially the applications of these in marine systems and underwater technology. The topics in TMR06 have given me an essential insight into the opportunities that exist in these fields and the challenges that must be conquered. In TTK25 I gained invaluable insight into the current evolution in the application of computer vision and deep learning methods in aquaculture. This has been a great inspiration for the design of this project. The work presented is solely done by me, unless otherwise stated.

Trondheim, July 9, 2021

Karoline Hokstad Barstein

Acknowledgements

I would like to express my deepest thanks to my supervisor Professor Martin Ludvigsen (NTNU), for his guidance and motivation throughout the project. His concise and honest feedback has been essential to conduct the writing of this thesis. Further, my co-supervisor Rune Volden (FiiZK) deserves my gratitude for providing me with inspiration and motivation during the process. He has also been the key person to retrieve access to the equipment needed, which was decisive to carry out the project. Finally, I would like to express my appreciation to my co-supervisor Christian Schellewald (SINTEF Ocean), a key person for acquiring image data, sharing computer vision knowledge, and providing essential guidance and motivation to finalize the thesis.

K.H.B.

Abbreviations

AP	Average Precision
AUC	Area Under Curve
AUV	Autonomous Underwater Vehicle
CNN	Convolutional neural network
COCO	Common Objects in Context
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
CV	Computer Vision
DL	Deep Learning
FC	Fully Connected
FN	False Negative
FP	False Positive
FPS	Frames Per Second
GAN	Generative Adversarial Network
GD	Gradient Descent
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IOU	Intersection Over Union
MOT	Multiple Object Tracking
MSE	Mean Square Error
NMS	Non-Max Suppression
NN	Neural network
PR	Precision Recall
R-CNN	Region-based CNN
RAM	Random Access Memory
ReLU	Rectified Linear Unit
RGB	Red Green Blue
ROI	Region Of Interest
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SSD	Single-Shot Multibox Detector
SVM	Support Vector Machine
TN	True Negative
TP	True Positive
VoTT	Visual Object Tagging Tool
WI	Welfare Indicator
WLS	Weighted Least Squares

Table of Contents

1 Introduction **1**

1.1 Background 1

1.1.1 Motivation 1

1.1.2 Underwater Imaging 2

1.1.3 Fish Welfare Indicators 4

1.1.4 Research Question 6

1.1.5 Related Work 7

1.2 Contribution 8

1.3 Thesis Outline 8

2 Deep Neural Networks **11**

2.1 Artificial Neural Networks 11

2.1.1 Basic Structure 11

2.1.2 Training 13

2.1.2.1 Backpropagation 13

2.1.2.2 Gradient Descent 15

2.1.2.3 Overfitting 16

2.1.2.4 Data Augmentation 17

2.2 Convolutional Neural Networks 18

2.2.1 Convolutional Layer 19

2.2.2 Rectified Linear Unit (ReLU) . . . 21
2.2.3 Pooling Layer . . . 21
2.2.4 Fully Connected Layer . . . 22
2.2.5 Overall View . . . 23
2.2.6 Transfer Learning . . . 23

3 Object Detection and Tracking **25**
3.1 Object Detection . . . 25
3.1.1 Basic Structure . . . 25
3.1.2 State-of-the-Art Models . . . 27
3.2 YOLOv4 . . . 30
3.3 Detection-Based Multiple Object Tracking . . . 35
3.3.1 Kalman Filter . . . 36
3.3.2 Optical Flow . . . 36
3.3.3 IOU Tracking . . . 37

4 Stereo Vision for 3D Reconstruction **39**
4.1 Single-View Geometry . . . 39
4.1.1 Pinhole Camera Model . . . 39
4.1.1.1 Principal Point Offset . . . 40
4.1.1.2 Camera Rotation and Translation . . . 41
4.1.2 Distortion Model . . . 42
4.2 Stereo-View Geometry . . . 43
4.2.1 Epipolar Geometry . . . 43
4.2.1.1 Epipolar Line . . . 44
4.2.1.2 The Fundamental Matrix . . . 44
4.2.2 Image Rectification . . . 45

4.3 Stereo Matching . . . 46
4.3.1 The Correspondence Problem . . . 46
4.3.1.1 Block Matching . . . 47
4.3.1.2 Semi-Global Block Matching . . . 50
4.3.1.3 Feature Matching . . . 51
4.3.2 Disparity Post-Processing . . . 51
4.4 The Reconstruction Problem . . . 52

5 Implementation **55**
5.1 Data Acquisition . . . 55
5.1.1 Data Collection . . . 55
5.1.2 Preparations for Object Detection . . . 56
5.1.2.1 Selection of Object of Interest . . . 57
5.1.2.2 Annotation . . . 59
5.2 Implementation Prerequisites . . . 62
5.2.1 Computer . . . 62
5.2.2 Software . . . 63
5.3 Object Detection Model . . . 64
5.3.1 Selection of Detection Model . . . 64
5.3.2 Training . . . 66
5.3.2.1 Data Augmentation . . . 66
5.3.2.2 Transfer Learning . . . 67
5.3.2.3 Network Configuration . . . 67
5.3.2.4 Validation . . . 69
5.3.3 Evaluation of Detection Model . . . 70
5.3.4 Object Detection Pipeline . . . 73
5.4 Stereo 3D Reconstruction . . . 73

5.4.1 Camera Calibration . . . 74
5.4.2 Undistortion and Rectification . . . 75
5.4.3 Stereo Matching . . . 75
5.4.3.1 Selection of Stereo Matching Algorithm . . . 76
5.4.3.2 Tuning of Stereo Matching Parameters . . . 76
5.4.4 Disparity Post-Processing and Triangulation Pre-Processing . . . 76
5.4.5 Triangulation and Pointclouds . . . 77
5.4.6 Depth Estimation Pipeline . . . 77
5.5 Tracking Algorithm . . . 78
5.5.1 Modified IOU Tracking Algorithm . . . 78
5.5.2 Evaluation of Tracking Algorithm . . . 80
5.6 Swimming Velocity Estimation . . . 80
5.6.1 3D Position of Individual Caudal Fins . . . 81
5.6.2 3D Velocity . . . 82
5.6.3 Final Pipeline . . . 83

6 Results **85**

6.1 Detection Model . . . 85
6.2 Depth Estimation . . . 89
6.2.1 Camera Calibration . . . 89
6.2.2 Disparity Post-Processing . . . 90
6.2.3 3D Reconstruction . . . 92
6.3 Video Analysis of Tracking . . . 94
6.4 Swimming Velocity . . . 98
6.4.1 Detections and Disparities . . . 98
6.4.2 3D Tracks . . . 101
6.4.3 Mean Velocity and Standard Deviation . . . 106

7 Discussion **107**

7.1 Dataset and Annotation . . . 107

7.1.1 Dataset . . . 107

7.1.2 Annotation . . . 108

7.2 Performance of Detection Model . . . 108

7.3 Stereo Matching and 3D Reconstruction . . . 109

7.3.1 Camera Calibration . . . 109

7.3.2 Stereo Matching Algorithm . . . 109

7.3.3 Disparity Post-Processing . . . 110

7.3.4 3D Reconstruction . . . 110

7.4 Tracking Algorithm . . . 111

7.4.1 Comparison of Algorithms . . . 111

7.4.2 Implementation and Tuning . . . 112

7.4.3 Influencing Factors . . . 112

7.5 Swimming Velocity Estimation . . . 113

7.5.1 Detections and Disparities . . . 113

7.5.2 3D Tracks . . . 113

7.5.3 Mean Velocity and Standard Deviation . . . 114

7.5.4 Pipeline Implementation . . . 114

8 Conclusions and Further Work **117**

8.1 Conclusions . . . 117

8.2 Further Work . . . 119

Bibliography **121**

Appendix **129**

A Source Code . . . 129

List of Figures

1.1 The underwater imaging process illustrating the losses of light to an image in an underwater imaging system. Image courtesy by (Funk et al. 1972). . 3

2.1 Graph model of a neuron. 12

2.2 Illustration of a single-layer, fully connected neural network. 12

2.3 Illustration of forward- and backpropagation during training in a neural network. 14

2.4 Early stopping point and overfitting. 17

2.5 Examples of data augmentation techniques. Image courtesy by Chen et al. (2020). 18

2.6 Fukushima's concept of convolutional networks from 1980. Basic features are extracted by receptive fields at the lower layers (left) and combined to more complex features at the higher levels (right). Image courtesy of Fukushima (1980). 19

2.7 A visual explanation of convolution. Based on Goodfellow et al. (2016). . . 20

2.8 Edge detection using two different kernels, one capturing vertical edges and one capturing horizontal edges. By combining the results we obtain the majority of edges in the image. Figure retrieved from *CNN Edge detection* (2018) and modified. 21

2.9 Max-pooling operation. 22

2.10 Layers of a typical CNN. 23

3.1 High-level architecture of an object detector. 26

3.2 Example result from object detection. The outputs are bounding boxes and class predictions for each object of interest in the image. Image courtesy by Redmon et al. (2016). 26

3.3 YOLO bounding box and class prediction. Image courtesy of Redmon et al. (2016). 29

3.4 SSD training process. Image courtesy of Liu et al. (2015). 30

3.5 Anchor box and predicted offsets. Width and height of the bounding box are predicted as offsets from cluster centroids. The center coordinates of the box relative to the top left corner of the cell are found by using a sigmoid function, which forces the output to be between 0 and 1. Image courtesy of Redmon & Farhadi (2016). 31

3.6 An example of the application of the CSPNet strategy (here, on DenseNet (Huang et al. 2017)). Image courtesy of Wang et al. (2020). 34

3.7 (a) Information propagation in FPN, red line. (b) Information propagation in PANet, green line. Image courtesy of Liu et al. (2018). 34

3.8 Original PAN vs. YOLOv4 modified PANet. Image courtesy of Bochkovskiy et al. (2020). 35

3.9 The aperture problem. The line appears to be moving to the right when viewing through the aperture, but is in reality also moving down. It is not possible to determine the correct direction of movement unless the ends of the line are visible. 37

3.10 Principle of the IOU tracking algorithm. Based on Bochinski et al. (2017). 38

4.1 The Euclidian transformation between the world and camera coordinate frames. Image courtesy by Hartley & Zisserman (2003). 42

4.2 Checkerboard pattern, appearing with no distortion, positive radial distortion, and negative radial distortion, respectively. Image courtesy by Ozcakir (2020). 42

4.3 Illustration of block matching process. Image courtesy by McCormick (2014). 48

4.4 SAD cost computation between two possibly matching blocks. Image courtesy by McCormick (2014). 48

4.5 SGBM block diagram, using five directions. Image courtesy by The MathWorks, Inc. (n.d.). 51

5.1 Salmon external anatomy. 58

5.2 Screenshot from annotation tool (VoTT). 60

5.3 Example of image frame with extensive motion blur. 60

5.4 Example of image frame of scarce quality. The image contains a high level of noise, several tale fin occlusions, and an unclear distinction between foreground and background. 61

5.5 Example of an accepted image frame. The image has clear distinctions between foreground and background, an acceptable amount of motion blur, and few tale fin occlusions. 62

5.6 Comparison of YOLOv4 and other state-of-the-art object detectors. Image courtesy of Bochkovskiy et al. (2020). 65

5.7 Four different augmentations of one training image. 67

5.8 Definition of Intersection over Union (IoU). 71

5.9 Calculation of average precision by using area under precision-recall curve. Image courtesy of Padilla (2019). 72

5.10 Detection pipeline. 73

5.11 Chessboard detections in two calibration stereo pairs, (a) and (b), and (c) and (d), respectively. 74

5.12 Example of rectified image pair. 75

5.13 Disparity estimation and 3D reconstruction pipeline. 77

5.14 Definition of the inner area of the bounding box, where the mean and variance of the disparity is calculated. 82

5.15 Final overall pipeline, including caudal fin detection, image scene depth estimation, and tracking. 83

6.1 YOLOv4 detections on images from test set. 87

6.2 Problematic YOLOv4 detections on images from test set. 88

6.3 YOLOv4 detections on images from test set, showing detections across scales. 88

6.4 Initial reprojection errors. Unit is pixels. 89

6.5 Improved reprojection errors. Unit is pixels. 89

6.6 Example of input images and resulting raw and filtered disparity maps. The weighted least squares (WLS) filter is used. 90

6.7 Example of input images and resulting raw and filtered disparity maps. The weighted least squares (WLS) filter is used. 91

6.8 Resulting pointcloud of a filtered disparity map, from two different views angles. Notice the fish surface marked in red and how the smoothed edges of the surface creates speckles/noise. 92

6.9 Measurement of checkerboard square size in four different pointclouds. The number behind "Distance" shows the measured distance of the red line. All units are millimeters. 93

6.10 Examples of 3D reconstruction from stereo image pairs. (a) and (b) are original (left) images. (c) and (d) are the pointclouds after 3D reconstruction for images (a) and (b), respectively. 94

6.11 Results from object tracking; notice the bounding boxes of frame $k+1$ and $k+2$. They are false positives, but get included in the track. 96

6.12 Results from object tracking; too small IOU overlap between consecutive frames. 97

6.13 Examples of good detections and disparity estimations. 98

6.14 Examples of disparity estimates discarded because of too small variance. . 100

6.15 Examples of disparity estimates discarded because of too high variance. . . 101

6.16 The longest track (128 frames, 5.33 s) of video 1 plotted in 3D coordinates. 102

6.17 The second longest track (108 frames, 4.50 s) of video 1 plotted in 3D coordinates. 103

6.18 The longest track (108 frames, 4.50 s) of video 2 plotted in 3D coordinates. 104

6.19 The second longest track (83 frames, 3.46 s) of video 2 plotted in 3D coordinates. 105

List of Tables

1.1 Evaluation and classification of WIs identified by Nofima (2018), based on their visual and temporal detectability. All WIs in the table are group based, except *gill cover frequency*, which is individual based. 5

4.1 Stereo matching approaches, as described by Brown et al. (2003). 47

4.2 Common cost functions for measuring similarity in correlation-based stereo matching methods. Based on Praveen (2019) and Brown et al. (2003). . . . 49

4.3 Nomenclature for cost function formulas. Based on Praveen (2019). 49

5.1 Camera information, taken directly from the information file provided from SINTEF with the LAKSIT dataset. Two of these cameras were used for the stereo setup. 56

5.2 Evaluation of objects of interest, using three criteria. 59

5.3 Main specifications of the computer used in the project. 63

5.4 Configuration of training parameters for YOLOv4. 68

5.5 Validation mAP after each 1000 iterations of the training process, together with the best weights result. 70

5.6 Confusion matrix. 70

5.7 Tuning parameters for OpenCV `StereoSGBM` algorithm. 76

6.1 Accuracy results on test dataset with different IOU thresholds. 86

6.2 Accuracy results on test dataset with different confidence thresholds. . . . 86

6.3 Measured checkerboard square sizes in Figure 6.9. 93

6.4 Quantitative results of tracking algorithms using detections from YOLOv4, obtained from two test videos with a length of 3120 frames. 95

6.5 Statistics for two test videos. 106

Chapter 1

Introduction

This report is a master's thesis conducted in collaboration with FiiZK Digital Integrator AS and SINTEF Ocean. Parts of the work are based on the work accomplished in my specialization project on detection and tracking of fish feed pellets.

This chapter focuses on the whole picture of the thesis, explaining the background, motivation, and some important topics. Further, research questions are defined, and the outline of the thesis is explained.

1.1 Background

1.1.1 Motivation

The history of Norwegian aquaculture started on Hitra in 1970 with a farm built by pioneers. Since then, several hundred fish farms have been built along the Norwegian coastline (Norwegian Seafood Federation 2011). The temperature and sea currents due to the Atlantic Gulf Stream provide optimum living conditions for salmon, and Norway accounts for over 50% of the global production of Atlantic salmon (Ernst & Young 2019). Nevertheless, only 0.5% of Norway's coastal zone area is used for salmon production (Norwegian Seafood Federation 2012), suggesting there are still huge opportunities for expanding production. However, several challenges need to be encountered to ensure a more efficient and sustainable industry while expanding.

An extensive problem in farmed fish production is monitoring fish welfare. The industry is closely governed; nevertheless, the mortality of fish is large during operations (Overton et al. 2019).

Today, fish welfare governing is mainly based on manual observations of the fish's behavior through video streams from submerged cameras or directly from above the water surface. This is more or less an infeasible task in terms of accuracy due to observers' fatigue and the lack of a complete overview of the situation in the sea cage. Increasing the level of autonomy will be important when going towards a more sustainable and efficient fish farming industry with a higher focus on fish welfare. Deep-learning-based computer vision techniques appear promising for observing and monitoring farmed fish. As most aquaculture sea cages today have underwater cameras installed for visual monitoring of the sea cage environment, a basic framework is already established to apply such techniques. However, underwater computer vision is subject to complex challenges in terms of both hardware and signal quality, including high device costs, complex device setups, and distortion in signals and light propagation introduced by the water as a medium (Lodi Rizzini et al. 2015). Also, due to the high costs of proper underwater cameras, most existing installed devices provide scarce data quality. As the computer vision and deep learning fields evolve, state-of-the-art models for image analysis are constantly getting more accurate and fast. Several frameworks have been developed to facilitate rapid and scalable development. But so far, few have investigated practical application and combination of such methods to monitor fish welfare by using low-cost camera setups in industrial sea cages.

1.1.2 Underwater Imaging

Application of standard computer vision techniques to underwater images requires dealing with some additional problems compared to terrestrial imagery. Underwater images can suffer from problems such as limited range visibility, low contrast, non-uniform lighting, blurring, bright artifacts, diminished color, and noise (Schettini & Corchs 2010).

In underwater imaging, an artificial light source projects light on the target, and the reflections are recorded by the camera. This happens when photons are transmitted and attenuated through the water before the reflected photons are registered by the camera (Ludvigsen et al. 2020). Underwater images are typically characterized by reduced visibility, caused by light being exponentially attenuated when traveling through the water. This limits the visibility to around twenty meters in clear water and five meters or less in turbid water (Schettini & Corchs 2010).

There are three important, general problems occurring in underwater imaging: attenuation, backscatter, and small-angle forward scattering. Six important, special cases of these are listed below, based on Funk et al. (1972), and visualized in Figure 1.1.

- Source light is outward scattered and does not reach the target.
- Source light is attenuated.

- Source light is backward scattered.
- Reflected light is attenuated.
- Reflected light is outward scattered and does not contribute to the image.
- Reflected light is small-angle forward scattered.

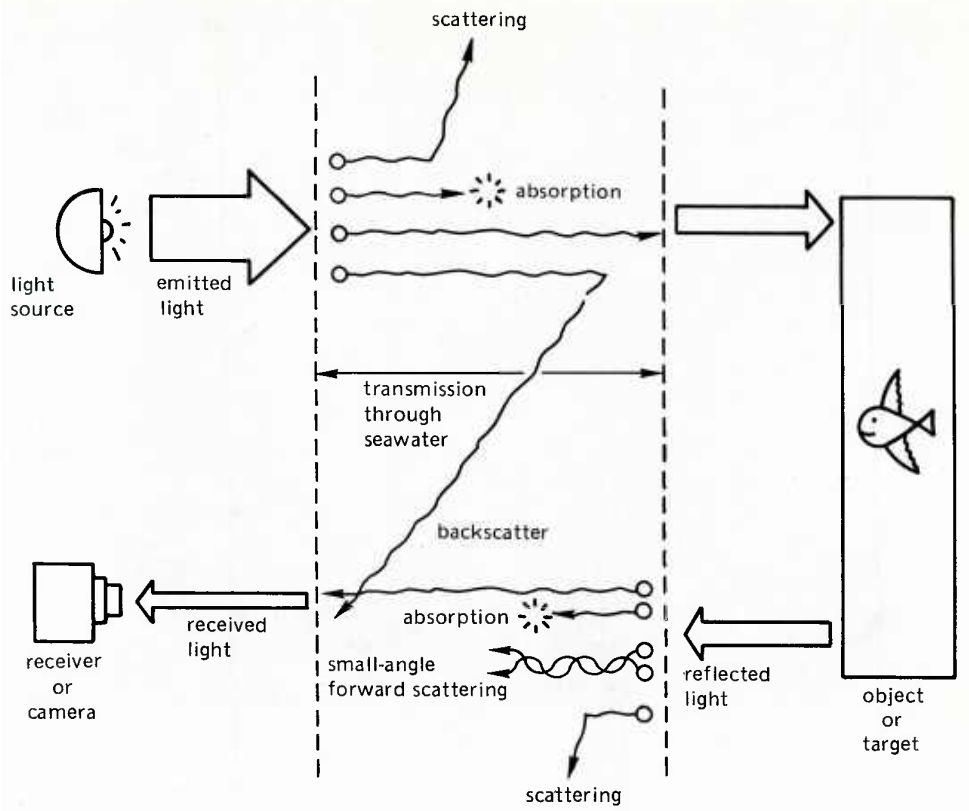


Figure 1.1: The underwater imaging process illustrating the losses of light to an image in an underwater imaging system. Image courtesy by (Funk et al. 1972).

The first problem, the spectral attenuation of visible light, is dependent on the colored dissolved matter, suspended matter, or plankton in the water. In addition, the water itself heavily attenuates the red part of the light spectrum. In the blue-green region in clear water, light can be transmitted with less attenuation than other wavelengths (Funk et al. 1972).

The second problem is backscatter. Seawater normally contains high concentrations of particles and when a photon hits a particle, its direction is changed either back towards the camera or out of the camera field of view (FoV). This leads to light-scattering, which again reduces the amounts of light that forms the imagery. Backscattering is when the scattered light reaches back to the sensor without hitting the object. This can degrade the image contrast seriously and cause blur (Funk et al. 1972).

The last problem, small-angle forward scattering, is when photons change direction without any other alternation. This can cause major losses of resolution. How serious these

losses are depends on the nature of the water, the imaging system and its geometry, and resolution requirements (Funk et al. 1972).

As one goes to deeper water, the amount of light is reduced. Depending on their wavelength, colors will be lost one by one with depth. Blue has the longest wavelength, thus, underwater images will normally be dominated by blue color (Schettini & Corchs 2010).

Motion blur can be present in underwater images. In conditions with low lightning (as in underwater imagery), the image quality is always a trade-off between motion blur and noise. When the illumination level is low, a long exposure time is required to obtain a sufficient signal-to-noise ratio. However, a long exposure time increases the risk of motion blur in the images. Motion blur is in general caused by relative motion between the camera and the subject (Kurimo et al. 2009). For underwater images, the causes for relative motion can be, e.g., dynamic water surface, moving subjects (e.g., fish), or moving camera due to currents.

1.1.3 Fish Welfare Indicators

With the recent evolution and growth in modern fish farms, fish welfare has become an emerging issue. The industry affects millions of individual fish, yet, the fish are treated legally and morally with less concern for their health and welfare than in other animal industries (Gismervik et al. 2020). Nevertheless, farmers themselves have been interested in the topic, and it has been covered in numerous research and reviews in the last years (Nofima 2018). This information can be widespread and not necessary accessible for the farmers. However, the greatest challenge occurs when the information about fish welfare is to be implemented in production. Measuring fish welfare can be difficult, and one might not even know *how* to measure it.

As a tool for measuring welfare of farmed Atlantic salmon, Nofima (2018) has developed a set of welfare indicators (WIs). They can be directly animal-based or indirectly resource-based, but some can be hard to implement on-site, e.g., if a laboratory must analyze samples.

Some WIs require special sensors or tools to be observed, some require human intervention, and some can be observed visually. There are WIs that can be observed in a snapshot or short video clips (e.g., surface activity, skin patterns, swimming speed), while others require observation over a long period of time (e.g., death rate or growth). In this thesis, visual detectability over a short period of time (a short video clip) was a constraint as it is a study of the application of computer vision techniques for monitoring fish welfare. Animal-based WIs identified in the Nofima (2018) report were reviewed and classified systematically to extract the possible welfare indicators to analyze. The result is shown

in table Table 1.1.

Table 1.1: Evaluation and classification of WIs identified by Nofima (2018), based on their visual and temporal detectability. All WIs in the table are group based, except *gill cover frequency*, which is individual based.

Indicator	**Visually detectable?**	
(animal based)	*In snapshot or short video clip*	*Over longer time*
Death rate	No	No
Swimming activity	Yes	No
Fin presence/orientation	Yes	No
Gill rate	Yes	No
Skin patterns	Yes	No
Positioning	Yes	Yes
Freezing behavior	Yes	No
School structure	Yes	No
Horizontal/vertical distribution	No	No
Swimming speed	Yes	No
Appetite	Yes	Yes
Growth	No	Yes
Cataracts	Yes	No
Red water (shells, blood)	Yes	No
Surface activity	Yes	No
Gill cover frequency	Yes	No

The WI *sickness* was also considered. However, it is difficult to evaluate it as a simple "yes" or "no" as sickness is a wide term. It could be diseases appearing on the exterior of the fish, making it possible to detect visually. If the sickness is only visual in the fish's interior, it will not be possible to detect on camera. Sickness could evolve with time, or it could be visible only by studying changes in behavior (detectable over a longer time).

As this project was restricted to analyzing short video sequences, all WIs not visually detectable *in snapshot*s or *short video clip*s were discarded. The dataset and the methodology settled the final constraints. Because of motion blur and lack of detail in the images, WIs requiring high detail level like gill rate and skin patterns were discarded. In the specialization project underwater object detection and tracking was studied, and it was desirable to continue working with these methods. The acquired dataset contained underwater imagery from sea cages, capturing swimming fish from a close distance. Thus, the WI surface activity, that requires imagery closer to the surface, was also discarded. The same applies for WIs like school structure and horizontal/vertical distribution, as they would require an overview of the fish in the cage from a farther distance. The final choice was the WI *swimming activity*, or more specifically, *swimming velocity*.

1.1.4 Research Question

This thesis examines the combination of computer-vision and deep-learning based methods to obtain information about salmon welfare from a dataset of stereo video sequences from an industrial sea cage. Hence, the research question addressed is *How can established computer-vision and deep-learning methods be applied and combined to obtain information about salmon welfare?* This question is answered by looking more specifically at the following questions:

1. How can information about fish welfare be obtained from a dataset of stereo video sequences of farmed salmon in a sea cage? How can welfare be defined, and which welfare indicators are visually detectable from video sequences?
2. How can depth information be extracted from the underwater image dataset provided?
3. How can fish, or parts of a fish, be detected using deep-learning based object detection?
4. How can one fish be tracked through several image frames simply and efficiently?
5. Can the methods found be combined to form a system to extract information about salmon welfare?

This will be studied through the following objectives:

1. Provide a brief overview of existing work on:
   - Underwater object detection
   - Underwater object tracking
   - Underwater stereo matching and 3D reconstruction
2. Perform a literature study on fish welfare indicators and from this construct the scope for the thesis.
3. Perform a literature study on traditional and deep-learning (DL) based computer vision techniques, including:
   - Artificial neural networks
   - Convolutional neural networks
   - State-of-the-art object detection models
   - Multiple object tracking
   - Stereo matching and 3D reconstruction
4. Acquire and prepare stereo image data of farmed Atlantic salmon retrieved from industrial sea cages for object detection tasks.
5. Train and validate an deep-learning-based off-the-shelf object detection model to investigate object detection in real-time.
6. Investigate the application of a simple object tracking algorithm on the results from the object detection.

7. Perform stereo matching and 3D reconstruction.
8. Combine the chosen methods into a pipeline to retrieve 3D positions for individual fish in consecutive frames and thereby estimate swimming velocities.
9. Analyze the results from the chosen methods and identify and discuss the weaknesses of the dataset, the individual methods, and the total pipeline.
10. Suggest recommendations for future work based on the results.

1.1.5 Related Work

In recent years, fish welfare has been emphasized in the fish farming industry at the same time as vision-based techniques have evolved. Lien et al. (2019) use an aerial camera platform to determine spatial feed distribution in sea cages, as a contribution to optimizing feeding. This is achieved by counting the splashes caused by feed pellets hitting the water surface. Splash pixels are distinguished from calm water surface pixels by their increased brightness. A video analysis procedure for assessing vertical fish distribution in sea cages is presented by Stien et al. (2007). They mark fish tank walls with black lines and identify those parts of the lines that are not obstructed by fish in the individual image frames. By comparing the visual part of the line with the known extent of the lines percentage coverage is calculated, which indicates the vertical fish distribution. Ziyi et al. (2014) measure the feeding activity of farmed fish based on difference frame (e.g., the subtraction of two consecutive images) analysis.

Traditional stereo vision techniques are not new, and they have been implemented for image analysis in aquaculture for several years. As early as 2001, Serna & Ollero estimated individual fish's biomass by using a stereo setup and traditional stereo matching algorithms to estimate the depth of key points determining the fish geometry. From this, they use a simple relation to estimate the biomass. Pérez et al. (2018) use a stereo camera setup to estimate fish size. They use traditional stereo matching to correct the pairs of images and segmentation to obtain silhouettes of fish and exclude objects not fulfilling certain criteria for being a fish. From the silhouettes, the length and height for each fish are estimated. Using this combined with the depth information from stereo matching, they determine the fish's position and size in space. They achieved a maximum error of 4%. Note that these results are obtained using a physical frame that the fish must pass through while photos are taken, thereby "forcing" a controlled environment.

Deep-learning based methods have been successfully applied for detecting objects in underwater environments. Fulton et al. (2018) investigate four different state-of-the-art methods based on convolution neural networks (CNN) for the detection of marine litter. The model that performed the best in terms of accuracy was Faster R-CNN with a mAP of 81%. Another CNN-based approach is presented by Han et al. (2020) for detecting ma-

rine organisms. They use a model based on Faster R-CNN, but apply several adjustments to be able to detect very small objects and to run the model in real-time. Xu & Matzner (2018) implemented and tested a YOLO model on fish detection using three different datasets, obtaining a mAP of almost 54% when training and testing on all three datasets. An interesting finding was that the model did not generalize very well when training on two of the datasets and testing on the third. For online fish detection and tracking Li et al. (2018) proposes a combination of YOLOv3 and parallel correlation filter, achieving promising results.

1.2 Contribution

The main contribution of this thesis is an experimental pipeline demonstrating how vision-based techniques can be used and combined to obtain information about farmed salmon welfare. In specific, we developed a proof-of-concept to estimate salmon spatial movement and swimming velocity by combining deep-learning based object detection, a tracking algorithm based on IOU overlap, and 3D reconstruction by semi-global block matching. In addition, the thesis includes considerations of the visual and temporal detectability of welfare indicators for farmed Atlantic salmon, which can serve as a basis for further research on vision-based monitoring of welfare in the fish farming industry.

1.3 Thesis Outline

The thesis has now been introduced and some issues of fish welfare have been uncovered and investigated. We have looked into the challenges of underwater imaging, proposed the research question for the thesis, and presented some related work. The remaining parts of the thesis will be structured as follows:

- Chapter 2 provides a theoretical background on deep learning, focusing on neural networks and in particular convolutional neural networks.
- Chapter 3 investigates object detection and multiple object tracking methods by presenting the most important theoretical parts and providing details on some central models.
- Chapter 4 introduces the concepts behind stereo vision and goes further into camera calibration, stereo matching, and 3D reconstruction.
- Chapter 5 describes the methods used in this project, including data acquisition, data processing, detection model, image scene depth estimation, and tracking algorithm, and the implementation of the proposed experimental system.

- Chapter 6 provides an overview of the results from the individual components and the proposed pipeline.
- Chapter 7 presents a discussion about the limitations of the project and an interpretation of the results.
- Chapter 8 concludes the findings in the project and proposes recommendations for future work.

Chapter 2

Deep Neural Networks

Although the greatest strength of machine learning models is their ability to adapt and learn, there is no single model that can fit all purposes. A neural network can be designed in numerous ways, and there exist many different classes of neural networks suited for different purposes or different data structures. To understand how neural networks learn and predict and to understand which type of network is appropriate for a specific objective is crucial to obtain optimal and, even more important, valid results. The following chapter will provide the most important theory behind neural networks and, in particular, convolutional neural networks, which are the core building blocks of object detectors.

2.1 Artificial Neural Networks

Artificial neural networks, or simply neural networks (NNs), are essential when considering artificial intelligence, machine learning, and deep learning. They are inspired by the natural neural networks in the brain and consist of many elementary processing units, encouraged by biological neurons, which are interconnected to gain and preserve knowledge through learning (Wu 1992). By processing examples, NNs have the ability to adapt and construct their own rules of behavior through experience, analogous to the learning process of the human brain (Haykin 1998). In addition, neural networks are powerful tools for performing information-processing tasks as they have the ability to recognize patterns in complex datasets.

2.1.1 Basic Structure

The structures of neurons and NNs are commonly visualized as weighted directed graphs. The neuron graph model in Figure 2.1 consists of the following elements (Haykin 1998):

1. ***Weights***. The synaptic weight w_i is multiplied by the input signal x_i. A larger weight means the particular input has a larger impact on the network.
2. ***Summing point*** $\sum$ for summing the weighted input signals together with a *bias*. Analogous to a constant in a linear function, the bias b can shift the input left or right to fit the prediction better with the data.
3. ***Activation function*** f to saturate the neuron's output amplitude and to introduce nonlinearity, making it possible for the network to learn nonlinear patterns. The results are passed as input for neurons in the next layer of the network.

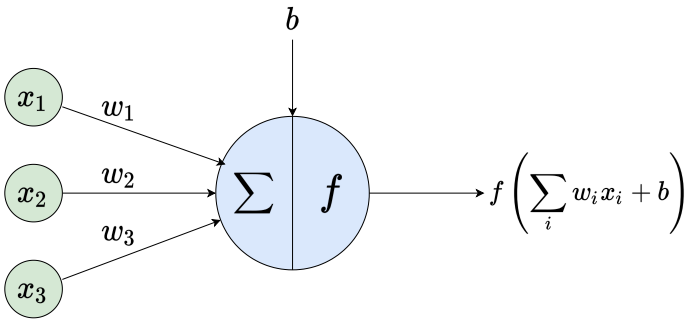


Figure 2.1: Graph model of a neuron.

A neural network contains sets of neurons structured in interconnected layers. Figure 2.2 shows a visual representation of a simple neural network. It consists of an input layer, a hidden layer, and an output layer. The network in the figure is *fully connected* since every neuron in layer l is connected to every neuron in layer $l+1$. It is a *single-layer* network because it has one hidden layer. All layers in a network that are neither the input nor the output layer are hidden layers. A network with several hidden layers is referred to as a *deep* network, giving rise to the term *deep learning*.

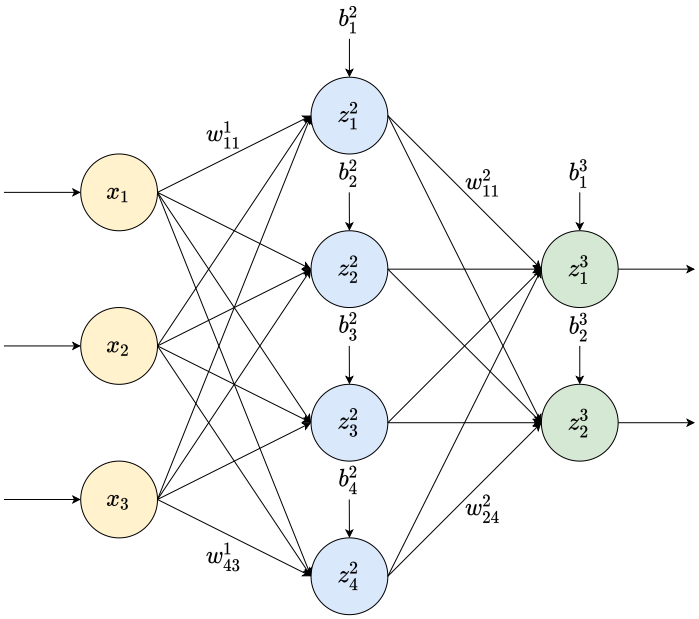


Figure 2.2: Illustration of a single-layer, fully connected neural network.

In Figure 2.2, an input signal x_i in the input layer $l = 1$ is multiplied by the weight w_{ji}^1 before it is passed to neuron j. The summation step adds the weighted input and the bias to produce the linear combination $z_j^2 = \sum_i w_{ji}^1 x_i + b_j^2$. This result is passed on to an activation function ϕ, resulting in the output $o_j^2 = \phi(z_j^2)$ which is provided as input for the next layer in the network. For an arbitrary layer l, the summing computations for a single neuron j can be generalized as

$$z_j^l = \sum_{i=0}^{m} w_{ji}^l o_i^{l-1} + b_j^l. \tag{2.1}$$

Then, after passing through the activation function, we achieve the output

$$o_j^l = \phi(z_j^l). \tag{2.2}$$

This can be extended to vector form, including all neurons spanning the layer, which gives

$$\mathbf{z}^l = \mathbf{W}^l \mathbf{o}^{l-1} + \mathbf{b}^l \tag{2.3}$$

and the output vector

$$\mathbf{o}^l = \phi(\mathbf{z}^l) = [\phi(z_1^l) \;\; \phi(z_2^l) \;\; \ldots \;\; \phi(z_j^l)]^T. \tag{2.4}$$

2.1.2 Training

To alter the weights of the connections between neurons and the biases to be optimal for a network, one must train the network. This is where the *learning* in machine learning appears.

2.1.2.1 Backpropagation

The training of a neural network is an iterative process, as visualized in Figure 2.3. The first phase, *forward propagation*, introduces training (labeled) data to the network. The labels are not exposed to the network during the forward propagation, and the network predicts and assigns labels on this data based on its current weights and biases. I.e., all neurons perform their calculations (as described in (2.1)-(2.4)) on the information received from the previous layer and pass this to neurons in the next layer. Finally, the output layer makes its prediction and outputs a label. Further, a *loss function* is used to estimate the error (loss). The output is a measure of how accurate the prediction was relative to the true label. Finally, the loss information is propagated backward in the *backpropagation* phase. This information propagates to all contributing neurons, layer by layer. Each neuron receives a part of the loss information based on how large a neuron's relative contribution had to the output and updates weight and biases accordingly. Ideally,

the network aims for zero loss, i.e., the backpropagation is to optimize a cost function C (the loss function) with respect to any weight w and bias b in the network.

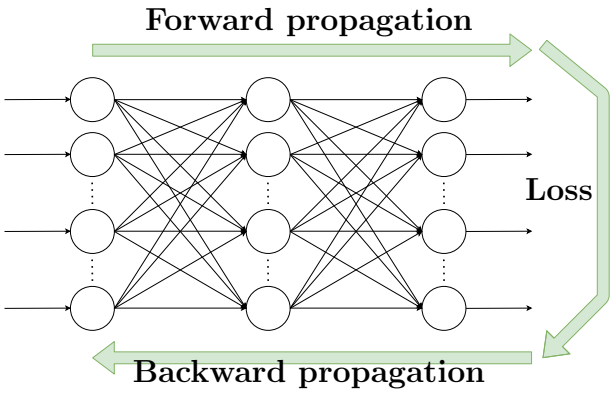


Figure 2.3: Illustration of forward- and backpropagation during training in a neural network.

An important assumption to be made about the cost function when backpropagation is applied is that the cost function can be written as an average over cost functions C_x for individual training examples, i.e.,

$$C = \frac{1}{n} \sum_x C_x. \tag{2.5}$$

This assumption is necessary because backpropagation computes the partial derivatives $\frac{\partial C_x}{\partial w}$ and $\frac{\partial C_x}{\partial b}$ for every training sample x. Then $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ are recovered by averaging over the training samples. A second important assumption is that the cost function can be written as a function of the outputs $\mathbf{o}^l$ from the network, i.e. $C = C(\mathbf{o}^l)$. Note that the desired output (label) $\mathbf{y}$ is also part of the cost function but is not considered as a variable in the individual cost functions, as its value is fixed for each sample.

Now we introduce a small change Δz_j^l to the neurons weighted input z_j^l, giving the activation output $\phi(z_j^l + \Delta z_j^l)$. This change propagates through subsequent layers of the network, changing the total cost by $\frac{\partial C}{\partial z_j^l} \Delta z_j^l$. Further, we define the error in the j^{th} neuron in the l^{th} layer, δ_j^l, as

$$\delta_j^l = \frac{\partial C}{\partial z_j^l}, \tag{2.6}$$

where z_j^l is as defined in (2.1). Backpropagation yields a procedure to compute this error for every layer and relate them to $\frac{\partial C_x}{\partial w}$ and $\frac{\partial C_x}{\partial b}$.

The error in the output layer, δ^L, can on vector form (including all neurons in the layer) be described as

$$\delta^L = \nabla_o C \odot \phi'(z^L), \tag{2.7}$$

where $\nabla_o C$ is the rate of change in the cost with respect to the output activations and $\phi'(z^L)$ is the rate of change in the activation function ϕ at z^l_j. (2.7) is necessary to compute the error in the former layers, and the error in layer l in terms of the error in layer $l+1$ is

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \phi'(z^l). \tag{2.8}$$

(2.8) can be interpreted as moving the error back through the network, which gives a measure of the error at the output in the previous layer. Combining (2.8) with (2.7) now gives the error for any layer in the network. Since the overall goal is to relate the change in the cost to the weights in the network, we can now define (2.9) and (2.10) which are the rates of change of the cost with respect to any bias and weight in the network, respectively.

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{2.9}$$

$$\frac{\partial C}{\partial w^l_{jk}} = o^{l-1}_k \delta^l_j \tag{2.10}$$

With the four equations (2.7)-(2.10) in mind, the backpropagation algorithm can now be formulated as in Algorithm 1.

Algorithm 1 Backpropagation algorithm (Nielsen 2015)

Input x**:** Set the corresponding activation o^l for the input layer.
Forward propagation: For each $l = 2, 3, ..., L$ compute $z^l = w^l o^{l-1} + b^l$ and $o^l = \phi(z^l)$
Output error δ^L**:** Compute the vector δ^L from (2.7).
Backpropagation of error: For each $l = L-1, L-2, ..., 2$ compute δ^l from (2.8).
Output: Calculate the gradient of the cost function, given by (2.9) and (2.10).

For further details on the algorithm and the derivation of the equations, see Nielsen (2015).

2.1.2.2 Gradient Descent

Now that the cost function C and the backpropagation are introduced, the next step is to look at how one can minimize the cost function. We can visualize the loss function as a landscape, where the goal is to find the global minimum. This is a comprehensive task, and there are no effective methods for finding the true global minimum. However, using the iterative technique *gradient descent* (GD) can provide the local minimum, which in most cases is sufficient. The weights are gradually changed in small increments by calculating the gradient of the loss function. The gradient $\nabla(f)$ of a scalar-valued multi-variable function $f(x_1, x_2, ..., x_D) \in \mathbb{R}^D$ is defined as

$$\nabla(f) = \left[\frac{\partial f}{\partial x_1}, \quad \frac{\partial f}{\partial x_2}, \quad ..., \quad \frac{\partial f}{\partial x_D} \right]^T, \tag{2.11}$$

i.e., the gradient captures the rate of change of the function with respect to each directional component; thus, it reveals in which direction to do the next increment. Consequently, one can update the weights and biases in the network component-wise according to

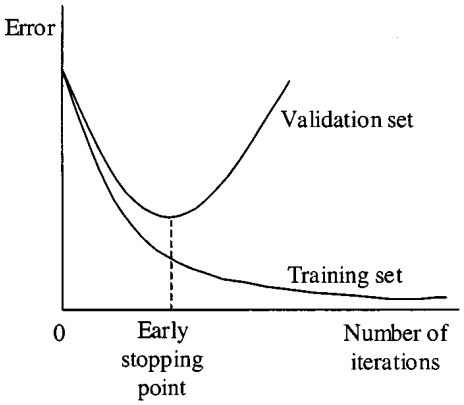
$$w^l_{jk} \to w^l_{jk}{}' = w^l_{jk} - \eta \frac{\partial C}{\partial w^l_{jk}}, \tag{2.12a}$$

$$b^l_j \to b^l_j{}' = b^l_j - \eta \frac{\partial C}{\partial b^l_j}. \tag{2.12b}$$

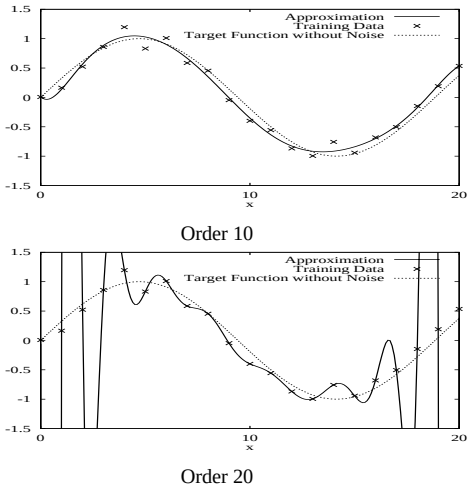
While the gradient decides in which direction one should adjust the weights and biases, the *learning rate*, η, determines the adjustment step size. It is important to choose the learning rate sufficiently, as it introduces a trade-off between fast convergence and overshooting. Further, the entire process is repeated with batches of data in consecutive iterations (epochs) until convergence or until a maximum number of epochs. In the GD approach, the gradient is computed separately for each training input and then averaged over all training inputs in one batch. This means that the computational cost increases linearly with the size n of the training dataset; hence learning will occur slowly when n gets large. To speed up the learning, one can apply *stochastic gradient descent* (SGD). The idea is to estimate the cost function gradient by evaluating the gradient of a set of randomly chosen training inputs $x_1, x_2, ..., x_m$, referred to as a *mini-batch*. If the mini-batch m is large enough, we can assume this set as representative for the entire batch. Although the estimates are influenced by statistical fluctuations and thus are not perfectly accurate, the direction will generally be sufficient to decrease the cost function and eventually find the local minimum at a lower computational cost than with the vanilla GD (Nielsen 2015).

2.1.2.3 Overfitting

Machine learning models are subjected to the problem of overfitting. Overfitting is the phenomenon where the model learns noise and detail in the training set too well; thus, it cannot generalize the patterns it learns and will perform badly when presented to new data. Figure 2.4b illustrates the concept by a polynomial approximation. We see that for order 10, the model quite accurately estimates the target function, which is the kind of result that is desirable. For order 20, the model fits the training data almost perfectly, but the interpolation is very bad between the training points.



(a) Early stopping point. Image courtesy of Gençay & Qi (2001).



(b) Example of overfitting. Image courtesy of Lawrence & Giles (2000).

Figure 2.4: Early stopping point and overfitting.

We can diagnose overfitting in a model by comparing training accuracy and validation accuracy. If the accuracy is much better on the training set than on the validation set, the model is likely overfitting. Now, how can we prevent this phenomenon? An obvious solution is to use a larger training set with more variance in the training data, but this is not always realizable. Another feasible method is *early stopping*. We identify the early stopping point as the point in the training where the validation accuracy goes from improving to perform worse, as visualized in Figure 2.4a, and then stop the training at this point. Data augmentation and regularization methods are also used to prevent overfitting.

2.1.2.4 Data Augmentation

Data augmentation is a strategy applied to neural network training datasets to obtain larger diversity in the data without collecting any new data. One can transform the images in position or in color, or one can generate synthetic images by using a Generative Adversarial Network (GAN). The former is the least time-consuming and computationally inexpensive and thus most relevant for this case. Many techniques exist for transforming images, e.g., tilt, shear, flip, noise, brightness, hue, and saturation. Some examples are shown in Figure 2.5.

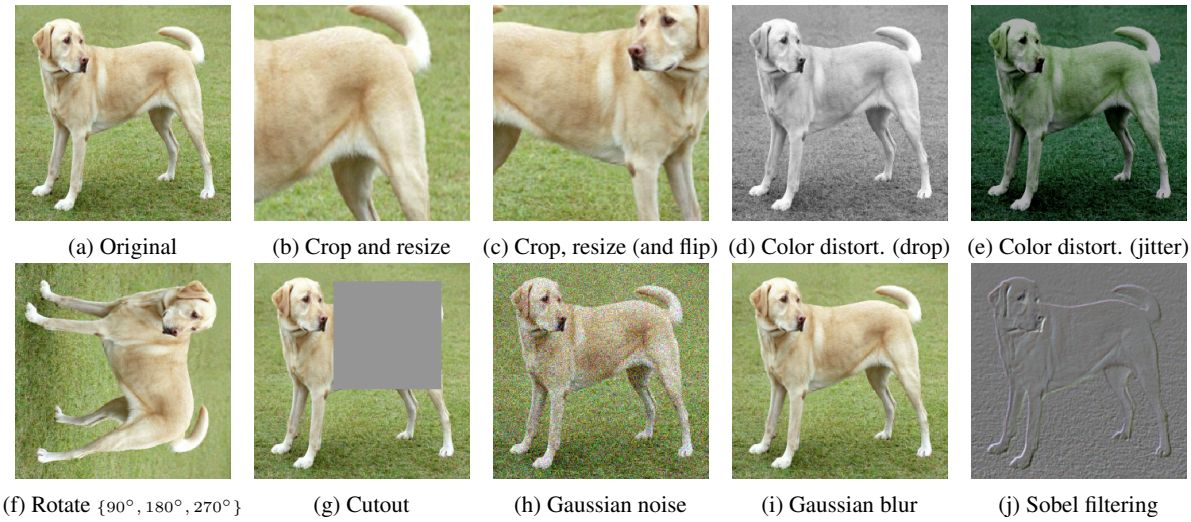


Figure 2.5: Examples of data augmentation techniques. Image courtesy by Chen et al. (2020).

2.2 Convolutional Neural Networks

A convolutional neural network (CNN) is a special class of NN that utilizes the mathematical *convolution* operation rather than matrix multiplication. The idea behind CNNs is, analogous to traditional NNs, also motivated by neurobiology. Hubel & Wiesel (1959) found that single neurons in cats' visual cortex were activated when triggered by visual stimulus. They referred to these neurons as *receptive fields*. They suggested that visual processing in the brain is a hierarchical process of feature detectors that identify basic features at the lower levels and combine these to more complex features at the higher levels. Fukushima (1980) proposed a neural network to recognize patterns, directly inspired by Hubel and Wiesel, using convolutional layers that exploit receptive fields to find the average of patches of an image. This concept is shown in Figure 2.6. One further improved the idea of CNNs in the following years by applying backpropagation (LeCun et al. 1989), max-pooling (Weng et al. 1993), and gradient-based learning (Lecun et al. 1998), where the latter can be seen as the birth of modern CNNs. In 2012, the popularity of CNNs increased drastically after a much deeper and wider CNN, AlexNet (Krizhevsky et al. 2017), achieved state-of-the-art performance in the ImageNet competition, learning much more complex features than its predecessors.

As of 2020, CNNs are widely used for image processing, image classification, and object detection. Their power lies in capturing spatial and temporal dependencies, which is essential when looking for patterns in images. To achieve this, the CNN architecture typically consists of three main layers, *convolutional layer*; *pooling layer*; and *fully connected layer*. The *rectified linear unit* is commonly used as an activation function in CNNs to

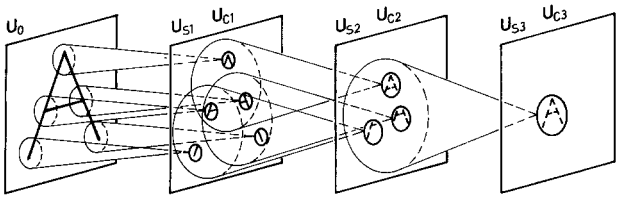


Figure 2.6: Fukushima's concept of convolutional networks from 1980. Basic features are extracted by receptive fields at the lower layers (left) and combined to more complex features at the higher levels (right). Image courtesy of Fukushima (1980).

introduce nonlinearity. These components are described in the following subsections.

2.2.1 Convolutional Layer

The convolutional layers are the core building blocks of a CNN. They embody the feature extraction in the network; hence, they learn the feature representations of the input images (Rawat & Wang 2017). Convolution, often denoted $f * g$, is a mathematical operation composed of two real-valued functions f and g. It can be defined as

$$(f * g)(t) \coloneqq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau. \tag{2.13}$$

If we see $f(\tau)$ as our input and $g(\tau)$ as a weighting function, (2.13) is the weighted average of f at time instant t weighted by g. This gives inputs that are more recent in time a higher weight. Looking at convolution in the context of neural networks, we can write a discrete version of (2.13):

$$s(t) = (x * w)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t-\tau). \tag{2.14}$$

The weighting function $w(\tau)$ is often referred to as a *kernel*, and the output $s(t)$ as the *feature map*. In the context of image recognition, the inputs are usually multidimensional arrays of data representing pixels, typically two-dimensional (width, height) or three-dimensional (width, height, depth), where the depth dimension contains the color channels of the pixel (normally RGB). The kernel is often of the same dimension as the input, typically with a smaller spatial extent than the input but with equal depth. If we look at a two-dimensional input image $\mathbf{I}$, the kernel $\mathbf{K}$ is also two-dimensional, and we can extend (2.14) to, as described in Goodfellow et al. (2016),

$$\mathbf{S}(i,j) = (\mathbf{I} * \mathbf{K})(i,j) = \sum_m \sum_n \mathbf{I}(m,n)\mathbf{K}(i-m, j-n). \tag{2.15}$$

Now, while (2.14) captures the temporal relationship by weighting more recent inputs higher, (2.15) captures the cornerstone in CNNs, namely the spatial relationship between individual pixels in an image. A visual approach by matrices may be more intuitive in the context of image recognition. Figure 2.7 shows visually how the convolution operation $\mathbf{I} * \mathbf{K}$ is executed on an input $\mathbf{I}$ and a kernel $\mathbf{K}$ to produce the output. Notice that the operation computes the dot product of the kernel and a local part of the input of the same size as the kernel. The kernel is slid across the input, and the dot product is computed for each part.

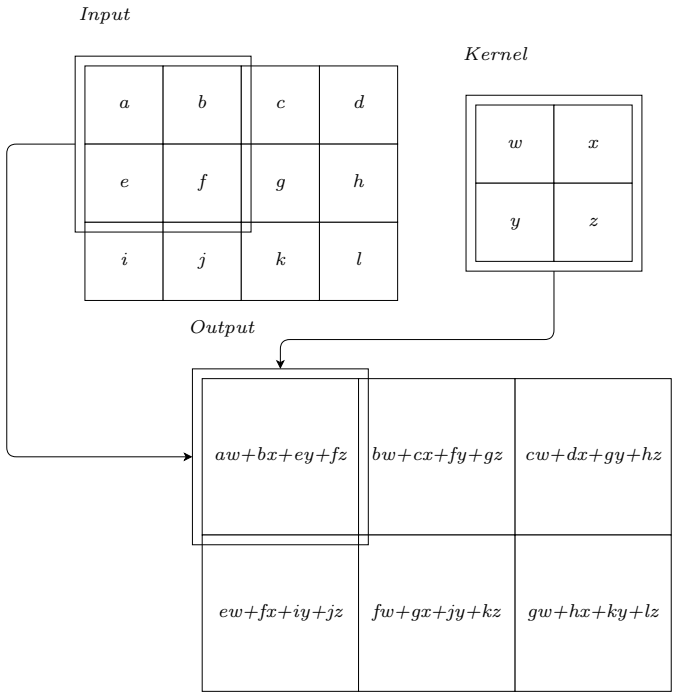


Figure 2.7: A visual explanation of convolution. Based on Goodfellow et al. (2016).

In Figure 2.7, the kernel is slid one pixel at a time to produce the output. The number of pixels slid each time, the *stride* $\mathbf{S}$, is one of the hyperparameters for the convolutional layer. Other tunable parameters include $\mathbf{K}$, the *number of kernels*; $\mathbf{F}$, the *receptive field size*, i.e., the kernel spatial extent; and $\mathbf{P}$, the size of *zero padding*, which controls the dimension of the output and preserves information at the boundaries. Figure 2.8 shows an example of the output after convolution with two different kernels capturing edges.

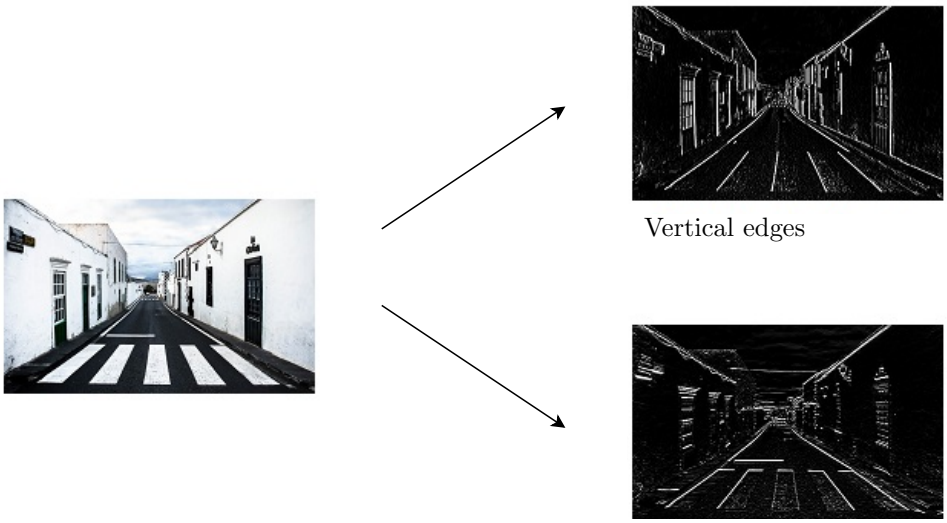


Figure 2.8: Edge detection using two different kernels, one capturing vertical edges and one capturing horizontal edges. By combining the results we obtain the majority of edges in the image. Figure retrieved from *CNN Edge detection* (2018) and modified.

2.2.2 Rectified Linear Unit (ReLU)

The activation function used in CNNs usually is the rectified linear unit (ReLU). It is defined as

$$ReLU(x) = \max(0, x). \tag{2.16}$$

Hence, it is mathematically fundamental and thus computationally inexpensive. Mathematically this also means that its derivative is either 0 or 1 for negative and positive inputs, respectively. In other words, it does not saturate since the gradient is always 1 if the neuron activates, which prevents the vanishing gradient problem (that the gradient becomes vanishing small as it recedes from the final layer). It is also convenient as it is easy to evaluate in the backpropagation equations (recall (2.7) and (2.8)).

2.2.3 Pooling Layer

A limitation of the resulting feature map after convolution is that it contains the exact locations of the features. If the features in the input change position, this can result in a different feature map. The pooling layer downsamples the feature maps by summarizing the presence of features in smaller patches of the feature map. This creates a lower resolution feature map but preserves the main structures. The intricate structures are left out, preventing overfitting (see section 2.1.2.3) in the learning phase. Hence, the

model becomes more robust to variations in the position of the features in the input image (Brownlee 2019). Another important function of the downsampling is that it reduces the number of parameters, consequently, it reduces the computational load. The pooling layer's effects are usually added periodically between the convolutional layers, after introducing nonlinearity with ReLU. Several types of pooling exist, resulting in different filtering of the feature map. The most common type of pooling is max pooling, which summarizes the most activated presence of the feature. It extracts the maximum value in each window, hence, it preserves the best fit of the feature in each window. The pooling operation requires two hyperparameters: stride $\mathbf{S}$ and spatial extent $\mathbf{F}$. The example in Figure 2.9 uses $\mathbf{S} = 2$ and $\mathbf{F} = 2 \times 2$. As for many aspects in neural networks, there is also a trade-off here: a small spatial extent will preserve more information but will also decrease the effect of the pooling layer, while a larger spatial extent leads to loss of information.

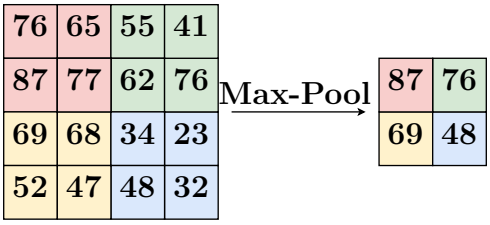


Figure 2.9: Max-pooling operation.

2.2.4 Fully Connected Layer

The final stage of a CNN is usually the fully connected (FC) layers, which perform the classification task. FC layers are the main building blocks of traditional neural networks. They are characterized by the full connectivity between neurons in one layer and the activations in the previous layer, as demonstrated in Figure 2.2. These layers take one-dimensional lists as input. Consequently, one must flatten the higher-dimensional feature map prior to the FC layers. The general architecture of a CNN can be seen in Figure 2.10.

The final FC layer provides the classification score of the image, which is a measure of the relative activation strength of a class compared to the other classes. This score is normally interpreted as a probability, meaning its value typically lies in the interval $[0,\ 1]$. To obtain this, a normalized softmax activation function is usually used,

$$\sigma(\mathbf{z}^l)_k = \frac{e^{z_k}}{\sum_{i=1}^{j} e^{z_i}} \text{ for } k = 1, ..., j, \tag{2.17}$$

which normalizes the input vector $\mathbf{z}^l$ into a vector $\mathbf{o}^l = \sigma(\mathbf{z}^l)$ of length j, where j is the number of neurons in the final FC layer, and thus also the number of classes. The entries

in $\mathbf{o}^l$ will sum up to 1, meaning we can interpret o_k^l as the *probability of the input belonging to class k*.

2.2.5 Overall View

We have introduced the main parts of a convolutional neural network, and it remains to take a step back and look at a typical CNN architecture. Several types of CNN architectures exist, but the most common approach is to stack the different layers onto each other in the order shown in Figure 2.10.

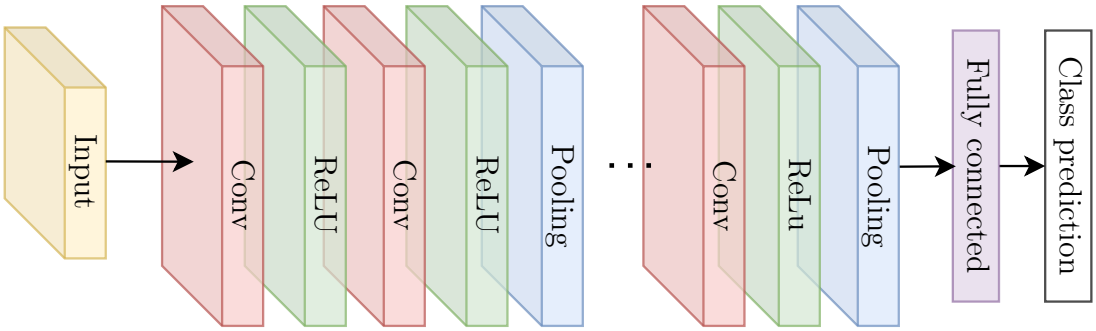


Figure 2.10: Layers of a typical CNN.

Notice the repeating pattern `Convolution-ReLU-Pooling` between the input layer and the fully connected output layer. For every repetition of this pattern, one will obtain more complex features. The output of the low layers (to the left) will be simple patterns like edges and bright spots. The middle layers will contain more prominent features like, in the context of, e.g., face detection, a nose, a mouth, or an eye, while the output of the high layers (to the right) will contain more sophisticated and complex patterns like an entire face. This is analogous to the network presented in Figure 2.6. In addition, we can add more fully connected layers before the output layer to make the network learn even more sophisticated and complex combinations of features.

2.2.6 Transfer Learning

A CNN is rarely trained from scratch on custom data as it is very time-consuming and would require vast amounts of data. Objects tend to share many of the same low-level features like edges, corners, or bright spots. This can be exploited to pre-train the network for general patterns on large, general, publicly available datasets such as ImageNet (Deng et al. 2009), before training on a custom dataset to learn the more complex and specific features. This is called *transfer learning*, and this technique is widely used for training neural network models.

Chapter 3

Object Detection and Tracking

Convolutional neural networks serve as a basis for most modern object detection algorithms. This chapter will explain the concept of object detection further and present some state-of-the-art models. Additionally, the chosen model for object detection in this thesis, YOLOv4, will be investigated more in detail. Finally, we will dive deeper into multiple object tracking and explore some essential methods.

3.1 Object Detection

Object detection is a computer vision technique that aims to detect, locate, and identify object instances of certain classes in images. The goal is to perform two primary tasks:

1. Locate an arbitrary number of objects in an image.
2. Estimate bounding boxes for, and assign a class to, each object.

3.1.1 Basic Structure

On a high level, an object detection model will have architecture, as shown in Figure 3.1. In the feature extractor, a CNN produces a feature map from an image as described in section 2.2, which is further fed into a detection part that localizes and classifies objects of interest. Figure 3.2 visualizes the output from an object detector: bounding box predictions describing the localization of the objects in the image and class predictions for each bounding box.

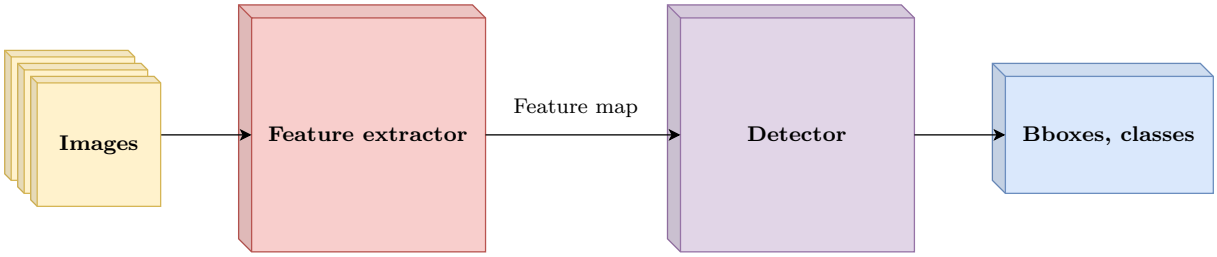


Figure 3.1: High-level architecture of an object detector.

The most basic form of object detection is a rather comprehensive technique, the so-called *sliding window* technique. Basically, a CNN performs image classification on a wide range of different crops of the image. The CNN is first trained for classification of images containing particular objects of interest. Then, detection is performed by sliding windows of different sizes over the entire image, usually starting from the top-left corner, going in small steps to the bottom-right corner, and classifying each crop (window). Hence, a single image will require many forward passes through the CNN, meaning this technique demands a high computational load. In the past years, there has been an evolution in the performance of object detectors, and newer detection models perform significantly faster and more accurately. Roughly, we can divide the modern object detection models into two main parts: one-stage and two-stage detectors. The principal difference is that one-stage detectors only contain *a single* neural network, completing the two tasks 1 and 2 in one step, while two-stage detectors separate the two tasks.

Figure 3.2: Example result from object detection. The outputs are bounding boxes and class predictions for each object of interest in the image. Image courtesy by Redmon et al. (2016).

Two-Stage Detectors

Two-stage detection models are usually R-CNN based, including R-CNN, Fast R-CNN, Faster R-CNN, and Mask R-CNN. The term *R-CNN* means region-based CNN and basically describes what the two stages do:

1. **Region-proposal**. Arbitrary objects are found and located in the image through a region-proposal system which can be based on classical CV methods (for instance,

selective search) or a region-proposal neural network.
2. **CNN**. The results from stage 1 (region candidates) are passed to a CNN to classify the object each of them contains.

These models prioritize and have been shown to obtain high accuracy, but they tend to be complex and not very fast due to the high computational load of the two stages.

One-Stage Detectors

The most common types of one-stage object detection models are the Single Shot Detector (SSD) and the You Only Look Once (YOLO) detectors. These treat the object detection problem as a regression problem. Hence, they only use a single deep neural network and skip the region-proposal stage. The network runs detection directly over a dense sampling of possible locations, learning the class probabilities and bounding box coordinates. These models typically have lower accuracy than two-stage detectors. However, they are significantly simpler and faster; thus, they are more suitable for real-time object detection systems.

3.1.2 State-of-the-Art Models

Which models represent state-of-the-art changes quickly due to the rapid evolution of the field, but in this section, some state-of-the-art models as of 2020/2021 will be presented.

Faster R-CNN

It should be noted that to categorize Faster R-CNN as state-of-the-art in 2021 is far-fetched, but it is an essential contribution to the field of object detection. The two-stage R-CNN-based models were introduced by Girshick et al. (2013) with R-CNN. The idea was to use a selective search algorithm to propose regions of interest (ROI) and feed each of them into a CNN to produce feature maps. Then, each feature map was further passed into a support vector machine (SVM) to be classified. Finally, linear regression is applied to fit the bounding boxes tightly to the detected and classified objects. Although the R-CNN model performed well at the time, it had some essential drawbacks. Firstly, the selective search algorithm is inefficient. Secondly, the multi-stage approach requires each stage to be trained separately, and the results of each stage are to be cached on disk to train the next stage. Lastly, each region proposal must be fed into the CNN for feature extraction, meaning several passes are required for each image. This makes it unfeasible to detect objects in real-time. In 2015 an improved version, Fast R-CNN (Girshick 2015), intended to solve the latter problem and part of the second problem. They proposed an

ROI pooling layer, where the features inside any valid ROI are converted to a fixed-size vector by using max pooling. This made it possible to share CNN computations across all proposals in an image, consequently, each image only needs one forward pass. Also, extracted features from the CNN were not cached anymore, meaning the extensive disk space required in R-CNN was discarded. The result was a faster and more accurate object detection model, but the first problem, the region-proposal stage, was still a barrier. Faster R-CNN (Ren et al. 2015) extends Fast R-CNN by introducing a region-proposal network (RPN) to replace the selective search algorithm. Specifically, the same CNN that is used for feature extraction is used for region proposal, drastically reducing the computational load. Faster R-CNN performs well in terms of accuracy, but as stated by the authors, it is still not fast enough for real-time detection.

Mask R-CNN

Mask R-CNN (He et al. 2017) extends Faster R-CNN by adding instance segmentation to create masks on the detected objects. The ROI pooling in Faster R-CNN is replaced by the so-called ROI Align, which is better at preserving spatial information necessary for creating masks lost in ROI pooling. After ROI Align, the result is fed into two convolutional layers creating the final pixel-level masks.

YOLO (You Only Look Once)

The first YOLO version was published in 2016 and was the first object detection model to perform object detection using a single CNN on the entire image only once. The object detection problem is treated as a regression problem to spatially separated bounding boxes and associated class probabilities (Redmon et al. 2016). The general YOLO model divides the input image into a grid of cells. As shown in Figure 3.5, each cell is responsible for some number of prior bounding boxes, called *anchor boxes*, with objectness scores and class probabilities for each box. The anchor boxes form the basis for the bounding box prediction, in which the final prediction consists of predicting offsets from these predefined boxes and then use non-max suppression (NMS). The objectness scores represent how confident the bounding box is about the presence of an object in a specific class. The class probability map is responsible for finding out the class of the possible object, whether or not an object is present. The combined result of these two is a final score representing the probability that a bounding box contains a specific type of object. From the first YOLO version, several improvements have been published in which the 2020 YOLOv4 (Bochkovskiy et al. 2020) is the latest published release. The first version struggled to detect smaller objects appearing in groups, and there were problems with accuracy and incorrect localization. YOLOv2 (Redmon & Farhadi 2016) led to significantly improved

accuracy with the introduction of the *Darknet-19* feature extraction network. The next version, YOLOv3 (Redmon & Farhadi 2018), intended to detect smaller objects better by proposing multi-scale prediction. However, this required a higher computational load. YOLOv4 aims to be faster while still providing multi-scale prediction. It yields a good trade-off between accuracy and speed compared to other models, hence it is a good choice for real-time accurate object detection. It is also optimized for GPU parallel computation in that it is possible to run in real-time on a conventional GPU. YOLOv4 is investigated more in detail in section 3.2.

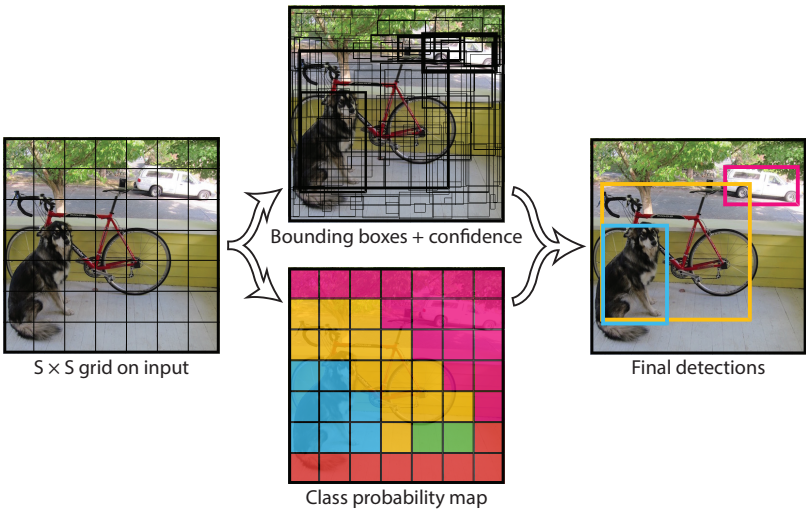


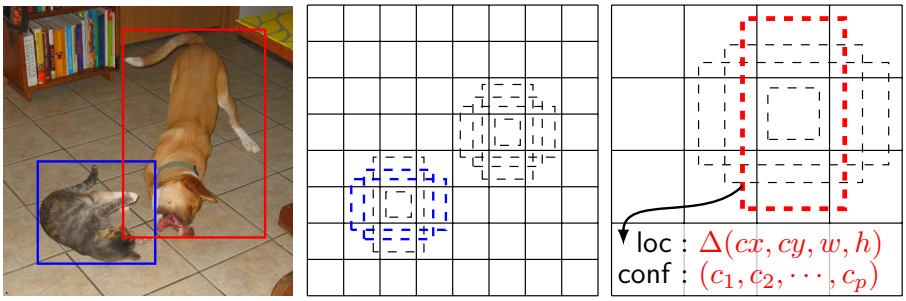
Figure 3.3: YOLO bounding box and class prediction. Image courtesy of Redmon et al. (2016).

SSD (Single Shot Multibox Detector)

Similar to YOLO, SSD (Liu et al. 2015) only uses a single shot to detect multiple objects in an image. The feature extraction network is the same as in Faster R-CNN, namely the VGG-16 network. As YOLO, SSD utilizes the concept of anchor boxes, but where YOLO is limited in that its grid cells are of fixed aspect ratios, SSD uses grids of different aspect ratios and scales. Further, SSD adds more convolutional layers instead of YOLO's fully connected layers in the detection stage. The result is a network handling differently sized objects better and achieving more tightly-fitted bounding boxes. However, the latest YOLO versions are still superior in terms of speed according to Redmon & Farhadi (2018).

RetinaNet

As previously stated, the two-stage R-CNN-based models are more accurate than one-stage detectors. One of the biggest problems in the latter is that there is an extreme



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 3.4: SSD training process. Image courtesy of Liu et al. (2015).

foreground-background class imbalance during training. These detectors do not distinguish between easy examples, such as an empty white background, and complex examples, such as a noisy, textured background. This leads to the loss being dominated by easy examples. The problem is addressed in RetinaNet by introducing *focal loss*, i.e., reshaping the standard cross-entropy loss to down-weight the easy examples and focus on training the complex examples (Goyal et al. 2017). This approach improves accuracy, but SSD and YOLO perform faster.

EfficientDet

On a high level, EfficientDet maintains the same structure as the previously mentioned one-stage detectors. The main contribution of EfficientDet was to build a scalable detection model with both higher accuracy and better efficiency compared to models such as YOLOv3, RetinaNet, and Mask R-CNN. Tan et al. (2019) compared different design choices for all parts of the model architecture and finally chose an optimized structure, which made EfficientDet consistently achieve better accuracy and efficiency than prior detectors. Note that YOLOv4 was published around the same time as EfficientDet as an important opponent.

3.2 YOLOv4

YOLOv4 is at the time of writing a state-of-the-art one-stage object detection model. It yields a good trade-off between accuracy and speed compared to other models, and by using the Darknet framework, it is a convenient off-the-shelf solution. This makes the model sufficient and applicable for industrial use. This section aims to provide a more detailed description of YOLOv4 as it is the model chosen for implementation. First, we will explain the most critical parts of the YOLOv4 architecture and concepts. Then, we will look at the improvements in YOLOv4 compared to previous versions and finally argue

why this version is the most suitable for this project.

YOLOv4 is an important improvement of YOLOv3. The model can generally be divided into three parts: *backbone*, *neck*, and *head*. Together, the backbone and the neck performs feature extraction and aggregation, while the head is responsible for the detection. Feature maps are obtained at three different scales in the feature extraction stage and then passed into the detector, where bounding boxes and classes are predicted for each scale.

Grid Cells

The general YOLO model divides the input image into an $S \times S$ grid of cells. This makes the detection more efficient than, e.g., R-CNN-based algorithms that run bounding box and class predictions over the same image several times to capture all ROIs. Each grid cell is responsible for predicting a number of bounding boxes with objectness scores and class probabilities, as illustrated in Figure 3.5. Note that one cell can only make predictions for one object.

Bounding Box Prediction

Using "brute force" to predict and learn bounding box sizes directly is inefficient and would make the network converge slowly. In general, most bounding boxes will have specific aspect ratios. One can utilize this to determine a set of prior bounding boxes, or *anchor boxes*, that the model can predict the offset from instead of directly predicting bounding box width, height, and location. K-means clustering is applied to find a selection of anchors that best represent the dataset. YOLOv3 and YOLOv4 use nine different anchor boxes, 3 for each scale.

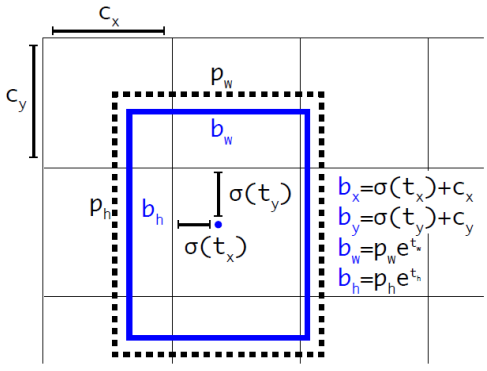


Figure 3.5: Anchor box and predicted offsets. Width and height of the bounding box are predicted as offsets from cluster centroids. The center coordinates of the box relative to the top left corner of the cell are found by using a sigmoid function, which forces the output to be between 0 and 1. Image courtesy of Redmon & Farhadi (2016).

The network predicts 4 offset coordinates for each anchor box, namely t_x, t_y, t_w, and t_h. If (c_x, c_y) is the offset of the cell from the top left corner of the image, and if p_w, p_h are width and height of the anchor box, the bounding box coordinate predictions correspond to:

$$b_x = \sigma(t_x) + c_x \tag{3.1a}$$
$$b_y = \sigma(t_y) + c_y \tag{3.1b}$$
$$b_w = p_w e^{t_w} \tag{3.1c}$$
$$b_h = p_h e^{t_h} \tag{3.1d}$$

I.e., YOLO actually predicts center offsets relative to the top left of the cell and dimension offsets relative to the prior bounding box size. Further, an objectness score is predicted for each bounding box using logistic regression. This score represents the probability that there is an object contained in the bounding box. If the bounding box prior overlaps a ground truth bounding box by more than any other bounding box prior, the objectness is set to 1. If a bounding box prior overlaps a ground truth object by more than an IOU threshold of 0.5 but is not the best overlap, the prediction is ignored (Redmon & Farhadi 2018).

Class Prediction

Each box predicts the probability of it belonging to the different classes using multi-label classification. Each cell in the $S \times S$ grid will predict C conditional class probabilities $P(c_i|object)$, i.e. the probability of class c_i given that the bounding box contains an object. The conditional class probabilities are multiplied by each of the cells' bounding box objectness score to yield each box's class probability score. This is further multiplied by the IOU of the predicted bounding box and the ground truth bounding box to obtain the class-specific confidence score of the bounding box, which encodes both the probability of that class appearing in the box and how well the predicted box fits the object (Redmon et al. 2016). While previous versions used a softmax for classification, assuming all classes mutually exclusive, YOLOv3 and YOLOv4 use independent logistic classifiers. This opens the door for multi-label classification, which is useful for overlapping classes like, e.g., Woman and Person.

Loss Function

A substantial improvement in YOLOv3 was the complex loss function which used binary cross-entropy for objectness and classification loss and mean square error (MSE) for bounding box offset loss. By using MSE, one treats the bounding box coordinates

as independent variables, but in fact, it does not consider the integrity of the object itself (Bochkovskiy et al. 2020). Thus, YOLOv4 also takes the IOU of the bounding box prediction and the ground truth into consideration. However, the IOU between two non-overlapping boxes would be zero, so using only IOU in the loss function would not help when the predicted box is close to but not overlapping a ground truth box. To solve this, they use something called CIOU-loss, which considers three geometric factors: overlap area, central point distance, and aspect ratio. This leads to faster convergence and better performance (Zheng et al. 2019).

Feature Extraction

YOLOv3 introduced a new feature extraction network called Darknet-53. This is a hybrid of the YOLOv2 backbone Darknet-19 and residual networks. YOLOv2 struggled with detecting small objects, and YOLOv3 aims to solve this problem by producing feature maps at three different scales (from the three last residual blocks) in the feature extractor. The feature maps are fed forward at different locations in the network to detect small, medium, and large-sized objects. Darknet-53 is a 53-layer CNN that is pretrained on ImageNet. In general, objects have a lot of similar features like edges, bright spots, and corners. Pretraining on a large, general dataset like this lets the network learn general patterns and prevents overfitting. In YOLOv4, Darknet-53 is improved by applying a CSPNet (Cross Stage Partial Network) strategy. We will not go through the details of this strategy, but the essence is that it deals with the following problems:

- *Strengthening the learning ability of a CNN.*
- *Removing computational bottlenecks.*
- *Reducing memory costs.*

CSPNet aims to solve these problems by allowing more gradient flow through the network by eliminating repeated gradient information and thus enable the network to be less computationally heavy. This is employed by separating the feature map of the base layer into two parts. One part goes through a dense block and a transition layer and merges with the other part through a cross-stage hierarchy (split-merge strategy). An illustration of the CSPNet strategy is shown in Figure 3.6 and more details are provided in Wang et al. (2020).

Multi-Scale Prediction

In YOLOv3, multi-scale prediction was introduced to improve the detection of objects of different sizes, maintained in YOLOv4. As explained in the previous section, feature maps at three different scales are fed from the backbone to 3 branches in the neck. At

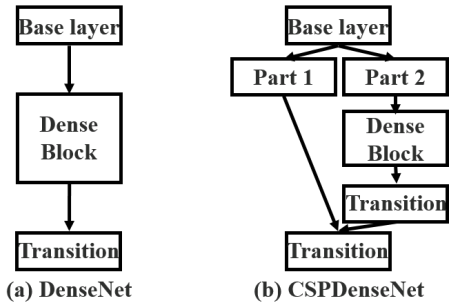


Figure 3.6: An example of the application of the CSPNet strategy (here, on DenseNet (Huang et al. 2017)). Image courtesy of Wang et al. (2020).

medium and small scale (medium and high resolution), the feature maps are concatenated with the previous feature maps to benefit from larger-scale detection and pull out finer-grained information. While YOLOv3 uses a modified FPN (Dollár et al. 2017) in the neck, YOLOv4 implements a modified PANet (Liu et al. 2018) architecture for this purpose. Model design in preliminary DL was simple: each layer only took input from the previous layer, similar to what is explained in section 2.2. This means localized information might be lost in the high layers, including essential information for fine-tuning the prediction. In modern DL, several layers can be connected, e.g., as in DenseNet (Huang et al. 2017), where all layers are connected to each other. Figure 3.7 shows the PANet for object detection. In (a), a red line shows how the FPN moves the localized spatial information upward. In reality, the red line goes through around ten layers to reach the top. PANet, however, introduces a shortcut, shown in (b) with a green line. This line only goes through around ten layers to get to the top layer. This way, fine-grained localized information is made available to the top layers.

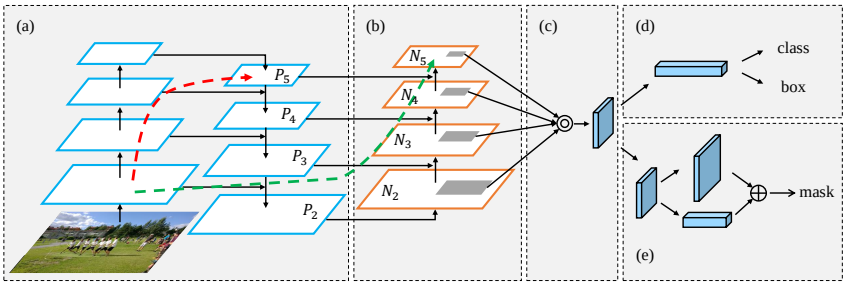


Figure 3.7: (a) Information propagation in FPN, red line. (b) Information propagation in PANet, green line. Image courtesy of Liu et al. (2018).

The original PANet *adds* neighboring layers together, but the YOLOv4 version concatenates them, as shown in Figure 3.8. In FPN, the objects are detected independently at different scale levels, which may create repeated predictions. Also, it might not exploit information from other feature maps. PAN, on the other hand, fuses information from all layers first to prevent this.

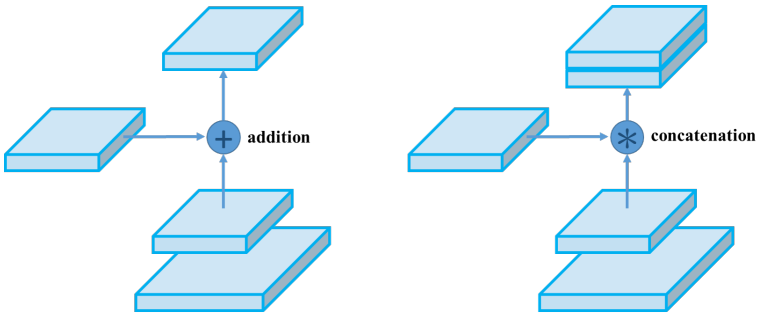


Figure 3.8: Original PAN vs. YOLOv4 modified PANet. Image courtesy of Bochkovskiy et al. (2020).

3.3 Detection-Based Multiple Object Tracking

Object tracking is to locate a moving object in consecutive image frames over time. From an initial set of bounding boxes for each frame, an object tracking algorithm aims to identify each object and track them as they move across image frames, maintaining their identity.

Object tracking can be performed on single objects or multiple objects. As our purpose is to track fish in a sea cage, we will focus on multiple object tracking (MOT) methods. Further, object tracking can be detection-based or detection-free, where the detection-based methods use results (bounding boxes) provided from an object detector. Detection-free algorithms require objects to be marked manually in the first frame, and then, they locate these objects in consecutive frames. The former is more popular because new objects are discovered and disappearing objects are terminated automatically, while the latter cannot deal with objects that disappear. However, an important drawback with the detection-based approach is that its performance is highly dependent on the object detection results (Luo et al. 2014).

An important aspect is to distinguish between online (real-time) and offline methods. Online methods must be able to track on a frame-to-frame basis, meaning they can only track based on the current frame and the past frames. Offline methods, however, can consider the whole timeline of frames. Classical object tracking algorithms involve well-established methods such as Kalman filter and optical flow. Further, as deep neural networks have evolved, they have also been applied for object tracking. Object tracking algorithms have a wide range of applications and can be used as a basis for calculating physical properties such as the direction or velocity of an object. This section will take a closer look at some non-deep learning, multi-object tracking algorithms that utilize results from an object detector and that can be applied for online purposes. First, two classical and well-established methods are introduced, followed by a more straightforward concept

based on bounding boxes and IOU.

3.3.1 Kalman Filter

Mathematically, the Kalman filter (KF) is a recursive state estimator that can predict and correct a wide range of linear processes where the state is assumed to be Gaussian distributed. The objective of the KF is to provide an optimal estimate at each time step, based on a motion model (with process noise) of the objects and several noisy measurements of the objects' positions. As long as all noise is Gaussian, optimality is guaranteed (Shantaiya et al. 2015). For MOT, in particular, the individual state of each object is usually composed of its position and velocity, and the measurements could be some properties of bounding boxes provided by an object detector, e.g., the center coordinates.

The KF model requires a proper motion model of the objects and a method (usually an optimization method, like the Hungarian algorithm) for matching predictions to their corresponding measurements for each step of prediction and correction. However, it does not need any semantic information from the image itself. It guarantees global optimality, especially its power lies in its prediction ability, which is essential for object occlusion.

3.3.2 Optical Flow

The concept of optical flow is, as many other concepts in machine vision, inspired by biological processes. James J. Gibson introduced optical flow in the 1940s to describe the visual stimulus presented to animals when moving through the world (Cutting 2000). Transferred to computer vision, optical flow is the *apparent* motion of the brightness variations in consecutive image frames, i.e., motion is caused by the relative movement between the observer and the scene. That means apparent motion can occur although no actual motion is present, e.g., consider the case of a moving ball in fixed lightning versus a stationary ball in moving lightning. The method's main goal is to capture pixel-wise image motion, achieved by estimating the apparent motion field between every pair of consecutive image frames. The most basic assumption of optical flow is that when a point $\mathbf{x} = (x, y)$ at time instant t moves to a point $(\mathbf{x} + d\mathbf{x})$ at time instant $t + dt$, the intensity of the point does not change. This is referred to as the *constant intensity assumption*:

$$f(\mathbf{x} + d\mathbf{x}, t + dt) = f(\mathbf{x}, t). \tag{3.2}$$

We have the relationship $d\mathbf{x} = \mathbf{v}dt$, where $\mathbf{v}$ is the velocity vector of $\mathbf{x}$ at time t, which is referred to as the optical flow vector. This allows rewriting (3.2) to

$$f(\mathbf{x} + \mathbf{v}dt, t + dt) = f(\mathbf{x}, t). \tag{3.3}$$

Further, dt is assumed infinitesimally small, and thus (3.3) can be approximated by first-order Taylor expansion to

$$f(\mathbf{x}, t) + (\nabla f)(\mathbf{x}, t)^T \mathbf{v} dt + f_t(\mathbf{x}, t) dt = f(\mathbf{x}, t), \tag{3.4}$$

which is equal to

$$(\nabla f)(\mathbf{x}, t)^T \mathbf{v} dt + f_t(\mathbf{x}, t) dt = 0. \tag{3.5}$$

The spatial gradient ∇f and the temporal derivative f_t can be approximated when given a sequence of images. However, the result is still an underdetermined problem, as the two elements of $\mathbf{v}$ are unknown, and we only have *one* equation. This is commonly referred to as the *aperture problem*, shown in Figure 3.9. Several methods aim to solve this problem, e.g., the Lucas-Kanade method, which uses the assumption that a point will move in the same direction as its neighbors and thus has the same optical flow vector. More details are provided in van den Boomgaard (2017).

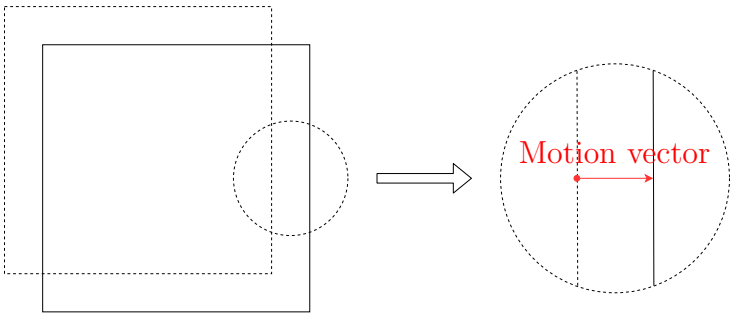


Figure 3.9: The aperture problem. The line appears to be moving to the right when viewing through the aperture, but is in reality also moving down. It is not possible to determine the correct direction of movement unless the ends of the line are visible.

We can distinguish between *dense* and *sparse* (feature-based) optical flow. The former tracks motion in the entire image, while the latter tracks specific points or features of interest (e.g., corner points or edges). Thus, when bounding boxes are provided from an object detector, one can, for instance, pass the center points of the bounding boxes to a sparse optical flow algorithm to track the boxes' movements.

3.3.3 IOU Tracking

The previously mentioned methods for tracking objects include sophisticated tasks such as modeling object motion and using image semantic information to track motion in subsequent image frames. Tracking methods based on deep learning introduce complexity that may not be necessary for many cases. Bochinski et al. (2017) propose and assess a much simpler method, assuming an object detector already determines bounding boxes. They present an *IOU tracking algorithm* based on the overlap between bounding boxes in

consecutive frames. It assumes that a unique object's bounding box in one frame will have a certain overlap with the same object's bounding box in the next frame. Hence, no image information is taken into account since all information is provided by bounding boxes determined by an object detector. This means the overall complexity of the algorithm is very low compared to other state-of-the-art trackers. It simply chooses the best match in terms of IOU overlap above a certain threshold σ_{IOU} for all bounding boxes in subsequent frames. Further, for a track to be included as valid, it must have at least one IOU score above a threshold σ_h, it must consist of at least t_{min} frames, and the detection must have a detection score of at least σ_l. Figure 3.10 visualizes the concept.

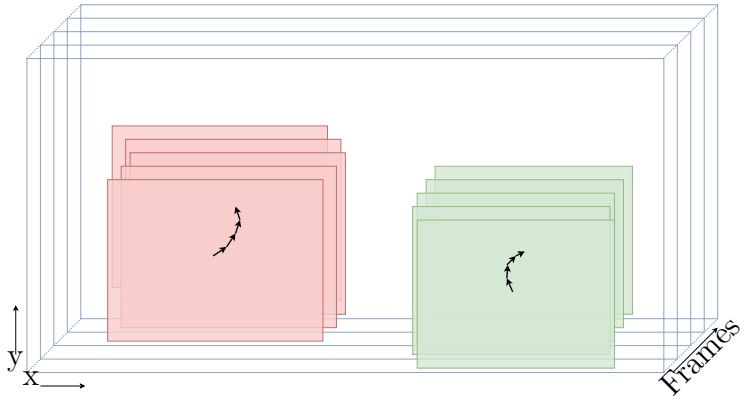


Figure 3.10: Principle of the IOU tracking algorithm. Based on Bochinski et al. (2017).

Chapter 4

Stereo Vision for 3D Reconstruction

Stereo vision – in the context of computer vision – refers to a technique where based on the 2D views from two cameras one is able to extract the 3D position of feature points or objects by triangulation. The extension to more views is also known as multiple-view geometry. This chapter will focus on the specific case of using two cameras observing the same scene. An object is detected in both cameras but at slightly different positions at the image, making it possible to extract estimated depth information when the relative position of the two cameras is known. Stereo vision is not a new technique; it has matured over the past three decades but still faces many challenges.

4.1 Single-View Geometry

To understand stereo vision, some basic knowledge about single-view geometry is needed. In principle a camera projects points from the 3D world onto a 2D plane, and this transformation can be approximated by a camera model. The contents of this section is largely based on Hartley & Zisserman (2003).

4.1.1 Pinhole Camera Model

Under the pinhole camera model, a point in space $P = (X, Y, Z)^T$ is projected onto a plane, to the point $p = (x, y)^T$, through the mapping

$$\mathbb{R}^3 \mapsto \mathbb{R}^2 : (X, Y, Z)^T \mapsto \left(f\frac{X}{Z}, f\frac{Y}{Z} \right)^T = (x, y)^T, \tag{4.1}$$

where $f = Z$ is the *image plane* or *focal plane*. Using homogeneous coordinates, (4.1) can simply be expressed as a linear mapping

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \text{diag}(f, f, 1) \left[\mathbf{I}^{3\times 3} \ \mathbf{0}^{3\times 1}\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{4.2}$$

where $\text{diag}(f, f, 1)$ is a diagonal matrix, $\mathbf{I}^{3\times 3}$ is the identity matrix and $\mathbf{0}^{3\times 1}$ is the zero vector. We define $\mathbf{X}$ as the world point represented by the homogeneous vector $[X, Y, Z, 1]^T$, $\mathbf{x}$ as the image point, and $\mathbf{P} = \text{diag}(f, f, 1) \left[\mathbf{I}^{3\times 3} \ \mathbf{0}^{3\times 1}\right]$ as the homogeneous *camera projection matrix*. This means we can write (4.2) as

$$\mathbf{x} = \mathbf{P}\mathbf{X}. \tag{4.3}$$

4.1.1.1 Principal Point Offset

In (4.3) we assume that the principal point (i.e., the point where the optical axis of the camera meets the image-sensor plane) is the origin of the image plane. This is usually not the case as the optical axis of the camera typically goes through the centre of the image. Hence, we have a mapping

$$(X, Y, Z) \mapsto (f\frac{X}{Z} + p_x, f\frac{Y}{Z} + p_y)^T \tag{4.4}$$

between world and image plane coordinates, where $P = (p_x, p_y)^T$ is the principal point. In homogeneous coordinates, one can write this expression as

$$\begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \mathbf{K} \left[\mathbf{I}^{3\times 3} \ \mathbf{0}^{3\times 1}\right] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{4.5}$$

where $\mathbf{K}$ is the camera calibration matrix,

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.6}$$

We need to change coordinates from meters to pixels by using the pixel density of the imaging sensor. We start by defining the number of pixels per unit of distance in image coordinates along the horizontal and vertical direction, respectively, as

$$m_x = \frac{n_x}{s_x}, \ m_y = \frac{n_y}{s_y}, \tag{4.7}$$

where n_x, n_y, s_x, and s_y are the number of pixels in the imaging sensor and the physical size of the sensor in meters, respectively. When (4.6) is multiplied by an extra factor $diag(m_x, m_y, 1)$ we obtain

$$\mathbf{K} = \begin{bmatrix} \alpha_x & 0 & c_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.8}$$

where $\alpha_x = fm_x$ and $\alpha_y = fm_y$ represent the camera's focal length in terms of pixel dimensions in x and y directions, respectively. x_0 and y_0 represent the principal point in pixel units. Pixels can be non-square, which has the effect of introducing unequal scale factors in each direction, meaning m_x and m_y can be different. However, for today's cameras $m_x = m_y$ (squared pixels) is a reasonable assumption. The parameters contained in $\mathbf{K}$ are the *internal*, or *intrinsic*, camera parameters. More details can be found in Hartley & Zisserman (2003).

4.1.1.2 Camera Rotation and Translation

Generally speaking, points in the real world will be expressed in terms of the *world coordinate frame*, another Euclidian coordinate frame. We can relate the two coordinate frames with a rotation $\mathbf{R}$ and a translation $\mathbf{t}$, as seen in Figure 4.1. Let $\mathbf{X}^w$ be an inhomogeneous 3-vector representing a point in the world coordinate frame, and let $\mathbf{X}^c$ represent the same point in the camera coordinate frame. Then we can, for the transformation from world to camera frame, write

$$\mathbf{X}^c = \mathbf{R}^c_w \mathbf{X}^w - \mathbf{t}^c_w, \tag{4.9}$$

where $\mathbf{R}^c_w$ is a 3×3 rotation matrix and $\mathbf{t}^c_w$ is a 3×1 translation vector relating the two coordinate frames. Sub- and superscripts c and w represents the camera and the world coordinate frames, respectively. In homogeneous coordinates, the same transformation can be written

$$\mathbf{X}^c_w = \mathbf{T}^c_w \mathbf{X}^w, \tag{4.10}$$

where

$$\mathbf{T}^c_w = \begin{bmatrix} \mathbf{R}^c_w & \mathbf{t}^c_w \\ \mathbf{0}^{1\times 3} & 1 \end{bmatrix}. \tag{4.11}$$

If (4.5) and (4.9) are combined, we obtain the *camera matrix* P, written as

$$\mathbf{P} = \mathbf{K} \begin{bmatrix} \mathbf{R} & \mathbf{t} \end{bmatrix}. \tag{4.12}$$

Hence, $\mathbf{P}$ relates a point expressed in the world coordinate frame with the image coordinate frame (pixel coordinates). While the parameters of $\mathbf{K}$ are called *intrinsic* parameters, the parameters of $\mathbf{R}$ and $\mathbf{t}$ are called *extrinsic* parameters. The intrinsic parameters

are usually determined during a single-view calibration. The extrinsic coordinates represent the relative position of the cameras in a stereo setup and are determined by stereo calibration.

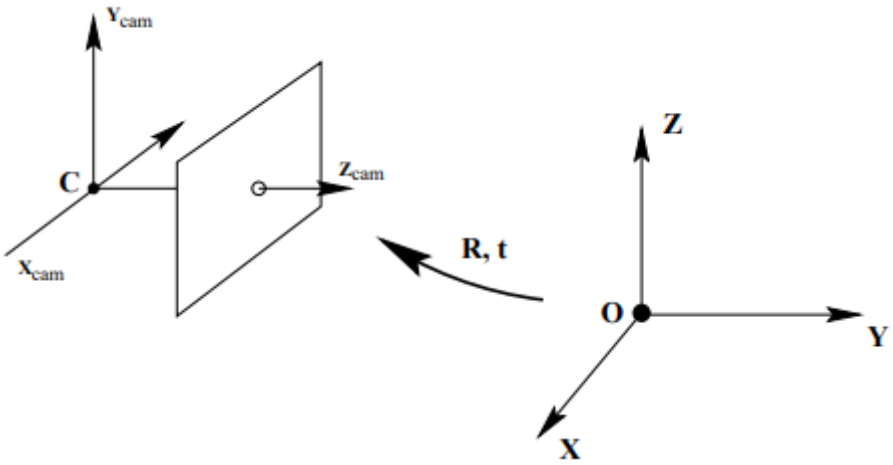


Figure 4.1: The Euclidian transformation between the world and camera coordinate frames. Image courtesy by Hartley & Zisserman (2003).

4.1.2 Distortion Model

The ideal pinhole camera model assumes a linear mapping between the image plane and the world coordinate system. For cameras with non-pinhole lenses (i.e., real lenses), this assumption does not hold. For wide-angle lenses, radial distortion can have a significant impact. I.e., if a pin-hole model is used, the error will become more prominent as the focal length decreases. When light rays bend more near the edges of a lens compare to its optical center, radial distortion appears. Figure 4.2 shows how a checkerboard pattern affected by positive and negative radial distortion, respectively.

Figure 4.2: Checkerboard pattern, appearing with no distortion, positive radial distortion, and negative radial distortion, respectively. Image courtesy by Ozcakir (2020).

A lens can also produce tangential distortion. This happens when the lens and the image plane are not parallel. To obtain a linear relationship between the image and the real world, nonlinearities introduced by this need to be removed. Considering the distorted image coordinates $[x_d, y_d]^T$ in relation to the pin-hole image coordinates $[x, y]^T$, the distortion can be approximated as

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} x(1 + k_1 r^2 + k_2 r^4 + ...) + (2p_1 xy + p_2(r^2 + 2x^2)) \\ y(1 + k_1 r^2 + k_2 r^4 + ...) + (2p_2 xy + p_1(r^2 + 2y^2)) \end{bmatrix}, \tag{4.13}$$

where the first terms express the radial distortion and the last terms express the tangential distortion. The term $(1 + k_1 r^2 + k_2 r^4 + ...)$ is a distortion factor affecting the distorted image coordinates $[x_d, y_d]^T$. The factors k_1, k_2, ... are radial distortion coefficients, and p_1 and p_2 are tangential distortion coefficients. From this, the distorted pixel coordinates can be obtained by multiplication of the distorted image coordinates with the camera calibration matrix $\mathbf{K}$, as

$$\begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}. \tag{4.14}$$

This is the forward projection from scene to image plane. We are facing the inverse problem in many cases, i.e., recovering 3D points given their corresponding image coordinates. However, there is no analytic solution to this problem if distortion is taken into consideration. Computer vision software as OpenCV does, however, provide functionality for undistortion if the distortion parameters are known.

4.2 Stereo-View Geometry

Stereo-view geometry is primarily inspired by how human eyes perceive the world. A stereo-vision setup is similar to the structure of the human eyes, namely two cameras places next to each other, sensing the same scene but at slightly different angles.

4.2.1 Epipolar Geometry

Epipolar geometry is the natural projective geometry between two views. It is only dependent on the cameras' internal parameters and pose and is thus independent of the scene structure (Hartley & Zisserman 2003).

4.2.1.1 Epipolar Line

Assume a 3D point $\mathbf{X}$ that is imaged in two views, at $\mathbf{x}$ in the first and $\mathbf{x}'$ in the second. We want to find the relation between $\mathbf{x}$ and $\mathbf{x}'$. Let these two image points, the space point $\mathbf{X}$, and the two camera centers ($\mathbf{C}$ and $\mathbf{C}'$) be co-planar, and denote this plane as π. The back-projected rays from $\mathbf{x}$ and $\mathbf{x}'$ will intersect at $\mathbf{X}$, and they will lie in π. Now, assume we only know $\mathbf{x}$, and we want to find $\mathbf{x}'$. We know the ray is defined by $\mathbf{x}$, that the camera baseline determines π, and that the ray corresponding to $\mathbf{x}'$ lies in π. Thus, $\mathbf{x}'$ must lie on the line of intersection $\mathbf{l}'$ of π with the second image plane. The line $\mathbf{l}'$ is the image in the second view of the ray back-projected from $\mathbf{x}$, also called the *epipolar line* corresponding to $\mathbf{x}$. We will see that this observation is highly beneficial in terms of the stereo correspondence problem as we can restrict the search for a point corresponding to $\mathbf{x}$ to the line $\mathbf{l}'$ instead of the whole image (Hartley & Zisserman 2003).

4.2.1.2 The Fundamental Matrix

The fundamental matrix $\mathbf{F}$ summarizes epipolar geometry in an algebraic matter, derived from the mapping between a point and its epipolar line. The previous findings suggest there is a map

$$\mathbf{x} \mapsto \mathbf{l}' \tag{4.15}$$

from a point in one image to its corresponding epipolar line in the other image. This is a projective mapping, represented by the fundamental matrix $\mathbf{F}$.

$\mathbf{F}$ can be found by geometric derivation, following two steps. In the first step, the point $\mathbf{x}$ in the first image is mapped to an arbitrary point $\mathbf{x}'$ in the other image lying on $\mathbf{l}'$. This point is a potential match for $\mathbf{x}$ and is found by a transfer via the plane π, a plane in space not passing through either of the camera centers. The ray through the first camera center corresponding to $\mathbf{x}$ meets π in the point $\mathbf{X}$, which is then projected to a point $\mathbf{x}'$ in the second image. The projected point $\mathbf{x}'$ must lie on the epipolar line $\mathbf{l}'$ since $\mathbf{X}$ lies on the ray corresponding to $\mathbf{x}$. Thus, the points $\mathbf{x}$ and $\mathbf{x}'$ are both images of $\mathbf{X}$ lying on a plane. This means the set of all such points $\mathbf{x}_i$ in the first image and the corresponding points $\mathbf{x}'_i$ in the second image are projectively equivalent. Hence, there is a 2D homography $\mathbf{H}_\pi$ mapping every $\mathbf{x}_i$ to $\mathbf{x}'_i$. In the second step, $\mathbf{l}'$ is obtained as the line joining $\mathbf{x}'$ to the epipole $\mathbf{e}'$. The epipolar line can be written as

$$\mathbf{l}' = \mathbf{e}' \times \mathbf{x}' = [\mathbf{e}']_\times \mathbf{x}' = [\mathbf{e}']_\times \mathbf{H}_\pi \mathbf{x} = \mathbf{F}\mathbf{x}, \tag{4.16}$$

since $\mathbf{x}' = \mathbf{H}_\pi \mathbf{x}$ and when we define $\mathbf{F} = [\mathbf{e}']_\times \mathbf{H}_\pi$.

4.2.2 Image Rectification

The search for corresponding points is now limited to a search along the epipolar line. However, the search can be simplified further. By performing image rectification, the epipolar lines with the same Y component in both images are in one-to-one correspondence. This is achieved by simulating rotations of the cameras to generate two co-planar image planes that are parallel to the baseline. Algebraically this means applying 2D projective transformations (homographies) in both images. The goal for the rectified images is to achieve a common image plane and parallel epipolar lines for the two images, meaning epipoles go towards infinity.

To reproject the image planes onto a common parallel plane, the first step is to map the epipole to infinity by finding a projective transformation $\mathbf{H}$. Precisely, the epipole must be mapped to the particular infinite point $(1, 0, 0)^T$.

First, recall the camera matrix decomposed as $\mathbf{P} = \mathbf{K}[\mathbf{R} \;\; \mathbf{t}]$ as defined in Equation 4.12, and let $\mathbf{x} = \mathbf{PX}$ be a point in the image. Assuming the calibration matrix $\mathbf{K}$ is known, we can apply its inverse to $\mathbf{x}$ to obtain the point

$$\hat{\mathbf{x}} = \mathbf{K}^{-1}\mathbf{x}. \tag{4.17}$$

When substituting Equation 4.17 into Equation 4.12, we can write

$$\hat{\mathbf{x}} = [\mathbf{R} \;\; \mathbf{t}]\,\mathbf{X}, \tag{4.18}$$

where $\hat{\mathbf{x}}$ is the image point expressed in *normalized coordinates*, and $\mathbf{P}_n = \mathbf{K}^{-1}\mathbf{P} = [\mathbf{R} \;\; \mathbf{t}]$ is a *normalized camera matrix*, i.e., the effect of the known calibration matrix $\mathbf{K}$ has been removed.

Further, we consider a pair of normalized camera matrices $\mathbf{P} = [\mathbf{I} \;\; \mathbf{0}]$ and $\mathbf{P}' = [\mathbf{R} \;\; \mathbf{t}]$, i.e., camera matrices of a stereo system with the world origin positioned at the first camera. The fundamental matrix corresponding to the normalized camera pair is called the *essential matrix*, $\mathbf{E}$, and has the form

$$\mathbf{E} = [\mathbf{t}]_\times \mathbf{R} = \mathbf{R}[\mathbf{R}^T\mathbf{t}]_\times = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_x \\ 0 & t_x & 0 \end{bmatrix}. \tag{4.19}$$

I.e., the essential matrix maps corresponding points in rectified stereo images by a horizontal shift t_x and the image planes are transformed to a parallel plane. This means the search for corresponding points can be simplified to a search along the horizontal axis as illustrated in Figure 4.3. For more details on the derivations, see Hartley & Zisserman (2003).

4.3 Stereo Matching

Stereo matching is the process of finding correspondences between two images of the same scene. The distance between matching points in the two images is used to find the disparity map, which can be used to compute the 3D position of points in the scene.

4.3.1 The Correspondence Problem

The correspondence problem involves finding which pixel in the first image corresponds to which pixel in the second image. Searching for matching pixels through the whole image would be highly computationally expensive; thus, it is crucial to decrease the search area to reduce the load. If the images are fully rectified, the corresponding pixel will have the same Y position in both images; hence, one can limit the search along the X-axis.

Since the two cameras in the stereo setup have different fields of view, and due to potential occlusion, some points in one image will have no corresponding points in the other image. I.e., the stereo system must also determine what parts of the image should not be matched.

There is a number of algorithms to determine disparities, and Table 4.1 briefly explains a selection of approaches. We can categorize the methods in to *local* and *global* methods. Local methods exploit constraints on a small number of pixels surrounding a pixel of interest. They are statistical methods and prioritize matching cost computation and cost aggregation, usually by exploiting a winner-takes-it-all strategy by choosing the disparity with the least cost for all pixels, rather than using an energy function as in the global methods. These methods can be very efficient, but also sensitive to ambiguous regions like occlusion regions or regions without any distinct texture. Global methods exploit constraints on scanlines or on the entire image, and optimize an energy function created from these regions. They have a higher computational complexity, but are less sensitive to locally ambiguous regions since they provide additional support for regions that are difficult to match locally (Liu & Aggarwal 2005).

A disparity map can be dense or sparse. Dense matching methods aim to reconstruct every pixel in the image, while sparse methods only reconstruct pixels for selected feature points in the image (Ahmed et al. 2020).

Table 4.1: Stereo matching approaches, as described by Brown et al. (2003).

Approach	**Brief description**
Local methods	
Block matching	Search for maximum match score or minimum error over small region, typically using variants of cross-correlation or robust rank metrics.
Gradient-based methods	Minimize a functional, typically the sum of squared differences, over a small region.
Feature matching	Match dependable features rather than the intensities themselves.
Global methods	
Dynamic programming	Determine the disparity surface for a scanline as the best path between two sequences of ordered primitives. Typically, order is defined by the epipolar ordering constraint.
Intrinsic curves	Map epipolar scanlines to intrinsic curve space to convert the search problem to a nearest-neighbors lookup problem. Ambiguities are resolved using dynamic programming.
Graph cuts	Determine the disparity surface as the minimum cut of the maximum flow in a graph.
Nonlinear diffusion	Aggregate support by applying a local diffusion process.
Belief propagation	Solve for disparities via message passing in a belief network.
Correspondenceless methods	Deform a model of the scene based on an objective function.

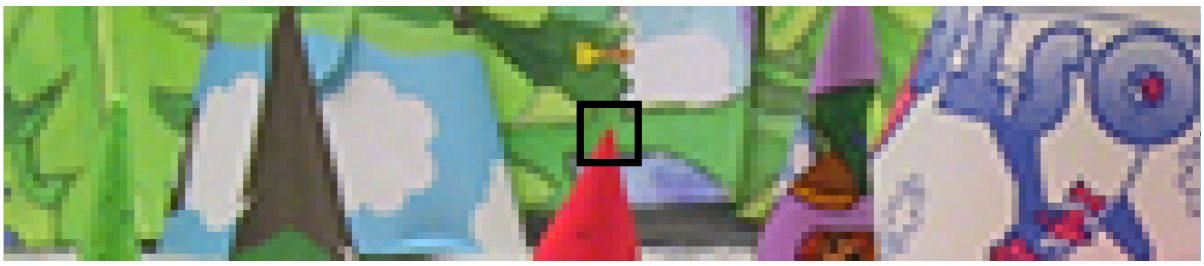
As the thesis focuses on methods suited for practical purposes, efficiency is prioritized and the emphasis will be on local methods. Of the local methods presented here, block matching and semi-global block matching provide dense disparity maps, as they use and compare image windows. Feature matching only considers a sparse set of image features, giving sparse density maps.

4.3.1.1 Block Matching

Block-matching algorithms look for matching locations in the two images. This class of methods is one of the earliest and simplest approaches for stereo matching. They match small image patches, or blocks, along the same horizontal axis in the rectified images.

A search strategy is employed to find the location on the same horizontal axis with the highest similarity between image patches. The matching cost aggregation is commonly accomplished by averaging or summing the matching cost in the blocks. The first block checked for a match is the block in the second image with the same center coordinate as the block in the reference image. Then the block is moved to the left in the second

image, until it hits the block most similar to the reference block. Figure 4.3 and Figure 4.4 illustrate the process.



(a) Reference block (black box) in the left image.

(b) Initial search position (white box) and true match (green box) in the right image. The yellow box shows the horizontal search line.

Figure 4.3: Illustration of block matching process. Image courtesy by McCormick (2014).

The block-matching methods differ mainly in the cost function used to measure similarity. An example of sum of absolute differences (SAD) cost computation between two possibly matching blocks is shown in Figure 4.4. The definitions of SAD and some other common methods are shown in Table 4.2.

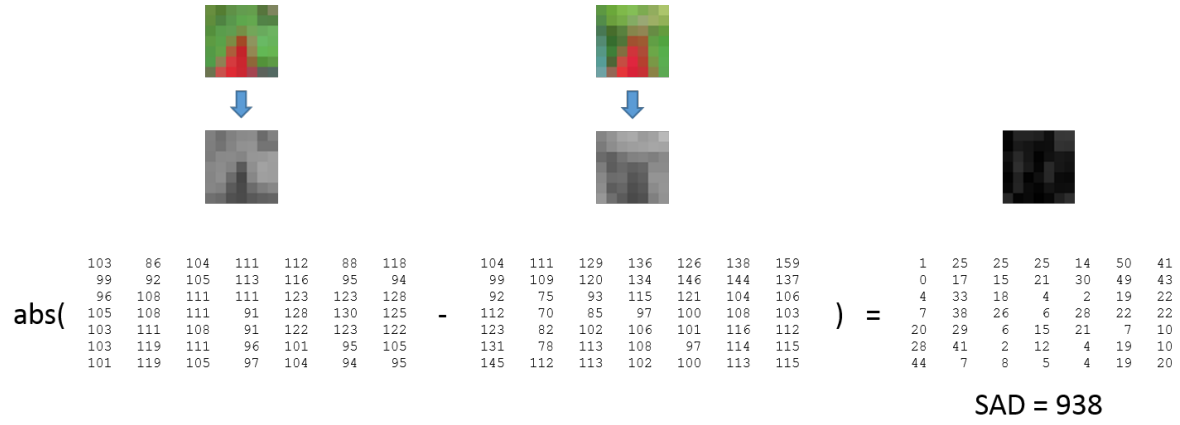


Figure 4.4: SAD cost computation between two possibly matching blocks. Image courtesy by McCormick (2014).

Table 4.2: Common cost functions for measuring similarity in correlation-based stereo matching methods. Based on Praveen (2019) and Brown et al. (2003).

Method	**Cost function**
Sum of squared difference (SSD)	$C_{SSD}(d) = \sum_{(u,v)\in W_m(x,y)} [I_L(u,v) - I_R(u-d,v)]^2$
Sum of absolute difference (SAD)	$C_{SAD}(d) = \sum_{(u,v)\in W_m(x,y)} \lvert I_L(u,v) - I_R(u-d,v)\rvert$
Normalized cross-correlation (NCC)	$\hat{I} = \frac{I(x,y)-\bar{I}}{\lvert\lvert I-\bar{I}\rvert\rvert_{W_m(x,y)}}$
	$C_{NCC}(d) = \sum_{(u,v)\in W_m(x,y)} \hat{I}_L(u,v)\hat{I}_R(u-d,v)$
Rank transform	$Rank(x,y) = \sum_{(i,j)(x,y)} L(i,j),\ L(i,j) = \begin{cases} 0: & I(i,j) < I(x,y) \\ 1: & \text{otherwise} \end{cases}$
	$C_{RT}(d) = \sum_{(u,v)\in W_m(x,y)} \lvert Rank_L(u,v) - Rank_R(u-d,v)\rvert$
Census transform	$Census(x,y) = Bitstring_{(i,j)\in W_m}(I(i,j) \leq I(x,y))$
	$C_{CT}(d) = \sum_{(u,v)\in W_m(x,y)} Census_L(u,v) - Census_R(u-d,v)$

Nomenclature for the equations in Table 4.2 can be found in Table 4.3.

Table 4.3: Nomenclature for cost function formulas. Based on Praveen (2019).

Symbol	**Explanation**
I_L	Left image or first camera image
I_R	Right image or second camera image
W_m	Matching window
d	Pixel disparity
$I(u,v)$	Image pixel intensity at location

As seen in the cost functions in Table 4.2, block matching requires that the scene points have the same intensity in each image. This requirement can only be strictly accurate if surfaces are perfectly matte. This is one of the dominant factors affecting the similarity measure (Praveen 2019):

- Photometric constraints (Lambertian/non-Lambertian surfaces): Lambertian surfaces look the same to the observer regardless of the angle of view. An example

of this is a perfectly "matte" surface. If a surface is not Lambertian, it might not appear similar regarding illuminance and brightness in the two images, which makes matching difficult.

- Noise in the images: Low-quality sensors or magnified unwanted light sue to high ISO settings can cause noise. For the special case of underwater images there are additional affects that can make images noisy (see section 1.1.2). The noise can be different for the two cameras, affecting the goodness of matching.
- Pixels containing multiple surfaces: This problem will mainly occur for an objects that lie too far away in the scene. The baseline is directly proportional to the distance of objects, i.e., stereo systems with small baseline will encounter this problem even at average distances. For systems with larger baseline the issue will occur at a greater distance. Thus, it is important to choose the baseline that fits the use case.
- Occluded pixels: There will be pixels in a scene that is visible in one image but not in the other due to occlusion. As there will be no match for these pixels, it is not possible to find the disparity for them. They can only be estimated by, e.g., interpolation techniques.
- The surface texture of the 3D object: Blank walls, roads, or skies have no useful texture, making it impossible to compute their disparity by block-matching algorithms. Computation of disparity for these points requires global methods that consider the information in the entire image.
- The uniqueness of the object in the scene: The algorithm can match incorrect pixels if objects in the scene are not unique. A larger block could help, but this is also more computationally expensive.
- Synchronized image capture from the two cameras: The images must be captured at the exact same time. This is especially important if there is movement in the scenes. This can be achieved by hardware trigger, or at software level if there is continuous recording. The former gives perfect synchronization, while the latter could be less accurate.

4.3.1.2 Semi-Global Block Matching

Semi-global block matching (SGBM) was introduced by Hirschmuller (2008). Simply explained, the SGBM algorithm differs from the simple block matching algorithm by enforcing smoothness constraints between neighboring pixels so that they take similar disparity values, in addition to calculating local optimal disparity value for each pixel. I.e., SGBM introduces global constraints to the local block matching method. This will in general lead to better results, but comes with an additional computational expence. The SGBM algorithm is explained by a block diagram in Figure 4.5.

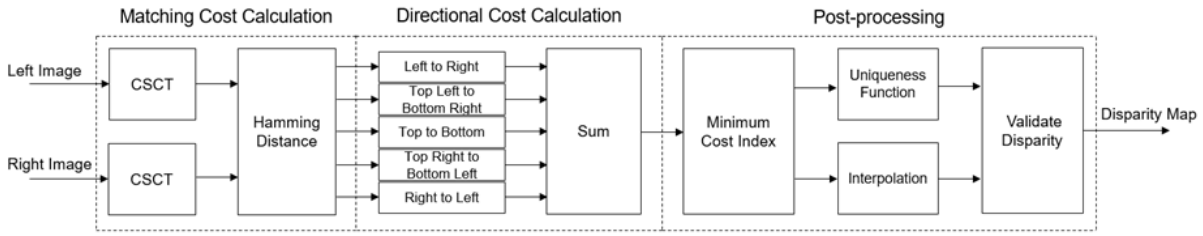


Figure 4.5: SGBM block diagram, using five directions. Image courtesy by The MathWorks, Inc. (n.d.).

4.3.1.3 Feature Matching

Feature-based methods involve examining features in the image and checking if the layout of a subset of features is similar in the two images. Image features can be, among others, corners, edges, circles, patches, curve segments, and line segments. Feature detection algorithms search for features in each images that can be matched with features in other images. The region around a feature is described compactly in the descriptor. Then, the features in the reference image are matched with features in the second image by using the feature descriptors. By cost-matching using brute force or nearest neighbor, corresponding feature pairs are extracted from the stereo pair.

Technological developments are advancing from block-matching methods, and feature-matching is a technique that is accelerating in stereo vision. Feature-matching can be accomplished in real time, but these methods are not able to detect small changes in the stereo images (Hämäläinen et al. 2005). Feature-matching methods are sensitive to the aperture problem, and to avoid this, one should require that a good feature has local variation in at least two directions.

4.3.2 Disparity Post-Processing

Some of the effects affecting the similarity measure when calculating the disparity maps can be handled by post-processing the disparity maps. Some examples of post-processing are, according to Praveen (2019):

- Removal of spurious stereo matches: A simple way to encounter this problem is to use the median filter, but it might fail if there are larger spurious speckles present. Speckle filtering can be achieved using other approaches, e.g., removal of tiny blobs that are inconsistent with the background. It removes most incorrect disparity values, but also leaves holes or blank values.
- Filling of holes in the disparity map: Holes are mainly caused by occlusion or

removal of false disparities. Left-right disparity consistency checks can be used to detect occlusion.

- Sub-pixel estimation: Most stereo matching algorithms output integer disparity values, meaning disparity maps become discontinuous and information is lost. Gradient descent and curve fitting are common methods to solve this problem.

4.4 The Reconstruction Problem

Given that the correspondence problem is solved and that we have pair of matching points (x, x') in the left and right image, respectively, it remains to reconstruct the corresponding scene point. This point, $X = (x_L, y_L, z_L)$ (relative to the left camera origin C), has pixel displacements u and u' along the horizontal image axis. We need to find the depth, z_L, by geometric derivation. While u and u' are the horizontal components of the image points, v and v' similarly represent the vertical component of the image points. Usually, these components are defined relative to the upper left corner of the image. The focal length f is the same for both cameras. 3D reconstruction from disparity maps given the camera parameters is called *triangulation*, as we use similar triangles to derive the following relationship:

$$u = f\left(\frac{x_L}{z_L}\right). \tag{4.20}$$

Similarly for $X = (x_R, y_R, z_R)$ relative to the right camera, we have

$$u' = f\left(\frac{x_R}{z_L}\right). \tag{4.21}$$

The length b relates the two camera centers, meaning $x_R = x_L + b$. The disparity is a pixel shift along the horizontal axis and can be defined as $d = u' - u$. This, together with (4.20) and (4.21), means

$$d = u' - u = f\left(\frac{x_R}{z_L}\right) - f\left(\frac{x_L}{z_L}\right) = f\left(\frac{x_L + b - x_L}{z_L}\right) = f\left(\frac{b}{z_L}\right). \tag{4.22}$$

From (4.22), z_L can be obtained by rearrangement, as

$$z_L = f\left(\frac{b}{d}\right). \tag{4.23}$$

A similar procedure can be done to obtain x_L and y_L, and the final 3D reconstruction is thus

$$\mathbf{X} = (x_L, y_L, z_L) = \left(u\frac{z_L}{f}, v\frac{z_L}{f}, f\frac{b}{d}\right). \tag{4.24}$$

(4.24) requires the baseline b, the disparity d, and the focal length f to be known to be able to obtain z_L. The image point $x = (u, v)$ is also assumed to be known such that x_L and y_L can be obtained from z_L.

A disparity-to-depth mapping matrix Q can be derived from these equations to reduce the 3D reconstruction calculations to

$$\begin{bmatrix} x_L \\ y_L \\ z_L \\ w_L \end{bmatrix} = Q \cdot \begin{bmatrix} u \\ v \\ d \\ 1 \end{bmatrix}. \tag{4.25}$$

The matrix Q is defined as

$$\begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{b} & \frac{p_x - p'_x}{b} \end{bmatrix}, \tag{4.26}$$

where p_x and p_y are the x and y components of the left principal point, respectively, and p'_x is the x component of the right principal point. Now, inserting this into Equation 4.25 gives

$$\begin{bmatrix} 1 & 0 & 0 & -p_x \\ 0 & 1 & 0 & -p_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -\frac{1}{b} & \frac{p_x - p'_x}{b} \end{bmatrix} \begin{bmatrix} u \\ v \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} u - p_x \\ v - p_y \\ f \\ \frac{p_x - p'_x - d}{b} \end{bmatrix} = \begin{bmatrix} x_L \\ y_L \\ z_L \\ w_L \end{bmatrix}. \tag{4.27}$$

When the left and right principal points are equal, this gives

$$z_L = \frac{-bf}{-d - (p_x - p'_x)} \Rightarrow \frac{z_L}{w_L} = \frac{fb}{d}, \tag{4.28}$$

which is the same as in Equation 4.23 using homogeneous coordinates.

Chapter 5

Implementation

The purpose of the following chapter is to present and substantiate the chosen methodology for the project. First, we provide a description of the dataset, including several aspects regarding the collection, annotation, and processing of the data. Then we will explain the selection of the detection model. Next, we will look further into the selection and modification of the tracking algorithm. The following section will present the prerequisites for the implementation of the methods. Next, the network configuration and the training of the object detector are described. The last section will explain how the methods are evaluated.

5.1 Data Acquisition

This section will present the collection and processing of the dataset, including the process of acquiring the dataset, and some details about the camera and the images. Further, we will present the pre-processing of the data.

5.1.1 Data Collection

Image data of the quality and quantity that was desirable for training neural networks was hard to collect. NTNU possesses a fish farm equipped with a variety of sensors for research means in Ålesund, Norway. It was possible to go there to collect camera data, but with the Covid-19 pandemic and limited time, it was desirable to get hands on an existing dataset. Also, after a meeting with doctoral fellows at NTNU Ålesund it was clear that the cameras and equipment available might not be suitable for acquiring high-quality video, and the environment might not be satisfactory considering algae blooms and weather conditions.

Table 5.1: Camera information, taken directly from the information file provided from SINTEF with the LAKSIT dataset. Two of these cameras were used for the stereo setup.

Camera model	Blackfly BLFY-PGE-13E4C
Camera vendor	Point Grey Research
Sensor	E2v EV76C560 (1/1.8" Color CMOS)
Resolution	1280×1024

SINTEF has a variety of image and video data which was made available for this project. One of these was the LAKSIT dataset, which was chosen to be examined further since it contained a large quantity of high-quality stereo video. The data was in the form of separate stereo video streams recorded for short periods of time with a frame rate of 24 FPS, ranging from 2000 frames (about 1.5 minutes) to 3120 frames (about 2 minutes). The video streams were recorded from within the same cage at an approximate depth of 5 meters and the camera was fixed by a rope. The videos were taken in the time between the 25th of October 2016 and the 21st of February 2017 including periods where the camera was taken out of the cage due to a few fish farming operations. Camera information for the two cameras is shown in Table 5.1.

5.1.2 Preparations for Object Detection

The used subset of recordings contained 17 video streams, in total 33 minutes and 42 seconds. Looking at previous studies and keeping in mind the fact that the dataset was limited, a decision was made to set some requirements for the training and validation sets for the object detection model.

1. the number of training image frames should be at least 80,
2. the datasets should contain image frames from all provided video data sources, and
3. all objects of interest in the chosen image frames should be possible to distinguish from other objects with a human eye.

Requirement (1) and (2) were to guarantee a certain variability in the datasets. Requirement (3) was important because of limited visibility in the image frames. A dangerous pitfall when choosing training data is that objects of interest actually are present in frames of the dataset, but not labeled. This can e.g. happen when objects are blurry or too similar to other types of objects. However, it is important to make sure the network can detect objects in the various data sources it is supposed to be applied to, hence, the training and validation sets must reflect this. To manifest these requirements and to keep a certain amount of randomness, a custom algorithm was used to pick a number of random frames from the video streams, using the open-source software FFmpeg and OpenCV, as follows:

1. Concatenate all video streams to *one* stream.
2. Count the total number of frames in the stream.
3. Pick a sample of n unique random indices between zero and the total number of frames.
4. Extract video frames corresponding to these indices.

The most suitable frames for training were chosen manually. The videos contained some motion blur, and since they were recorded in dynamic environments some of the random frames extracted did not contain any fish or was not suitable because of conditions like too much motion blur, inadequate lightning, or turbidity.

The resulting dataset contained 260 images and was further split randomly, 80% for training, 10% for validation, and 10% for testing. Next, all images were labeled, and images in the original training set were augmented (see section 5.3.2.1) before the training phase was initiated.

5.1.2.1 Selection of Object of Interest

To be able to analyze the swimming velocity of fish from video sequences, each fish must be identified and tracked through a sequence of frames, and the depth of an individual fish must be estimated. Because of the problems introduced by the water as a medium (see section 1.1.2), depth estimation of underwater images is more challenging than terrestrial images.

To be able to estimate the fish velocity, the position of a fixed point on the fish must be estimated in each frame. By using object detection we can use the middle point of each bounding box as this fixed point. It remains to decide *what object* to detect and track. Detecting the entire fish could be a possible approach for tracking and estimating depth of the fish. However, based on initial tests of depth estimation algorithms, it was difficult to obtain a good and contiguous depth estimate for the entire surface of the fish. Thus, it was decided to focus on an object of interest on the fish surface, and use this object to identify the fish for tracking, and for estimation of the depth of the fish.

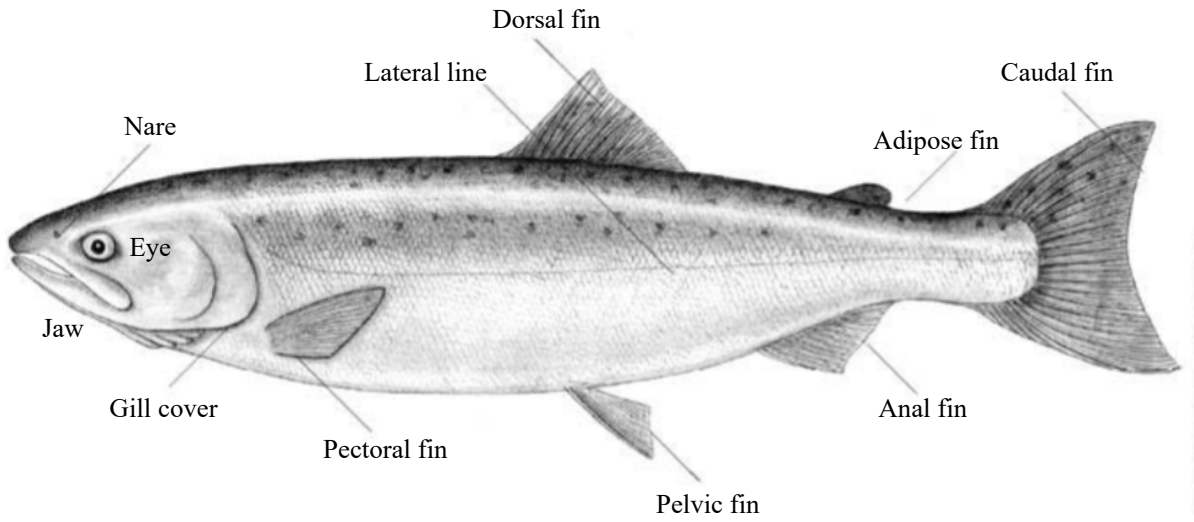


Figure 5.1: Salmon external anatomy.

Table 5.2 summarizes the author's subjective perception of some of the fish anatomic objects shown in Figure 5.1, and their appearance in underwater images form the LAKSIT dataset. Each object is evaluated by three criteria, based on observations of the image frames. The first criterion, *highly affected by motion blur*, is important because objects tend to get their edges blurred out because of motion blur in the data, making it harder for the object detection model to recognize the object and for the stereo matching algorithm to match pixels. The next criterion is *easily confused with other objects*. E.g., a salmon has a variety of fins, some of them with a similar shape. Since the detected object will identify a single fish, it is important to avoid that several objects are detected on the same fish. The last criterion is *highly affected by low contrast*. This relates to the same problems as for the first criterion. If an object is highly affected by low contrast (which is common in underwater images, see section 1.1.2), it will be hard for the object detector to distinguish the object from other objects and the background. Edges will be less prominent, and the object will diminish. The more of these criteria are evaluated to "no" for an object, the better the object is for our pipeline.

Table 5.2: Evaluation of objects of interest, using three criteria.

Anatomic object	**Highly affected by motion blur**	**Easily confused with other objects**	**Highly affected by low contrast**
Eye	Yes	No	Yes
Jaw	Yes	No	Yes
Gill cover	Yes	No	Yes
Dorsal fin	No	Yes	No
Adipose fin	No	Yes	No
Tail (caudal) fin	**No**	**No**	**No**
Anal fin	No	Yes	No
Pelvic fin	No	Yes	No
Pectoral fin	No	Yes	No

The tail fin is not as affected by motion blur as the fish eye. It has a distinct shape which is hard to confuse with other objects that could appear in image data from a sea cage. The tail fins are larger than other fins, making them easier to track in consecutive frames when using a tracking algorithm based on IOU overlap.

There are some drawbacks of using the caudal fin as the object of interest, especially related to depth estimation. The fin is moving sideways, meaning the depth of the fin will vary relative to the body of the fish from the camera perspective in the dataset (looking at the fish from one side). This was however considered as less important as the goal is an *estimation* of the depth.

5.1.2.2 Annotation

Only raw data was provided, hence, the labeling/annotation process was to be done from scratch. In this case, Microsoft's Visual Object Tagging Tool (VoTT) was used. It is an open-source annotation and labeling tool for image and video assets. It provides a GUI where one can simply draw bounding boxes around the objects and label them as desired. Despite the fact that VoTT does not support the YOLOv4 Darknet annotation format, it is user friendly and fast, and runs on Windows 10. We used the web tool RoboFlow to easily convert to the correct format and custom algorithms to list all training, validation, and test files in separate text files, like Darknet requires.

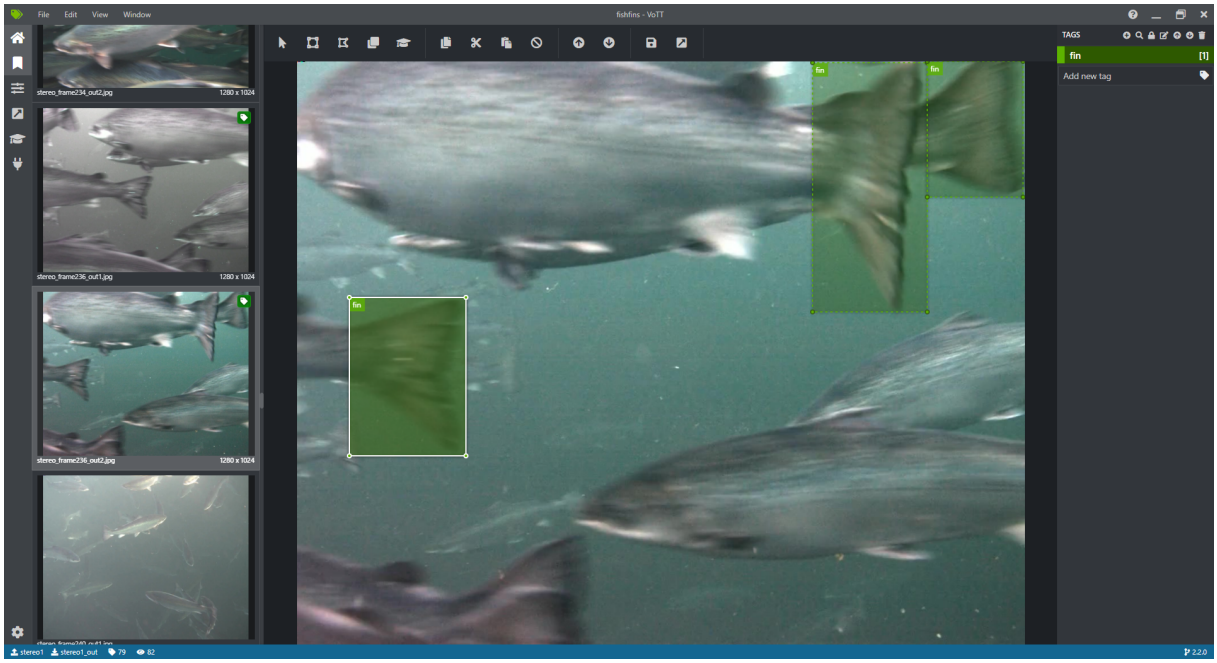


Figure 5.2: Screenshot from annotation tool (VoTT).

Figure 5.3: Example of image frame with extensive motion blur.

Figure 5.4: Example of image frame of scarce quality. The image contains a high level of noise, several tale fin occlusions, and an unclear distinction between foreground and background.

Figure 5.5: Example of an accepted image frame. The image has clear distinctions between foreground and background, an acceptable amount of motion blur, and few tale fin occlusions.

5.2 Implementation Prerequisites

In this section, different prerequisites for running the proposed detection and tracking model are described, including hardware, software, and configuration of the detection neural network.

5.2.1 Computer

The computer used in this project is an ASUS ROG Strix G15CK. For training and validating the two deep learning models, a computer with a powerful GPU is required to be able to run in a satisfactory time perspective. Further, an NVIDIA designed GPU is required to run on the parallel programming platform CUDA which was used in this project. The main specifications are listed in Table 5.3.

Table 5.3: Main specifications of the computer used in the project.

Producer	ASUS
Product series	ROG Strix G15CK
Model	NR008T
CPU	Intel Core i5 (10th generation)
GPU	NVIDIA GeForce RTX 2060 SUPER
RAM	2×8 GB
Operating system	Windows 10 Home

5.2.2 Software

This section presents the main software frameworks used in this thesis. These are related to the deep-learning, computer vision, and image processing tasks.

CUDA

To accelerate the deep learning process, it can be sufficient to take advantage of the parallel computing power of the computer's graphics processing unit (GPU). While GPU computing traditionally has been used for graphics-heavy operations such as video games, it has now become more mainstream and can greatly improve the performance of other computation-heavy operations such as deep learning. There are several platforms implementing parallel computing on GPU. One of the most popular platforms is CUDA which is developed by NVIDIA. This is the parallel computing platform used in this project, firstly because it significantly speeds up the training time, and secondly because it is required for running YOLOv4 in real-time. In this project, CUDA version 10.2 was used.

OpenCV

OpenCV is an open-source computer vision and machine learning software library. It includes both classic and state-of-the-art algorithms for computer vision and machine learning and contains more than 2500 optimized algorithms. OpenCV is commonly used by a wide range of companies, from large international companies to start-ups. OpenCV version 4.1.0 was used for this project, and it has been extensively used both in processing raw image frames and as a part of the YOLOv4 model.

Darknet

Darknet is an open-source neural network framework. According to Redmon (2013–2016), it is fast, easy to install, and supports CPU and GPU computation. The framework supports running several YOLO versions, including YOLOv4, and additional modules such as ImageNet Classification and Recurrent Neural Networks (RNN). To run Darknet in real-time, OpenCV and CUDA are required.

5.3 Object Detection Model

This section presents the process of selecting the object detection model and aspects with training the model to recognize fish tale fins.

5.3.1 Selection of Detection Model

Comparing the different detection models presented in section 3.1 is a demanding task. One reason for this is that there is no single general measure that can take all factors into consideration. Most detection models are tested on standard, large datasets with a wide range of classes, such as the MS COCO dataset (Lin et al. 2014) or the PASCAL VOC datasets (Everingham et al. 2014). While the performance on these datasets can provide a general opinion of the performance, it cannot capture particularities that may occur in specific datasets. Also, several metrics are commonly used for comparison. One detection model can perform better than another in terms of one metric but can be significantly worse using a different metric. For instance, YOLOv3 performs great on the AP metric with a 0.5 IOU threshold, but bad on the AP metric averaging over IOU thresholds between 0.05 and 0.95 (Redmon & Farhadi 2018). Another aspect is the trade-off between speed and accuracy. Different ratios of importance will favor different models. If we compare the models in Figure 5.6, EfficientDet will be a better choice if only accuracy is taken into account, while YOLOv4 is an obvious choice when time is crucial. Finally, the architecture design choices are of high importance when it comes to details such as the size of the objects to be detected, the density of objects in a typical image, and the number of classes involved. As a model optimized for multi-scale prediction will perform better on a general dataset with objects of many different sizes, it may not be superior on a custom dataset containing e.g. only small objects. For implementation considerations, most of the highlighted models in this section are open-source and off-the-shelf, meaning they can be implemented with little effort. There are however variations in how extensively they can be configured to custom purposes, and how well they are supported in terms of different frameworks and integrations. Hardware requirements as e.g. GPU support

are also aspects to consider, particularly when the model is to be applied in production systems.

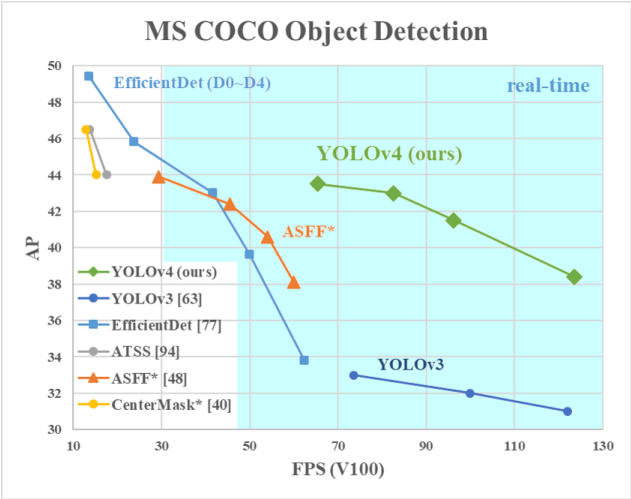


Figure 5.6: Comparison of YOLOv4 and other state-of-the-art object detectors. Image courtesy of Bochkovskiy et al. (2020).

How is the detection model chosen, then? The ideal solution would be to try out all models and all their configurations on our specific dataset, but this is obviously not feasible, especially not for a project with time and effort limitations like in our case. Figure 5.6 shows how some state-of-the-art models perform in terms of the average AP between 0.05 and 0.95 IOU metric on the COCO dataset. It shows that YOLOv4 outperforms several state-of-the-art detectors on the COCO dataset. While this comparison, as discussed, cannot provide a perfect basis for the choice for our specific problem, it is a good starting point. Further, certain priorities must be determined. In our case, speed is highly prioritized as we aim for detections in real-time. This speaks for choosing a one-stage model. Caudal fins will appear in different sizes depending on their distance from the camera, meaning the chosen model must be able to detect objects at different scales. Hence, both SSD and YOLOv4 can be considered further. Since the model is envisioned implemented for industrial use with limited financial resources, the ability to run online on a conventional GPU is highly weighted. Taking all of the mentioned aspects into consideration, YOLOv4 appeared to be the best choice for the purpose of this project.

Further, there exist several YOLO architectures. The *YOLOv4-tiny-3l* model is a modified design that is customized to run faster than the original. It is thus more convenient to use on smaller and lighter computers like e.g. edge devices in AUVs or in other sensor-carrying underwater platforms. This model was studied and compared with the original YOLOv4 model on detection of pellets in underwater images in the specialization project and the original YOLOv4 model was found to perform better. Hence, the original model was chosen for the work in this thesis.

5.3.2 Training

In this section, we will look into several aspects of the training process. First, the important phenomenon of overfitting is discussed. Further, we look at a method to achieve larger variability in the training data: data augmentation. Next, the training configuration of the detection network is presented. At last, we will explain the concept of transfer learning and how it is used in this project.

5.3.2.1 Data Augmentation

YOLOv4 already integrates four different augmentation methods that can be adjusted in the configuration file: angle, saturation, exposure, and hue. Note that *angle* (which is supposed to decide how much an image can be rotated) is not yet supported for object detection according to Bochkovskiy (2020*b*), thus the value is set to 0 as default. The other values were also kept as default:

- Saturation: 1.5 (means random value, between $\frac{1}{1.5}$ and 1.5)
- Exposure: 1.5 (means random value, between $\frac{1}{1.5}$ and 1.5)
- Hue: 0.1 (means random value, ± 0.1)

To achieve even more variability to the dataset, the images were augmented additionally before training. Rotation augmentation was added since camera setups in sea cages vary (meaning data can be recorded from many different points of view) and we wanted to train the model to be able to handle this. Since caudal fins can be present at various distances from the camera, we found it useful to add random crop augmentation such that the model could be exposed to additional object sizes. The fins can appear at any rotation, hence, rotation and horizontal and vertical flipping was added to provide a wider dataset. The original images were highly influenced by variations in hue and exposure. To prevent the model from placing a high importance on color, a gray-scale version of each image was added. RoboFlow, the web tool used for converting annotation format (see section 5.1.2.2), provides functionality to easily perform augmentation.

The final augmentations were:

- Rotation: Random value, between $-15°$ and $+15°$
- Crop: Random value, up to 60% zoom
- Flip: Horizontal and vertical (randomly)
- Gray-scale: All images

Thus, the resulting dataset before training contained four augmented versions of each image in addition to the originals, in total $5 \cdot 208 = 1040$ training images. Examples of augmented images are shown in Figure 5.7.

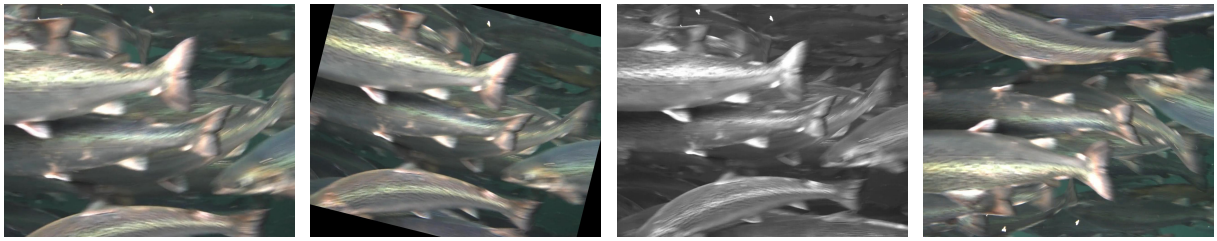


Figure 5.7: Four different augmentations of one training image.

5.3.2.2 Transfer Learning

For the object detector we use pre-trained model parameters provided by the authors behind the YOLO model, achieved by training on ImageNet, and more specifically we use the *yolov4.conv.137* and the *yolov4-tiny.conv.29* weights.

5.3.2.3 Network Configuration

Before training, the network architecture must be set up and training parameters must be tuned. This happens in the configuration file. The authors of the YOLOv4 model provide configuration files describing the general architecture of each model.

Since the chosen YOLOv4 model was the original YOLOv4 model, we used the corresponding configuration file *yolov4.cfg*. Further, several parameters could be adjusted to tune the models. We tested various configurations before choosing the final values. E.g., we tested a strategy recommended by the YOLOv4 authors, changing stride and filters in the upsampling layers in the model to better detect small objects. However, we obtained worse results with this configuration, probably because the fins also appear in large sizes (although the majority appear quite small). Table 5.4 shows the final values used.

Table 5.4: Configuration of training parameters for YOLOv4.

Configuration parameter	**Value**
Batch	64
Subdivisions	64
Width	544
Height	544
Channels	3
Momentum	0.949
Decay	0.005
Learning rate	0.001
Burn-in	1000
Max batches	10000
Policy	steps
Steps	8000, 9000
Scales	0.1, 0.1

Batch size

During training, the network iteratively uses subsets of the training dataset to compute the loss and update the weights, instead of running the entire dataset at once. The batch size determines the size of the subset of images processed in one iteration. This value was set to 64, which is the default value.

Subdivisions

The batches are again divided into mini-batches. The size of the mini-batches is the number of images the GPU will process simultaneously in parallel. Due to the high image resolution, the batches had to be divided into 64 mini-batches to prevent the GPU from running out of memory. Channels was set to 3 because the network was set up to process RGB (i.e., 3 channels) images.

Width, height, and channels

These parameters determine how the input images should be resized. Width and height were set to a moderate resolution, 544×544, to more likely be able to capture small objects and at the same time run within the capacity of the GPU.

Momentum and decay

Momentum and decay are used to control how the weights are updated. Momentum is an optimizer that can help to speed up learning when the cost gradient is not changing fast enough, and it prevents the learning to become unstable when the cost gradient is high. The momentum values was set to 0.949, the default value. The decay parameter helps prevent overfitting by introducing a penalty in the loss function when weights have

high values, and this was also set to default.

Learning rate, burn-in, max batches
As explained in section 2.1.2, the learning rate determines how fast the model should learn, i.e., the step size of the adjustment of weights and biases. A too-small learning rate will result in slow learning, while a too-large value may cause divergence in the training loss which means the model will never find the optimal weights. Throughout the learning process, the learning rate will be decreased to slow down the process and to approach an optimum. The initial values were set as default.

In many cases, the training speed increases when the learning rate is kept lower in the first iterations of the training. Burn-in decides how many iterations the learning rate should slowly increase before it reaches its final value. The value is set to 1000 iterations, which is the default value.

Max batches decide the maximum number of iterations/epochs before training is terminated. According to the instructions in the Darknet repository (Bochkovskiy 2020*a*), this value should be set to at least 3 times the number of classes, or a minimum value of 6000. The value was set to 10000 after testing with several configurations showing that the best weights were found between 5000-8000 batches in most cases.

Policy, steps, and scales
Policy decides how the learning rate should increase and decrease. This was set to be done in steps in our model. The steps parameter is used when the steps policy is chosen and decides at which steps (iterations) the learning rate should be decreased. Following the instructions in the Darknet repository, this value was chosen to be $8000, 9000$, i.e., 80% and 90% of max batches. Further, the scales decide how much the learning rate should decrease at these steps. Scales are set to $0.1, 0.1$, which means the learning rate will be multiplied by 0.1 at iterations 8000 and 9000.

5.3.2.4 Validation

Due to the problem of overfitting, it is desired to validate the performance of the detection model throughout the training process. As stated in item 5.1.2, the validation set is a randomly chosen subset of the original dataset. The model continuously validates its own performance by calculating the mAP on the validation set, providing a basis for choosing the best model when the training is terminated. To find the early stopping point, we would ideally calculate the validation loss after each epoch and choose the model with the smallest value. The model saves the weights for each 1000 epochs, but in addition it always keeps the best weights, i.e., the weights achieving the highest validation mAP. The mAP metric is widely used and adequate to make the final choice (see section 5.3.3

Table 5.5: Validation mAP after each 1000 iterations of the training process, together with the best weights result.

Epoch #	**mAP@0.50**
1000	70.93%
2000	84.75%
3000	87.27%
4000	77.59%
5000	73.59%
6000	74.51%
7000	85.87%
8000	80.33%
9000	82.98%
10000	80.15%
Best	**87.79%**

for more details on mAP). In Table 5.5, the mAP values after each 1000 iterations and the best mAP are presented. The model performs quite stable throughout the training and it seems adequate to use the weights giving the best mAP for the final model.

5.3.3 Evaluation of Detection Model

Important parameters to measure when evaluating an object detector are accuracy and speed. *Average precision* (AP) is a widely used metric for measuring the accuracy of object detection models. It is based on the *precision-recall* curve (PR) for a given set of validation data. The *precision* component is a measure of how accurate the predictions are, i.e., the percentage of correct predictions among all predictions. The *recall* component is a measure of how good the positives are considered. To explain this further, it is necessary to understand how a detection case is considered true or false in this context. We define four distinct output instances, the statistical terms *True Positive*, *True Negative*, *False Positive*, and *False Negative*, in Table 5.6.

	Predicted Positive	**Predicted Negative**
Real Positive	True Positive (TP)	False Negative (FN)
Real Negative	False Positive (FP)	True Negative (TN)

Table 5.6: Confusion matrix.

Intersection Over Union

The intersection over union (IOU) is what determines which of the four categories in Table 5.6 a prediction will constitute. The training dataset should be labeled with *ground-truth bounding boxes*, i.e., real object boundaries, on all objects of interest. Under training and validation, the predicted bounding boxes are compared to the ground-truth bounding boxes. IOU measures the overlap between these two boundaries and can be defined as

$$IOU(a, b) = \frac{A_i}{A_u}, \tag{5.1}$$

where A_i is the area of intersection, and A_u is the area of union, as illustrated visually in Figure 5.8.

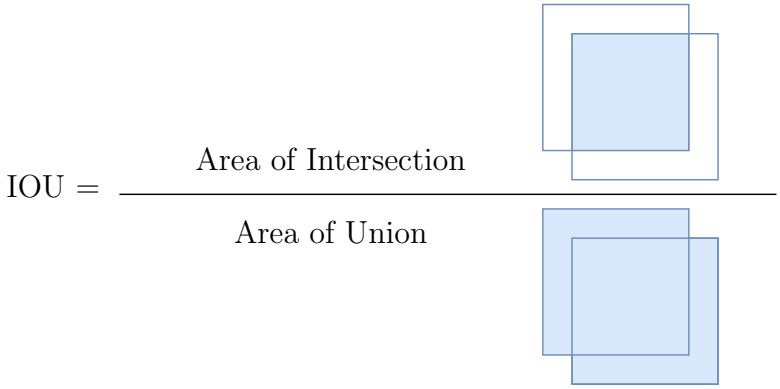


Figure 5.8: Definition of Intersection over Union (IoU).

An IOU threshold is often predefined to determine what category to classify the prediction as. That is, if a prediction gets an IOU value above the defined threshold, it is TP, while IOU below the threshold value indicates FP.

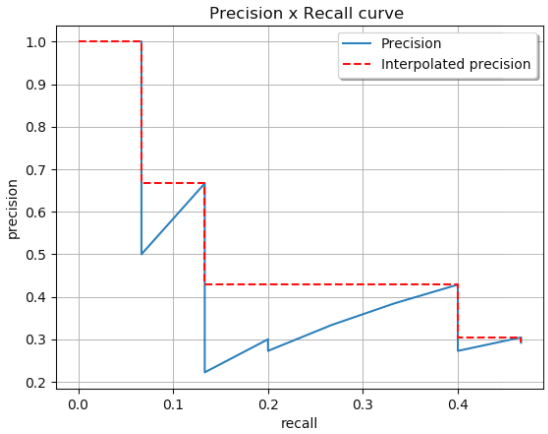
Precision and Recall

With Table 5.6 in mind, we can further define

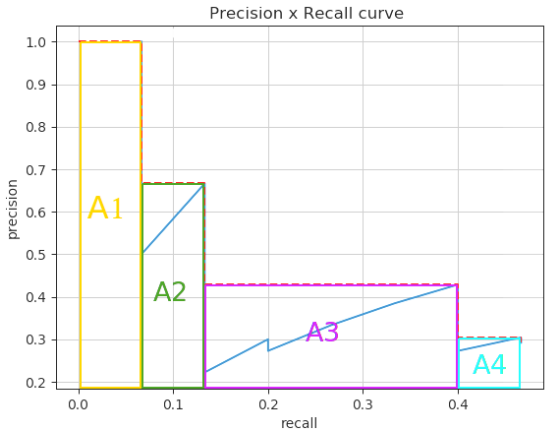
$$p = \frac{TP}{TP + FP} = \frac{TP}{\text{all detections}}, \tag{5.2}$$

$$r = \frac{TP}{TP + FN} = \frac{TP}{\text{all ground truths}}, \tag{5.3}$$

where p denotes the precision and r denotes the recall. The precision-recall curve shows the trade-off between precision and recall, and a typical example is illustrated in Figure 5.9a.



(a) Example of precision-recall curve, including the interpolated precision over all points.



(b) Interpolated precision areas.

Figure 5.9: Calculation of average precision by using area under precision-recall curve. Image courtesy of Padilla (2019).

Average Precision

With the definitions of precision and recall, we can now find the *Average Precision* (AP). The computation of AP can, according to the PASCAL VOC2012 documentation, be accomplished by the Area Under Curve (AUC) method as follows:

1. Interpolate the precision-recall curve over all points, keeping the maximum precision value at each level. The result of the example in Figure 5.9a is the red line.
2. Calculate the AP as the area under the interpolated precision curve. The obtained areas for the example are shown in Figure 5.9b.

For the provided example in Figure 5.9a and Figure 5.9b, the average precision is

$$AP = A1 + A2 + A3 + A4. \tag{5.4}$$

A high area under the curve represents both high precision and high recall, hence, we aim for a high AP. It should be noted that the exact computation of AP can be done differently for different datasets. Moreover, for a multi-class problem, the mean average precision (mAP) is commonly used. That is, we compute the AP for each class and then compute the mean value. For the sake of consistency, we will use the term mAP for both single-class and multi-class cases in this report. Finally, it should be mentioned that the notation mAP@0.50 is short for mAP calculated with an IOU threshold of 0.50.

5.3.4 Object Detection Pipeline

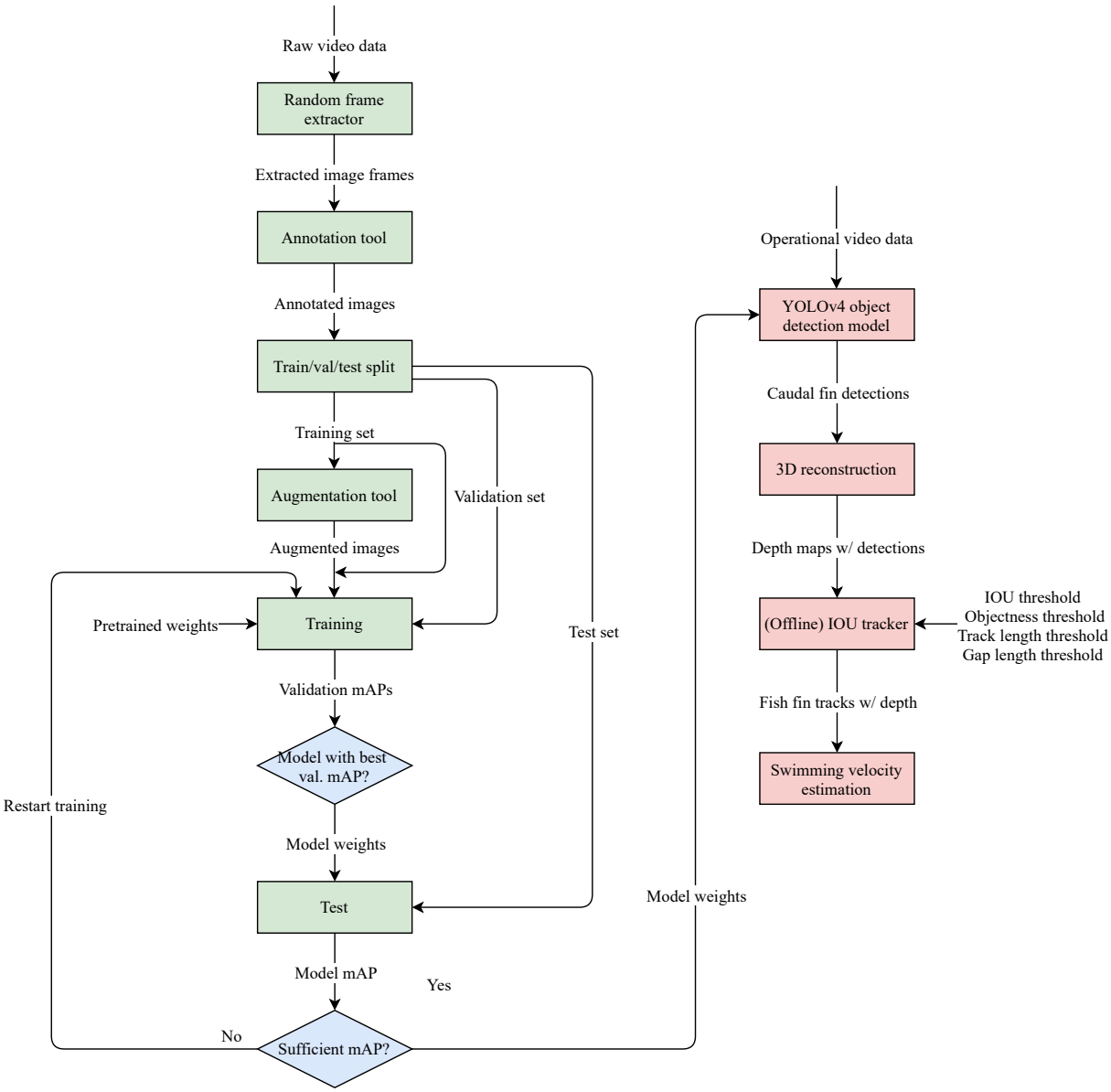


Figure 5.10: Detection pipeline.

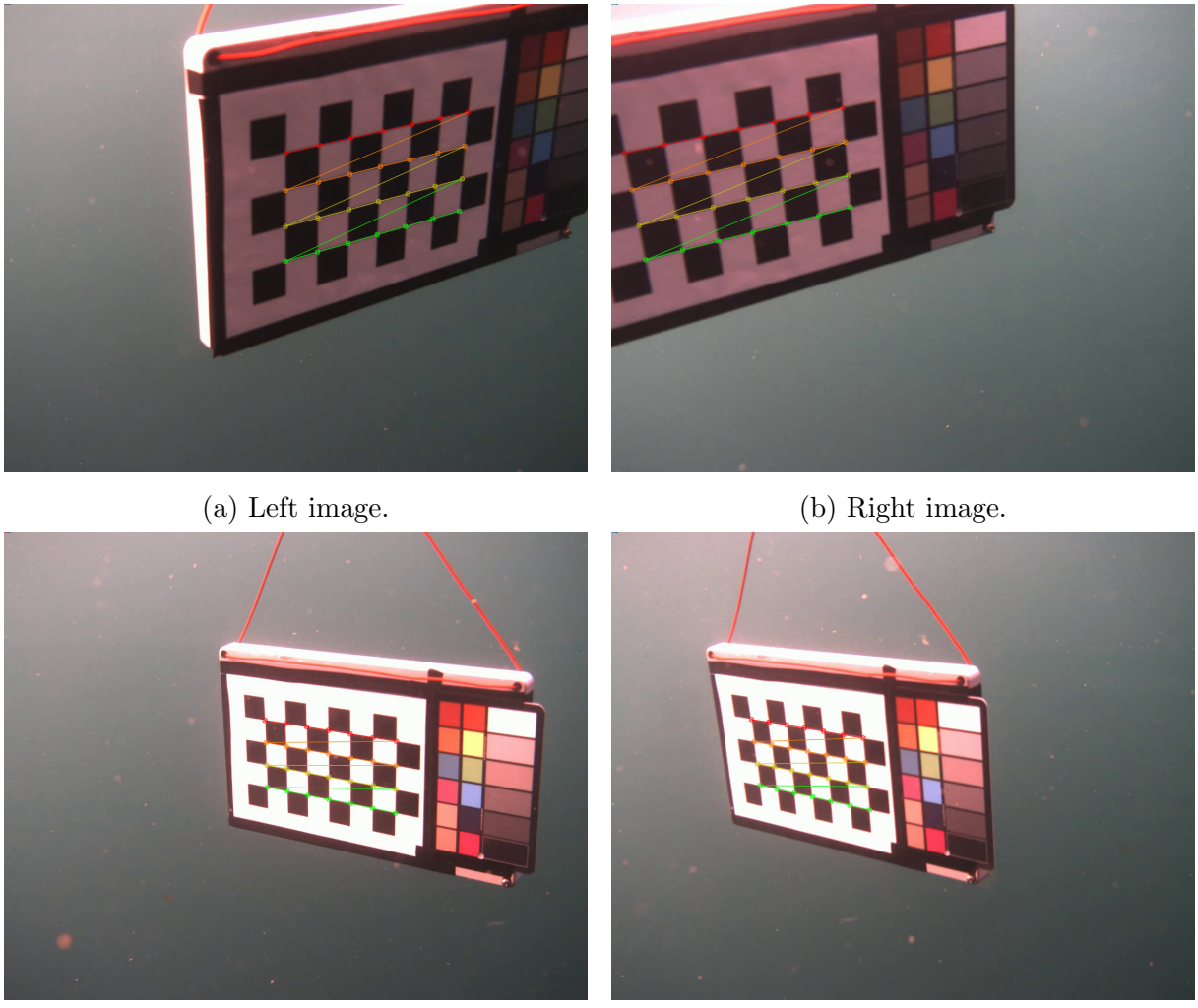
Figure 5.10 shows the object detection pipeline from raw video sequences to the combined system.

5.4 Stereo 3D Reconstruction

OpenCV's implementations of functionality for stereo 3D reconstruction were used to calibrate the stereo cameras and to estimate disparity maps.

5.4.1 Camera Calibration

The camera pair was calibrated using OpenCV Python implementations for camera calibration. To estimate the camera parameters, 3D points in real world space and 2D image points must be obtained. This mapping can be acquired using multiple image pairs of a known calibration pattern from different viewpoints, e.g., chessboard patterns as included in the LAKSIT dataset. In total, 21 image pairs of calibration images of a 7×4 checkerboard were used for this task. The dimension of the checkerboard squares was given, as 30.4 millimeters, or 0.0304 meters. First, two empty lists for each camera are created. One is for storing object points, corresponding to 3D points in real world space. The other is for storing image points, i.e., 2D points in the image plane. Then, for each of the cameras, we go through all calibration images. The chessboard corners are detected by using OpenCV functionality. Examples of chessboard detections are shown in Figure 5.11.



(a) Left image. (b) Right image.

(c) Left image. (d) Right image.

Figure 5.11: Chessboard detections in two calibration stereo pairs, (a) and (b), and (c) and (d), respectively.

After detecting chessboard corners, the camera intrinsic and extrinsic parameters are found by using the OpenCV function `calibrateCamera()` using the object points and image points for each camera. This function computes the camera matrix, distortion coefficients, and rotation vectors and translation vectors estimated for each pattern view.

The reprojection error was used as a measure to evaluate the camera calibration. The goal was to achieve a reprojection error below 1.0 pixels, and as close to zero as possible. First, the reprojection error for each image and the total reprojection error was computed. Then, outliers (image pairs with high individual errors) were removed to decrease the total error. After obtaining sufficient results from camera calibration in terms of a low reprojection error, the calibration parameters were stored and used for each image pair in undistortion, rectification and finally disparity estimation.

5.4.2 Undistortion and Rectification

Unlike the camera calibration, which is only carried out one time for the stereo setup, the undistortion and rectification process must be executed for each stereo pair. The OpenCV function `undistort()` takes the intrinsics and extrinsics from the calibration stage, to undistort and rectify a stereo pair. An example of a rectified image pair after this process is shown in Figure 5.12.

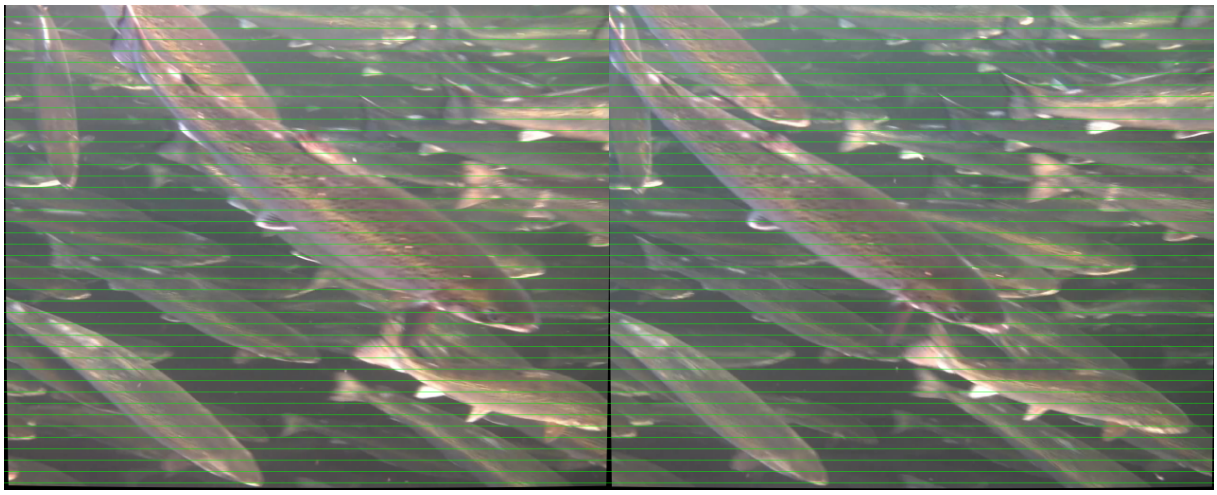


Figure 5.12: Example of rectified image pair.

5.4.3 Stereo Matching

OpenCV's implementation of semi-global block matching was used for obtaining disparity maps. In the following sections, some details about the choice of matching algorithm and tuning of the algorithm are explained.

5.4.3.1 Selection of Stereo Matching Algorithm

As the brief overview in Table 4.1 shows, there are several approaches for stereo matching. Dense disparity maps were desired to obtain depth estimates of entire surfaces. A local method was preferred to reach a certain performance. Hence, block matching was chosen for calculating disparity maps. OpenCV's implementations of block matching (`StereoBM`) and semi-global block matching (`StereoSGBM`) were tested on a selection of stereo pairs. In general, the disparity maps obtained with simple block matching had less remaining points after disparity filtering than disparity maps obtained with SGBM. Despite a higher computational complexity, SGBM was considered as a good compromise between accuracy and performance.

5.4.3.2 Tuning of Stereo Matching Parameters

The `StereoSGBM` algorithm takes a number of parameters that must be tuned for each specific case or dataset. OpenCV's documentation (OpenCV 2021) explains the details of the different parameters. The parameters were tuned by OpenCV's recommendations, and the final values can be seen in Table 5.7.

Table 5.7: Tuning parameters for OpenCV `StereoSGBM` algorithm.

Parameter	**Value**
minDisparity	16
numDisparities	96
blockSize	11
P1	8·`number_of_image_channels`·`blockSize`·`blockSize`
P2	32·`number_of_image_channels`·`blockSize`·`blockSize`
disp12MaxDiff	-1
uniquenessRatio	5
speckleWindowSize	10
speckleRange	32
mode	`StereoSGBM::MODE_HH`

5.4.4 Disparity Post-Processing and Triangulation Pre-Processing

The raw disparity maps were post-processed by adding OpenCV implementation of weighted least squares (WLS) filter, based on Min et al. (2014). Before the 3D positions are computed to create pointclouds, the processed disparity maps were filtered by keeping only disparities larger than the minimum disparity value. Without this filtering, there will be

a "wall" with all the points with the minimum disparity (i.e., invalid points, or points far away) in the back of the pointcloud.

5.4.5 Triangulation and Pointclouds

To compute the 3D positions of the points of an image from the disparity map to obtain pointclouds, the OpenCV function `reprojectImageTo3D()` was used. This function takes the disparity map and the Q matrix, which is a 4×4 perspective transformation matrix (disparity-to-depth mapping matrix) that can be obtained with the OpenCV function `stereoRectify()`. These functions are based on the derivations in section 4.4.

5.4.6 Depth Estimation Pipeline

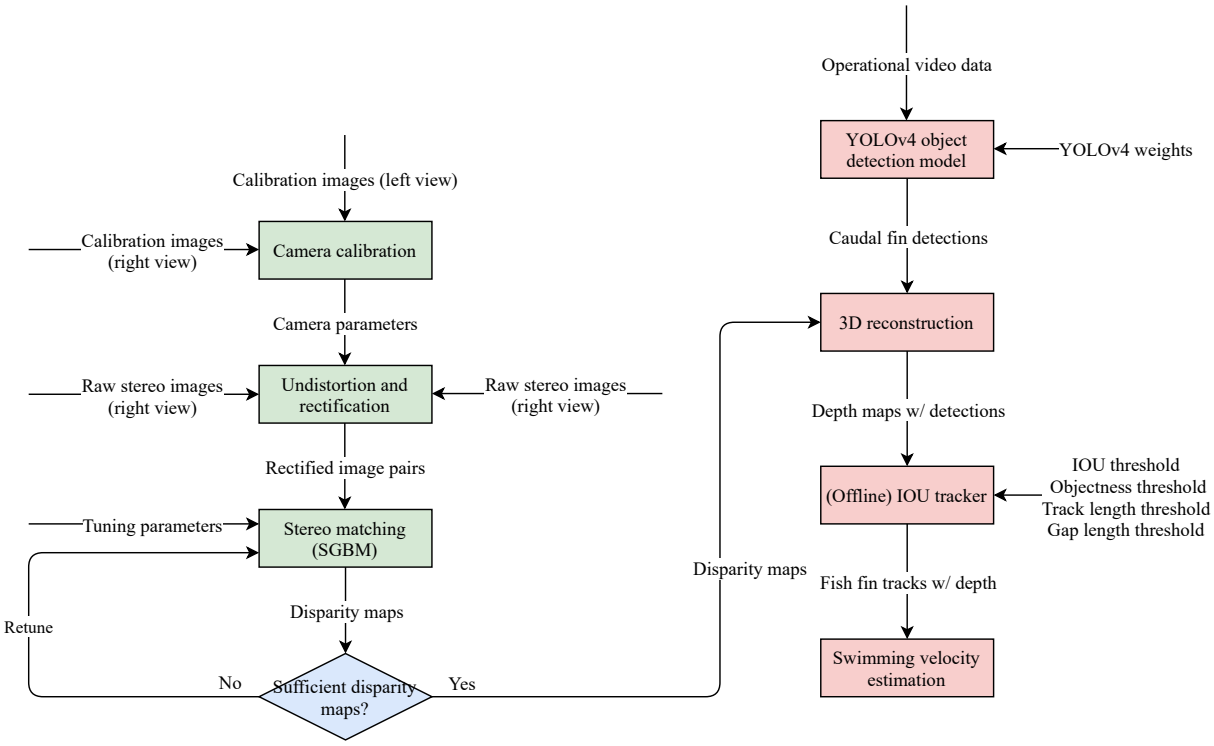


Figure 5.13: Disparity estimation and 3D reconstruction pipeline.

Figure 5.13 shows the final pipeline of the depth estimation model, from raw stereo images to the overall pipeline of object detection, image scene depth estimation, tracking, and finally swimming velocity estimation.

5.5 Tracking Algorithm

Since this project mainly focuses on industrial applications, it was desired to investigate an algorithm that could run online and with minimal computational complexity. Also, simplicity of implementation and scalability was highly weighted for the same reasons as for the detection model. An algorithm with prediction capabilities, like the Kalman filter, would normally be preferred when aiming to track objects that are subject to occlusion like in this case. This would however lead to another problem; it would be challenging to develop a motion model since the fins' relative motion to the camera varies due to the different camera positions and rotations. In this case, simplicity was prioritized. We attempted to improve the impact of occlusion and the performance of the detection model by modifying the IOU tracking algorithm by Bochinski et al. (2017) described in section 3.3.3 to allow "gaps" in the detection.

5.5.1 Modified IOU Tracking Algorithm

The original IOU tracking algorithm by Bochinski et al. assumes that there are none or few gaps in the detections. The detection results had several object sequences with gaps of one or more frames due to both occlusion and detection accuracy, hence the proposed method had to be extended to handle such cases. This was solved by keeping a temporary list with all active tracks with one or more frames that could not find a new match in the current frame, and introduce a threshold t_{gap} for the maximum gap length acceptable, as a "buffer-and-recover" strategy (Luo et al. 2014). If the length of the gap exceeded t_{gap}, the track was concluded as finished. The resulting algorithm is described in Algorithm 2. Since the algorithm was intended for online applications and since it had no predictive capabilities and did not consider image information, we added the previous certain bounding box to cover the gaps. If the algorithm was intended for offline applications, we could utilize the next certain frame after the gap, and do an interpolation over the gap to find suitable bounding boxes during where detection failed. D_f denotes detections in frame f, d_j is the j^{th} detection in D_f, T_a are active tracks, T_f are finished tracks, and T_t is a set of temporary tracks. All active tracks that do not find a match in frame f will be removed from T_a and added to T_t, and will remain there until a new match is found in a subsequent frame or until the gap is larger than the gap threshold t_{gap}.

Algorithm 2 IOU Tracker Bochinski et al. (2017)

Input:
$D = \{D_0, D_1, ..., D_{F-1}\} = \{\{d_0, d_1, ..., d_{N-1}\}, \{d_0, d_1, ..., d_{N-1}\}, ...\}$
$\sigma_{IOU}, \sigma_l, \sigma_h, t_{min}, t_{gap}$

Initialize:
$T_a = \emptyset, T_f = \emptyset, T_t = \emptyset$
$D = \{\{d_i | d_i \in D_j, d_i \geq \sigma_l\} D_j \in D\}$
for D_f in D:
  for $t_i \in T_a$:
    $d_{best} = d_j$ where $max(IOU(d_j, t_i)), d_j \in D_f$
    if $IOU(d_{best}, t_i) \geq \sigma_{IOU}$:
      add d_{best} to t_i
      remove d_{best} from D_f
    else if $max\_score(t_i) \geq \sigma_h$:
      add t_i to T_t
      add t_i to T_f
  for $t_i \in T_t$:
    $d_{best} = d_j$ where $max(IOU(d_j, t_i)), d_j \in D_f$
    if $IOU(d_{best}, t_i) \geq \sigma_{IOU}$:
      add d_{best} to t_i
      $t_a = t_f$ where $id(t_f) = id(t_i), t_f \in T_f$
      $t_a = t_a + t_i$
      remove t_a from T_f
      remove d_{best} from D_f
      remove t_i from T_t
    else if $gap\_length(t_i) \leq t_{gap}$:
      add t_i to T_t
  for $d_j \in D_t$:
    start new track t with d_j and insert to T_a
$T_f = T_f + T_a$
for $t_j \in T_f$:
  if $max\_score(t_i) \geq \sigma_h$ and $len(t_i) \geq t_{min}$:
    add t_j to T_{res}
return T_{res}

The parameters can be tuned to fit the detection input. A small σ_{IOU} will give more matches but increases the possibility of mismatches. If detections are not very accurate or objects are very small, it might be desirable to keep this value small. Similarly for σ_h, if this parameter is small the requirements for a "good enough" track are decreased. Since

σ_l decides how good a detection must be to included as a candidate for a track, a small value could induce false positives to be included in tracks. t_{min} determines the minimum length of a track, hence, a small value will give fewer, but better tracks, which especially is desirable if detection results are good in the first place. However, a larger value may be better if detection results are questionable. Lastly, t_{gap} decides the maximum acceptable length of gaps in the tracks. This means, if the value is small, we require detections in almost every frame.

It must however be noticed that there is at least one pitfall here: if detection results are highly inaccurate, changes in the parameters will not necessarily improve the tracking results. This may for instance cause numerous tracks to be created based on false positives.

5.5.2 Evaluation of Tracking Algorithm

Evaluation of object tracking algorithms is definitely not trivial. How such an algorithm is evaluated is obviously dependent on what aspects we want to assess. Speed can easily be determined by, e.g., measuring frames per second (FPS). Further, metrics for evaluation of detection-based MOT algorithms can be divided into four subsets, namely, accuracy, precision, completeness, and robustness. Accuracy measures how accurately the algorithm tracks the targets. Precision is, as described in the previous section, a measure of how precisely the objects are tracked. Metrics for completeness indicate how completely ground truth trajectories are tracked, and robustness assesses how capable the algorithm is to recover from occlusion. For further aspects of MOT evaluation, Luo et al. (2014) provide a review of MOT and MOT evaluation metrics.

The formal methods for evaluating these aspects are based on metrics such as the number of true positives, amount of false positives, the overlap of bounding boxes, or deviation from the ground-truth trajectory. All of these metrics require labeling of ground-truth behavior. This requires extensive work beforehand and would go beyond the time limits of this project as it would also require additional work to set up and implement a code framework for the assessment. Hence, the tracking algorithm's performance in this thesis will be evaluated mainly by qualitative measures, i.e., it will be discussed based on manual observations of its behavior on test video clips. However, some metrics will be provided to compare the relative behavior of the algorithm with and without modification.

5.6 Swimming Velocity Estimation

This section describes how detection, depth estimation, and tracking are combined to obtain 3D positions of caudal fins in a video sequence, and how velocity vectors are

computed from these.

5.6.1 3D Position of Individual Caudal Fins

To obtain velocity vectors and absolute velocity of the caudal fins, their 3D position must be determined for each detection. Each detection consists of a rectangular bounding box surrounding the fin. This means, the entire area in the bounding box will not contain fin surface, since the fin has a triangular shape. From investigating the detections, it is likely that the middle inner area of each bounding box will cover only fin surface. Hence, a small area in the center of the bounding box was used to determine the goodness of the disparity estimation of the fin. The inner middle area was defined as shown in Figure 5.14, i.e., with a width and height of 10% of the bounding box width and height, respectively. A minimum size of 10×10 pixels was set to get a certain number of disparity points to consider even for small bounding boxes. For each bounding box in each frame, the following steps are performed:

1. Compute the width, height, and position of inner bounding box/area.
2. Compute the mean disparity value of the inner area.
3. Compute the variance σ_i of the disparities in the inner area.
4. Keep the disparity only if $\sigma_l < \sigma_i < \sigma_h$, else discard the disparity value.

The low threshold σ_l introduced to filter out areas of invalid disparity, like the left area of the left view and the right area of the right view, i.e., areas not covered by both cameras' field of view. The high threshold σ_h is introduced to filter out non-smooth surfaces. For the final system, σ_l was set to 10^{-6} and σ_h to 10^{-1}.

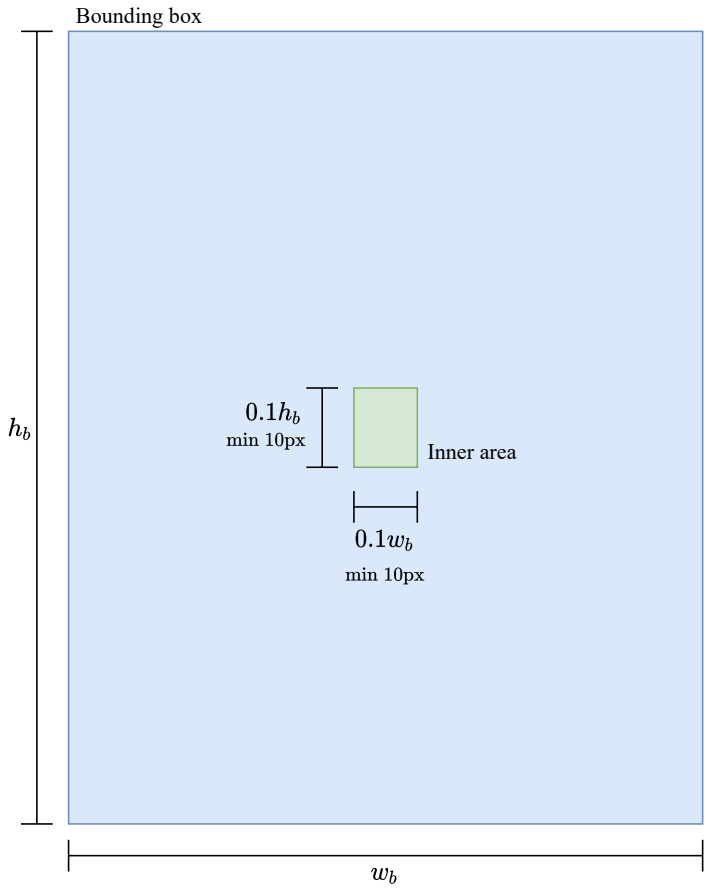


Figure 5.14: Definition of the inner area of the bounding box, where the mean and variance of the disparity is calculated.

Further, the mean disparity value of each bounding box is used to find the 3D position of each caudal fin detection, together with the center point of the bounding box, according to the equations in section 4.4. This serves as an estimate for the the individual fish's position in a certain image frame.

5.6.2 3D Velocity

When each bounding box center point in each frame is assigned a 3D position estimate, the final step is to find the mean velocity of each fin track in the video sequence. First, the velocity vector $v = (v_x, v_y, v_z)$ for each fin, i.e., each 3D point P_i, between every pair of frames is calculated, as

$$v_x = \frac{P_{i+n,x} - P_{i,x}}{n}, \tag{5.5}$$

$$v_y = \frac{P_{i+n,y} - P_{i,y}}{n}, \tag{5.6}$$

$$v_z = \frac{P_{i+n,z} - P_{i,z}}{n}, \tag{5.7}$$

where n is the number of frames between the consecutive valid detections/depth estimates. Further, the absolute velocity $|v|$ of a fish between two consecutive frames is calculated, as

$$|v| = \sqrt{v_x^2 + v_y^2 + v_z^2}, \tag{5.8}$$

which has the unit of millimeters per frame. Then, the mean velocity is calculated for each track and converted to meters per second by using the number of frames per second of the video sequence, representing the mean velocity of an individual fish.

5.6.3 Final Pipeline

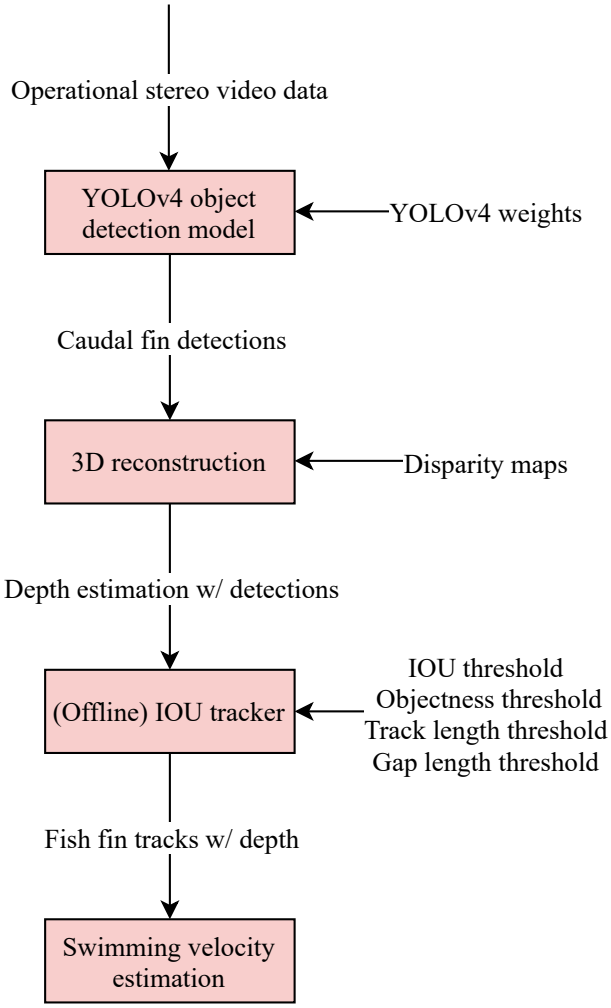


Figure 5.15: Final overall pipeline, including caudal fin detection, image scene depth estimation, and tracking.

Figure 5.15 shows a flow diagram of the final pipeline. Operational stereo video data is passed into the object detector for detection of fish tale fins. The results are, together

with disparity maps calculated from the same stereo images, passed into the depth estimation part, which estimates the 3D position of each tale fin. Further, the detections and the depth estimates are passed into the tracker, using IOU overlap to track the individual caudal fins (corresponding to individual fish). From the tracks and depths, the swimming velocity can be estimated by calculating the linear movement between 3D points in consecutive frames. As indicated, the tracking algorithm was implemented to run offline in this project, i.e., detection and 3D reconstruction is first run on all frames before the results are passed into the tracking algorithm. Recall that the tracking algorithm works by the concept of an online tracker; it only considers current frame and past frames. It is implemented like so in this project as a proof-of-concept and for easing the debugging process.

Chapter 6

Results

This chapter is divided into four parts. First, the performance results from the object detection on a custom validation set are presented. Next, we introduce the results from stereo matching and 3D reconstruction. The following section presents results from testing the tracking algorithm on small video clips to provide semi-quantitative and qualitative assessments of its performance and its limitations. The last section introduces the results from the final pipeline estimating salmon swimming velocity.

6.1 Detection Model

After the final detection model was chosen based on performance on the validation data, it remained to test performance on new, unseen data. First, a speed test was performed, showing that the model was running up to 52.3 FPS, which is more that the frame rate of the videos (24 FPS). Note that the speed is also dependent on the type of hardware (GPU) used, but the result indicates that the model is able to perform in real-time on a low-cost GPU (see section 5.2 for hardware details).

Then the model was tested on a set of unseen images. The test set consisted of 26 images, retrieved from the same video sources as the training and validation data. The model was first tested with different IOU thresholds (0.25, 0.50, and 0.75) at confidence 0.25, and measures for AP, TP, FP, and FN are presented in Table 6.1. This was conducted to assess how well the network performed in fitting good bounding boxes to the objects. Finally, the model was tested with different confidence scores at fixed mAP@0.50. It was expected that the model would struggle to detect caudal fins with high confidence because of the limitations in the dataset, hence, it was desirable to assess this aspect. The results are shown in Table 6.2. Recall section 5.3.3 for definitions of the metrics used.

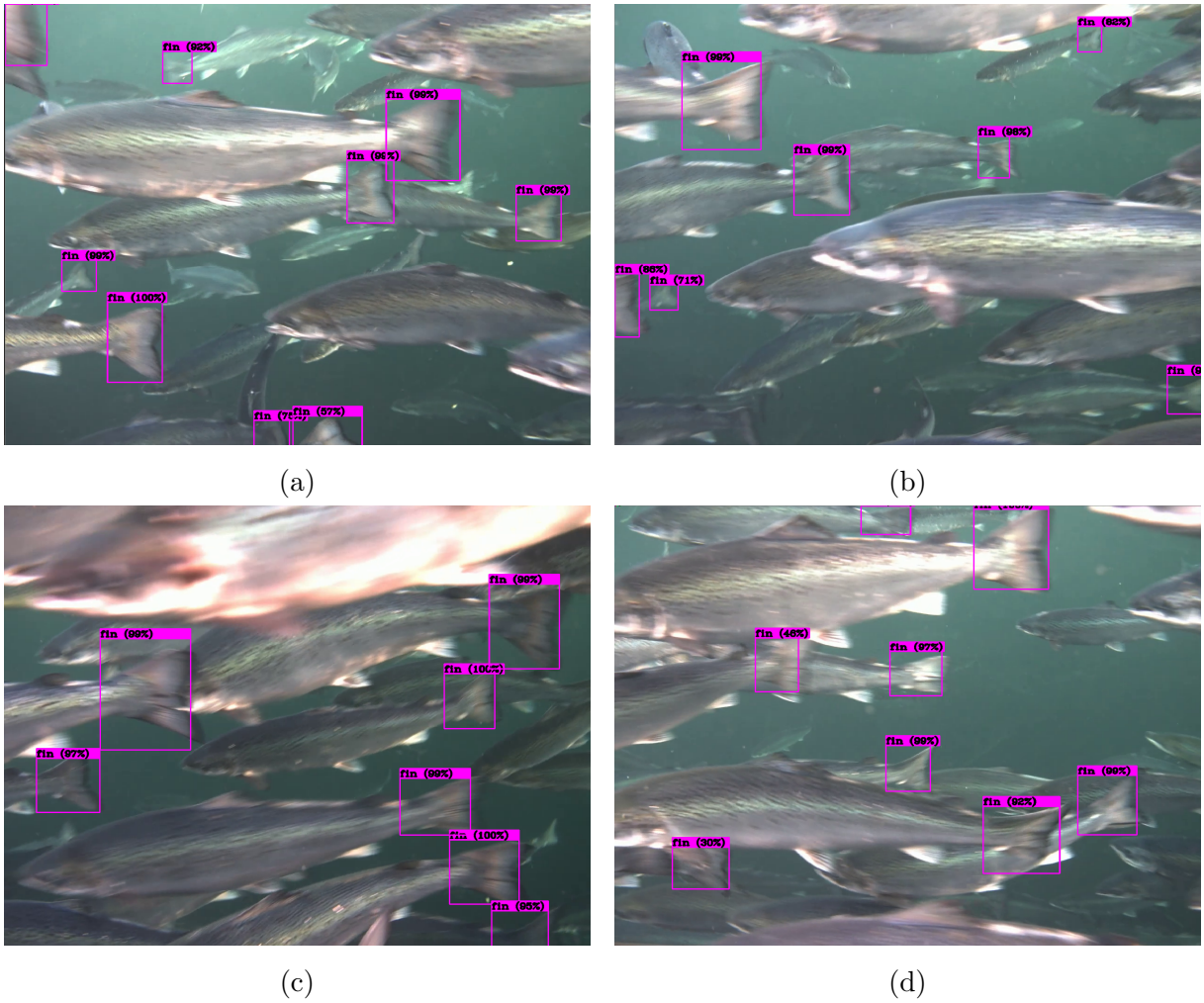
Table 6.1: Accuracy results on test dataset with different IOU thresholds.

IOU threshold	AP	TP	FP	FN
0.25	88.23%	159	75	15
0.50	87.49%	158	76	16
0.75	49.93%	110	124	64

Table 6.2: Accuracy results on test dataset with different confidence thresholds.

Confidence threshold	TP	FP	FN
0.10	161	102	13
0.30	157	71	17
0.50	154	56	20
0.70	147	38	27
0.90	127	17	47

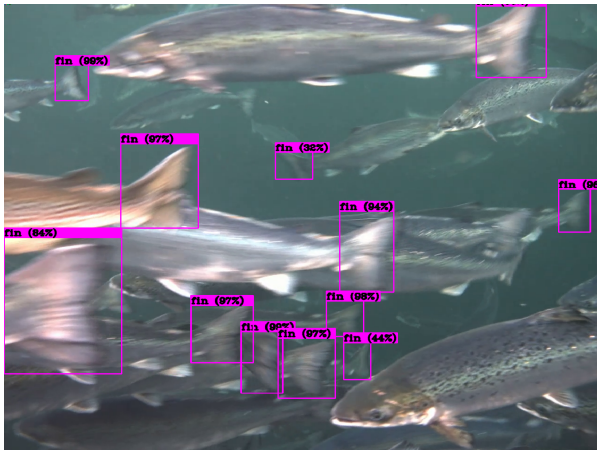
Figure 6.1 shows four examples of detections on images from the test set. Figure 6.2 shows four examples of issues with the detection model. Figure 6.3 shows how the YOLOv4 model detects caudal fins across scales.



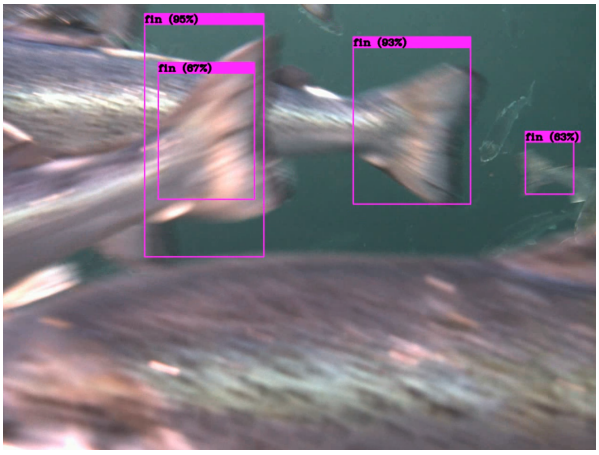
(a) (b)

(c) (d)

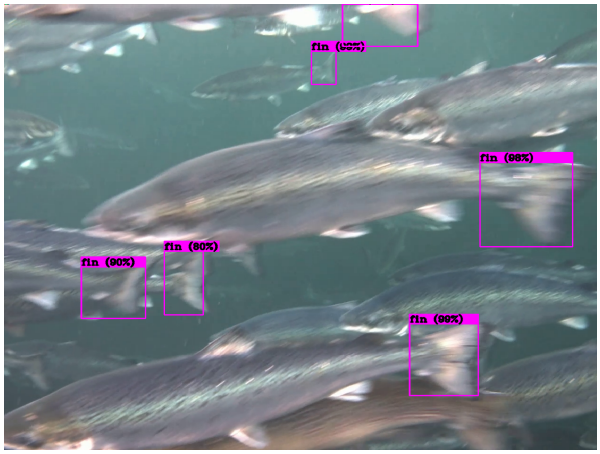
Figure 6.1: YOLOv4 detections on images from test set.



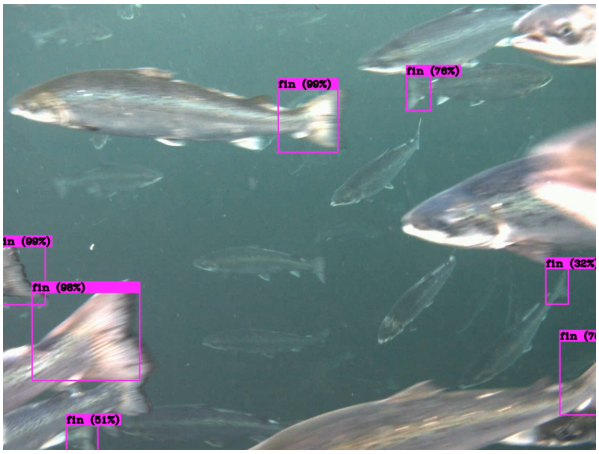
(a) Multiple bounding boxes “clustered” in lower middle part of image.



(b) Double bounding boxes.

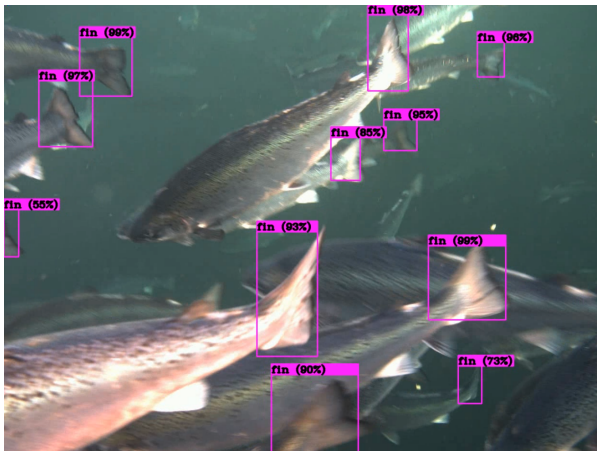


(c) False positive bounding box in upper middle part of image.

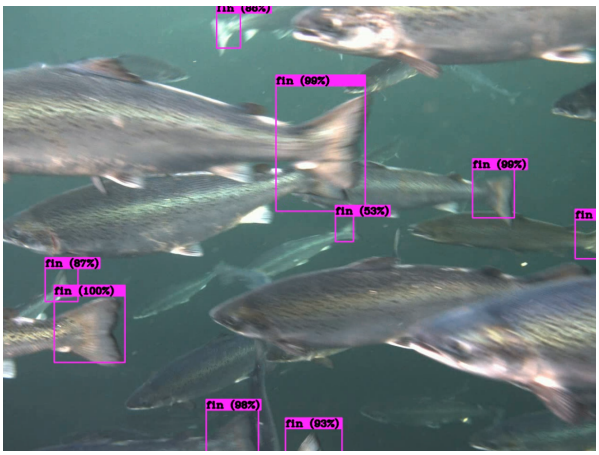


(d) False negative detections, i.e., some caudal fins are not detected.

Figure 6.2: Problematic YOLOv4 detections on images from test set.



(a)



(b)

Figure 6.3: YOLOv4 detections on images from test set, showing detections across scales.

6.2 Depth Estimation

6.2.1 Camera Calibration

The initial mean reprojection errors are shown in Figure 6.4. The total error was measured to 0.2517 pixels. The four image pairs with the highest reprojection errors were removed, improving the reprojection error to 0.2234 pixels (Figure 6.5).

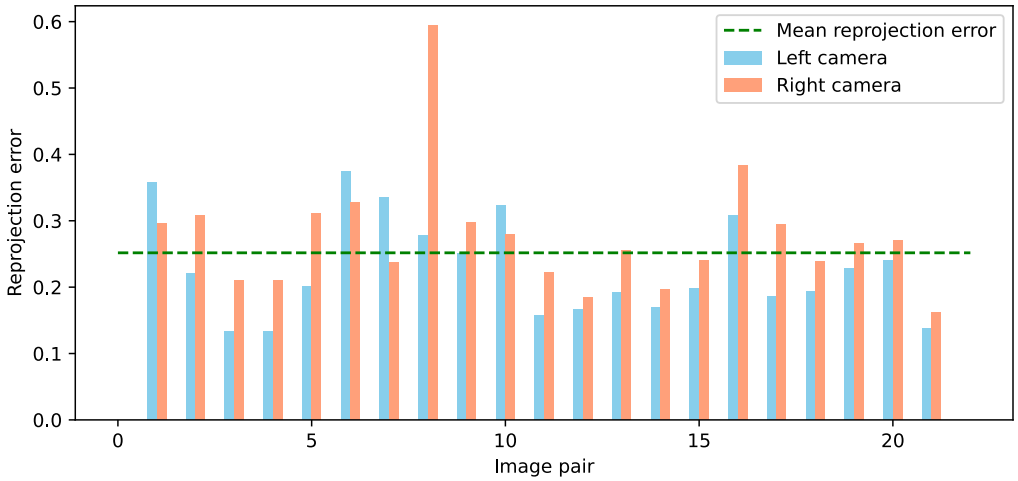


Figure 6.4: Initial reprojection errors. Unit is pixels.

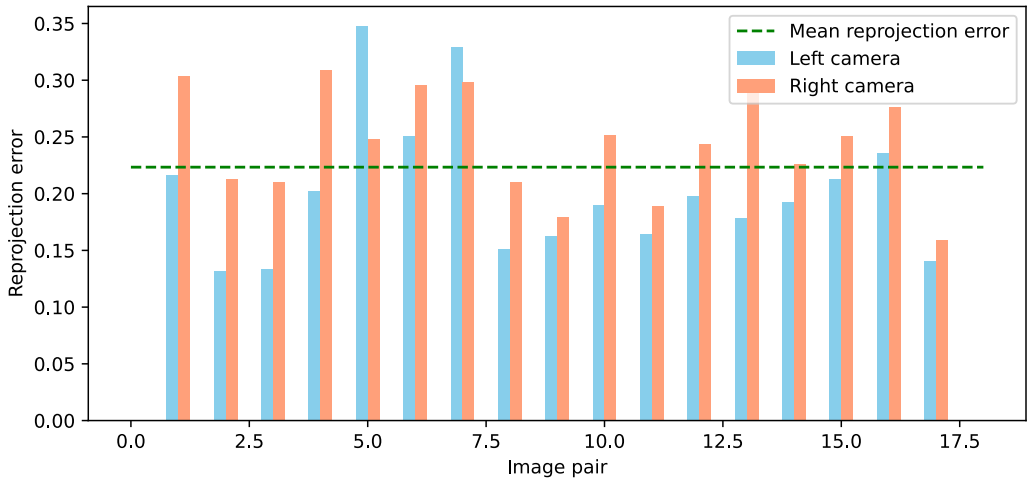


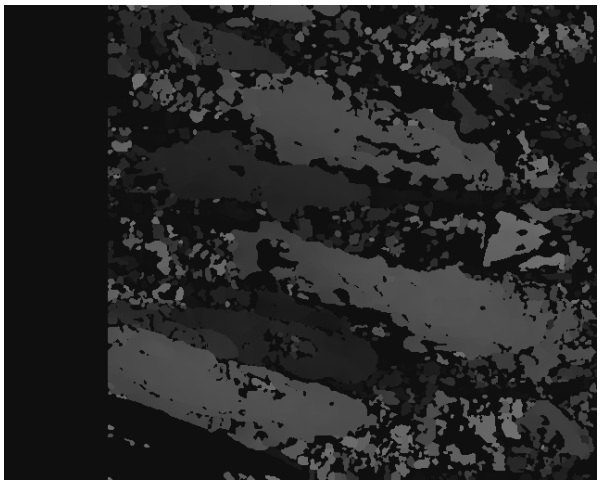
Figure 6.5: Improved reprojection errors. Unit is pixels.

6.2.2 Disparity Post-Processing

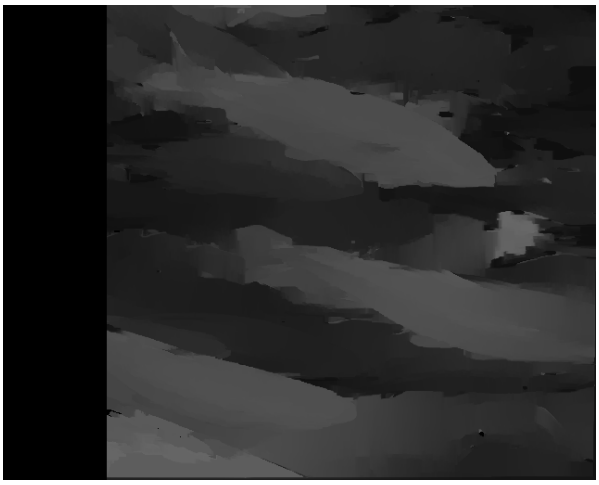
Figure 6.6 and Figure 6.7 show examples of input images and their respective disparity maps before and after filtering with the WLS filter.

(a) Left input image.

(b) Right input image.



(c) Raw left disparity map.

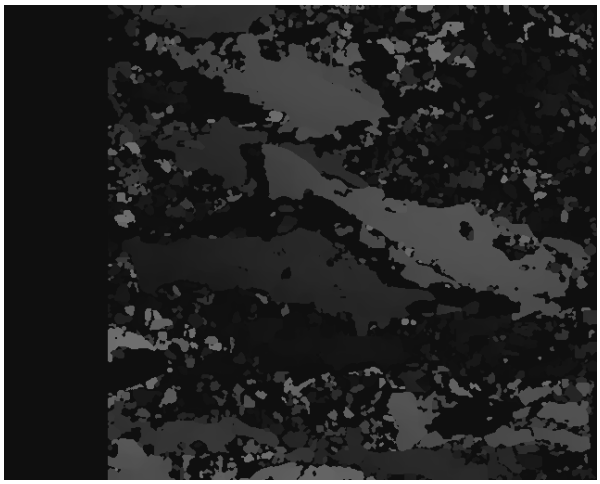


(d) Filtered left disparity map.

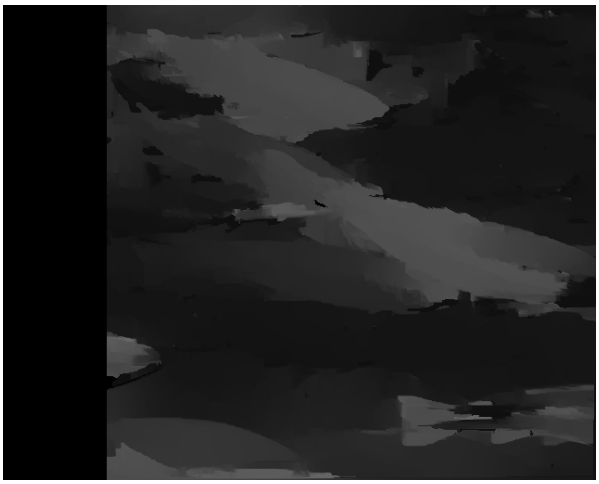
Figure 6.6: Example of input images and resulting raw and filtered disparity maps. The weighted least squares (WLS) filter is used.

(a) Left input image.

(b) Right input image.

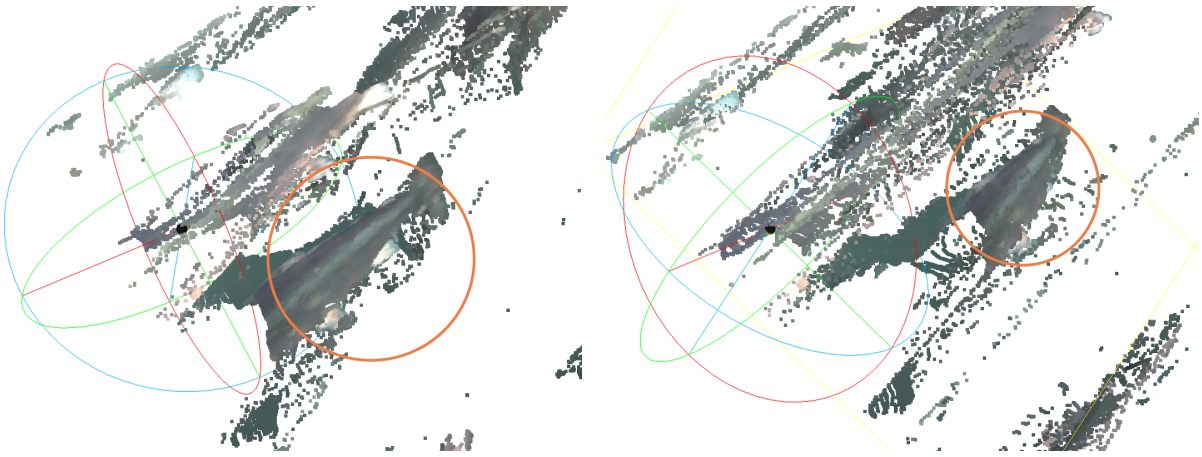


(c) Raw left disparity map.



(d) Filtered left disparity map.

Figure 6.7: Example of input images and resulting raw and filtered disparity maps. The weighted least squares (WLS) filter is used.

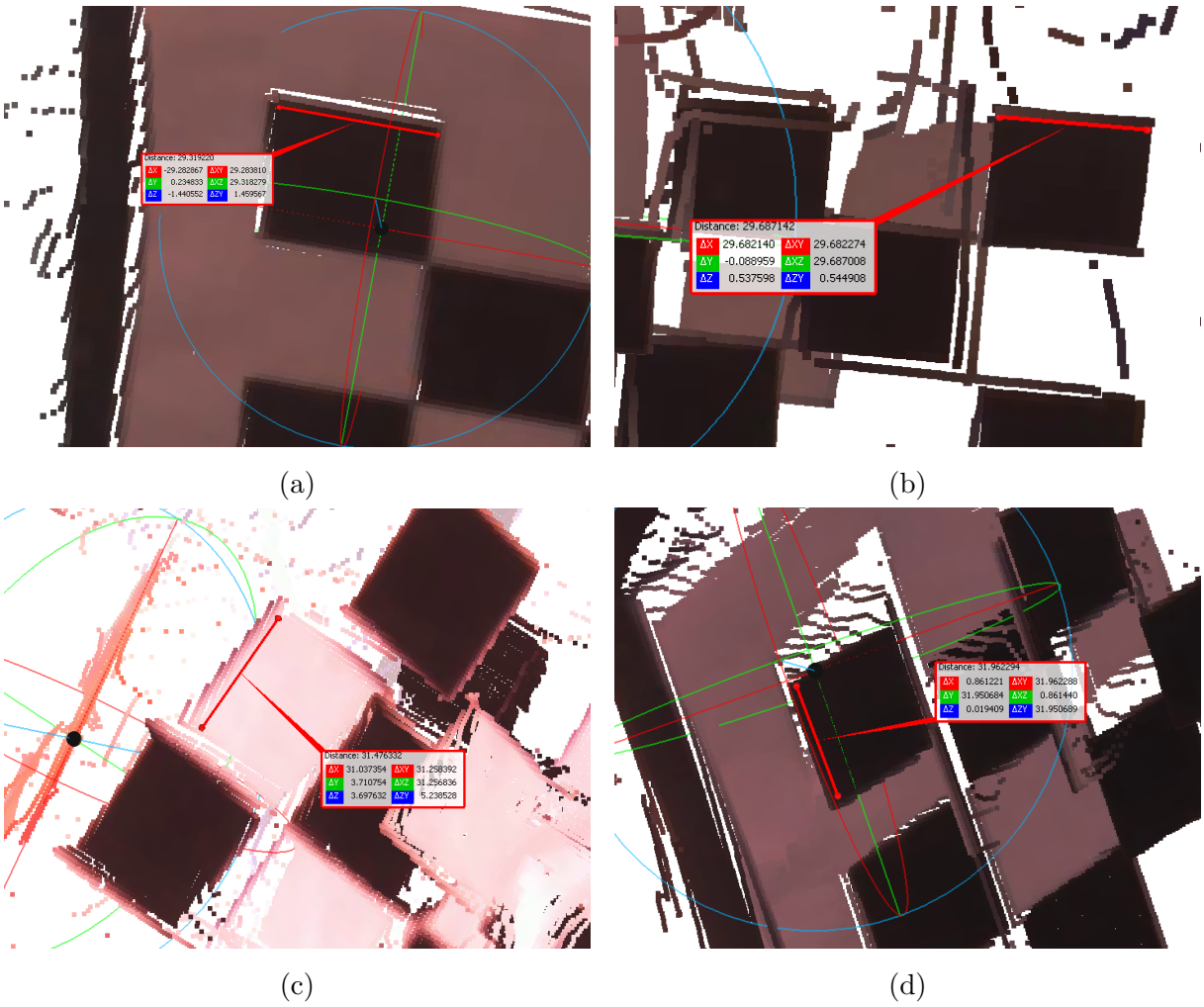


(a) (b)

Figure 6.8: Resulting pointcloud of a filtered disparity map, from two different views angles. Notice the fish surface marked in red and how the smoothed edges of the surface creates speckles/noise.

6.2.3 3D Reconstruction

Figure 6.9 shows measurements of checkerboard square size on pointclouds calculated with the same parameters as for the stereo images. The ground truth square size is 30.4 mm.



(a) (b)

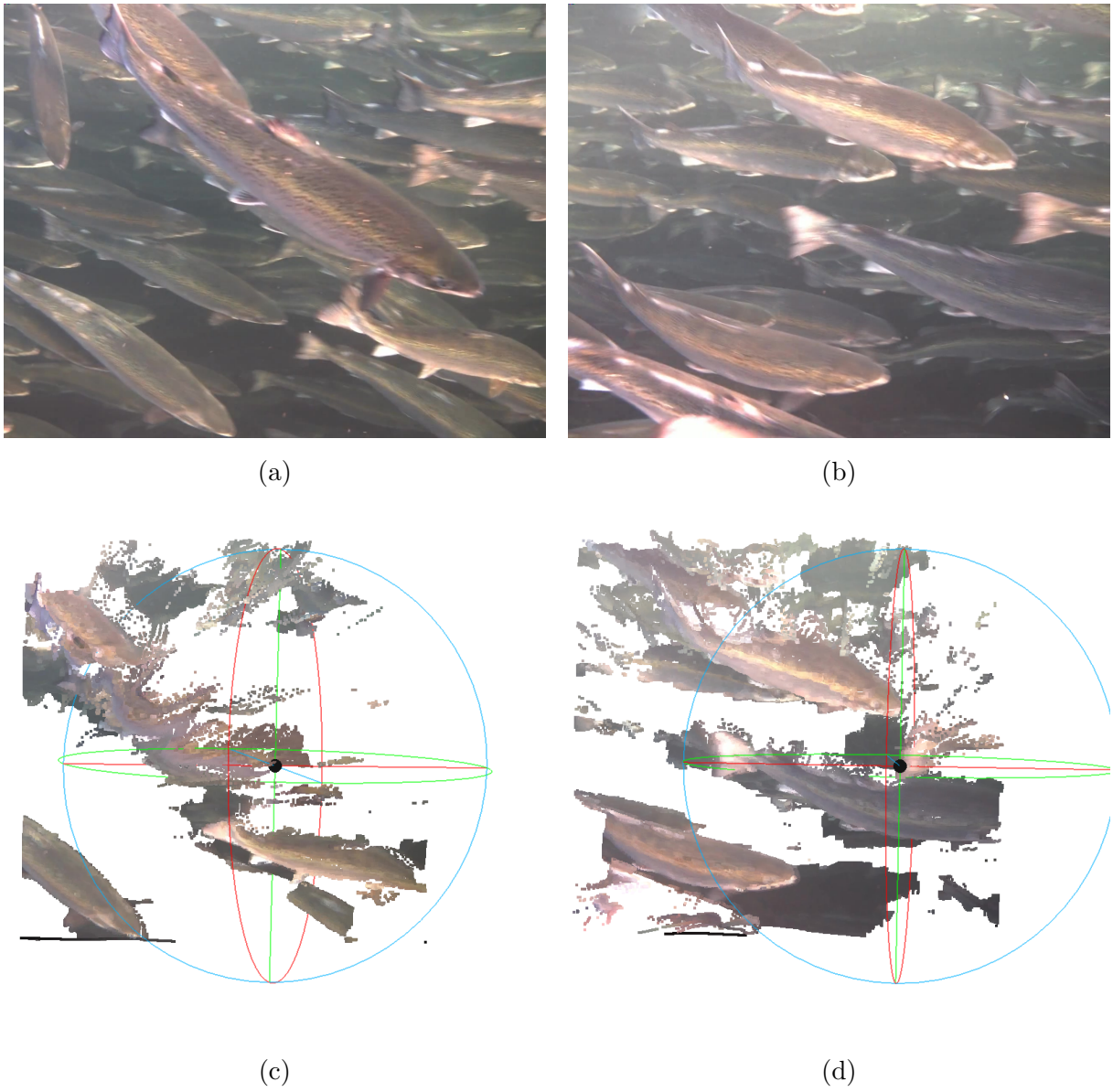
(c) (d)

Figure 6.9: Measurement of checkerboard square size in four different pointclouds. The number behind "Distance" shows the measured distance of the red line. All units are millimeters.

	Measured square size [mm]
Ground truth	*30.40*
Figure 6.9a	29.32
Figure 6.9b	29.68
Figure 6.9c	31.48
Figure 6.9d	31.96

Table 6.3: Measured checkerboard square sizes in Figure 6.9.

Figure 6.10 shows two examples of resulting pointclouds after stereo matching and 3D reconstruction.



(a) (b)

(c) (d)

Figure 6.10: Examples of 3D reconstruction from stereo image pairs. (a) and (b) are original (left) images. (c) and (d) are the pointclouds after 3D reconstruction for images (a) and (b), respectively.

6.3 Video Analysis of Tracking

As stated in section 5.5.2, the tracking algorithm will mainly be assessed in qualitative manners. Nevertheless, some metrics are presented in Table 6.4 to provide some insight into the quantitative performance of the tracking and to compare the algorithms with and without the modification to handle gaps. They are obtained from two test videos, both with a length of 3120 image frames. Note that they are not set in a larger context and are not comparable to results of other tracking algorithms or on other data. The results

in Table 6.4 are obtained with the following values:

$$\sigma_{IOU} = 0.3$$
$$\sigma_h = 0.3$$
$$\sigma_l = 0.7$$
$$t_{min} = 10$$
$$t_{gap} = 10$$

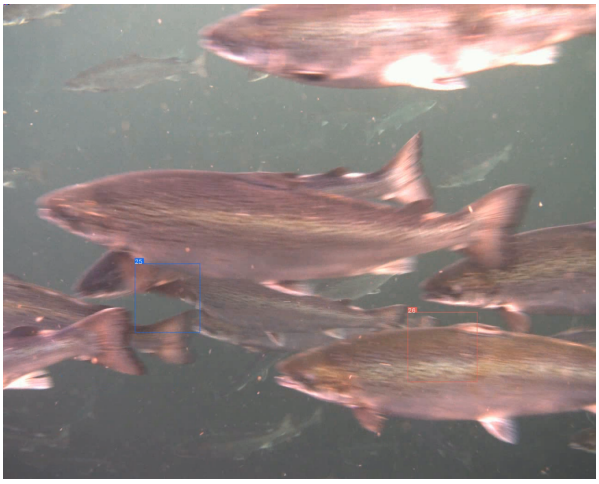
The value of t_{gap} was set to just below half the number of frames per second (24) in the test video since, by prior observation, most gaps could last for up to 0.5 seconds. Since the detection results included several gaps and the camera position was not stable, small σ_{IOU} and σ_h thresholds were chosen. σ_l was chosen as 0.7 as the detection results were reasonably accurate.

Table 6.4: Quantitative results of tracking algorithms using detections from YOLOv4, obtained from two test videos with a length of 3120 frames.

Video no.	**Metric**	**Original**	**With gap modification**
1	Maximum track length	96 frames	128 frames
	Average track length	21.8 frames	28.3 frames
	Total tracks	544	397
2	Maximum track length	100 frames	108 frames
	Average track length	17.9 frames	21.3 frames
	Total tracks	111	238

In Figure 6.11 and Figure 6.12, two different considerations of the choice of IOU overlap threshold are presented.

(a) Frame k.

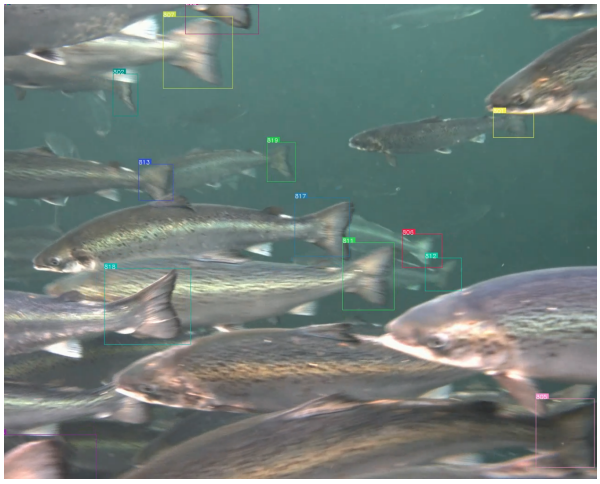


(b) Frame $k+1$.

(c) Frame $k+2$.

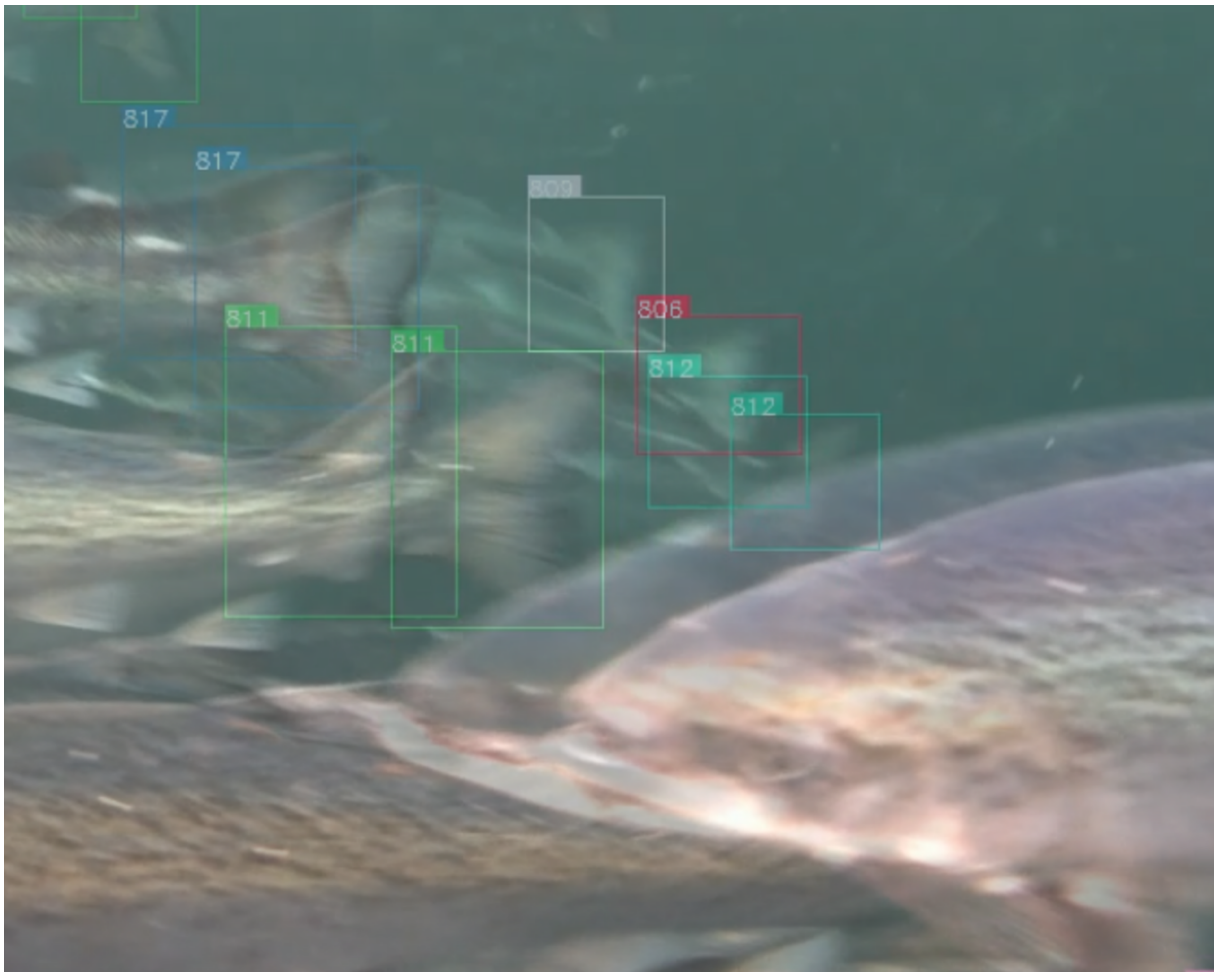
(d) Frame k (a), $k+1$ (b), and $k+2$ (c) layered over each other (zoomed in).

Figure 6.11: Results from object tracking; notice the bounding boxes of frame $k+1$ and $k+2$. They are false positives, but get included in the track.



(a) Frame k.

(b) Frame $k+1$.



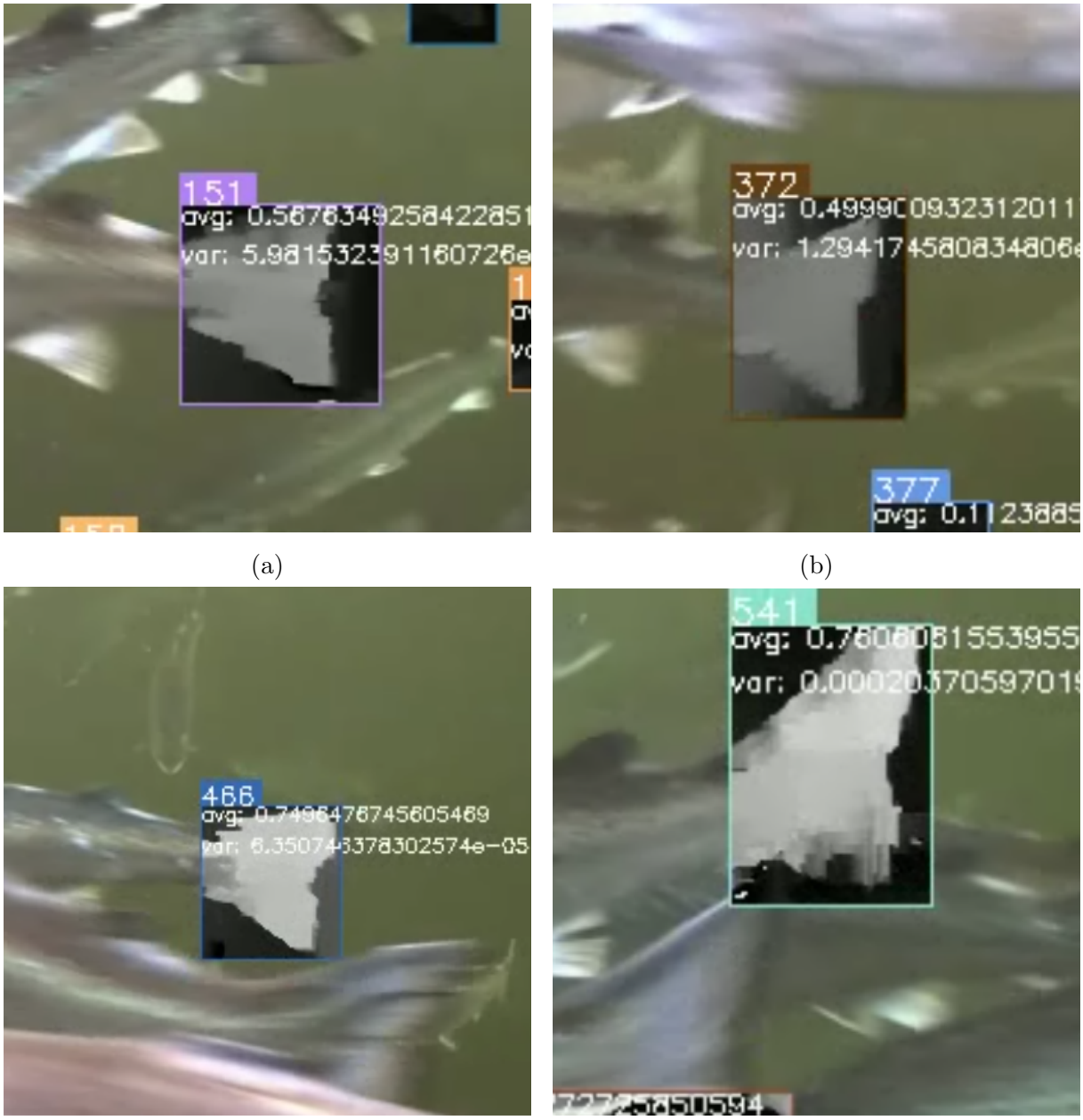
(c) Frames k (a) and $k+1$ (b) layered over each other. Notice the bounding boxes marked 806 and 809, and their IOU overlap.

Figure 6.12: Results from object tracking; too small IOU overlap between consecutive frames.

6.4 Swimming Velocity

This section presents the results from studying the final pipeline on the same test videos as in section 6.3. Note also here that the results are not set in a larger context and are meant to explore the possibilities of 3D tracking and velocity estimation of fish from image/video data.

6.4.1 Detections and Disparities



(a) (b)

(c) (d)

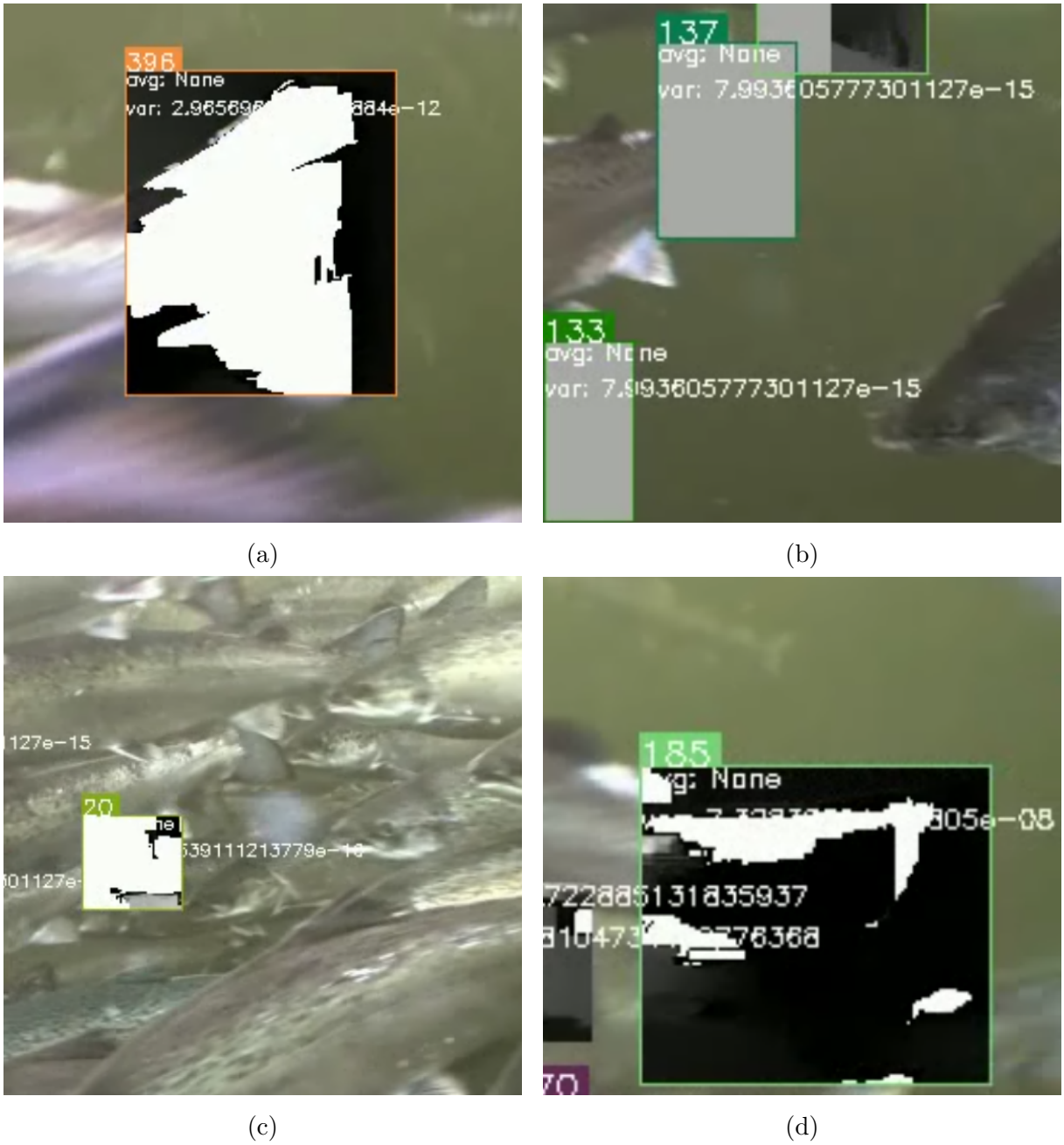
Figure 6.13: Examples of good detections and disparity estimations.

In figures Figure 6.13, Figure 6.14, and Figure 6.15, the text can be interpreted as follows:

- "avg": The mathematical average (mean) disparity value of the inner area of the bounding box, as defined in Figure 5.14. "None" means the variance was found too high and the depth estimation of that detection is discarded.
- "var": The variance of the disparity values of the inner area of the bounding box, as defined in Figure 5.14.

Figure 6.13 shows four examples of good detections and disparity estimations. The bounding boxes are tight around the caudal fins. Most of the caudal fins have a smooth disparity surface, and there is a clear distinction between the fin and the background.

Figure 6.14 shows four examples of discarded depth estimations. Depth estimations are discarded if the variation in the inner area is below a minimum threshold. Figure 6.14a shows a depth estimation that looks sufficient, but is discarded because the entire caudal fin has (close to) the maximum disparity, leaving the variation very small. Figure 6.14b shows depth estimations for fin detections in the left area of the left image, outside the right camera field of view. Since the disparity cannot be calculated in areas not captured by both cameras, this area gets an invalid disparity and must be removed. Figure 6.14c and Figure 6.14d show two depth estimates correctly discarded, as the real surfaces of the fins are not estimated accurately.



(a) (b)

(c) (d)

Figure 6.14: Examples of disparity estimates discarded because of too small variance.

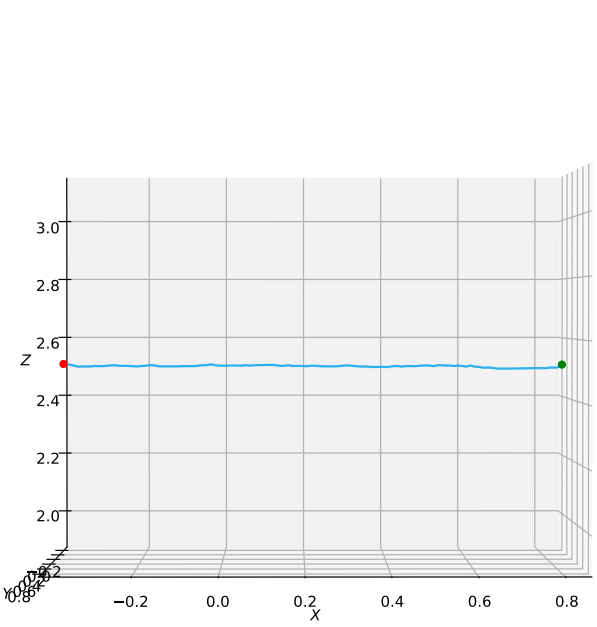
Figure 6.15 shows two examples of disparity estimates discarded because the variation in the inner area is above a maximum threshold.

(a) (b)

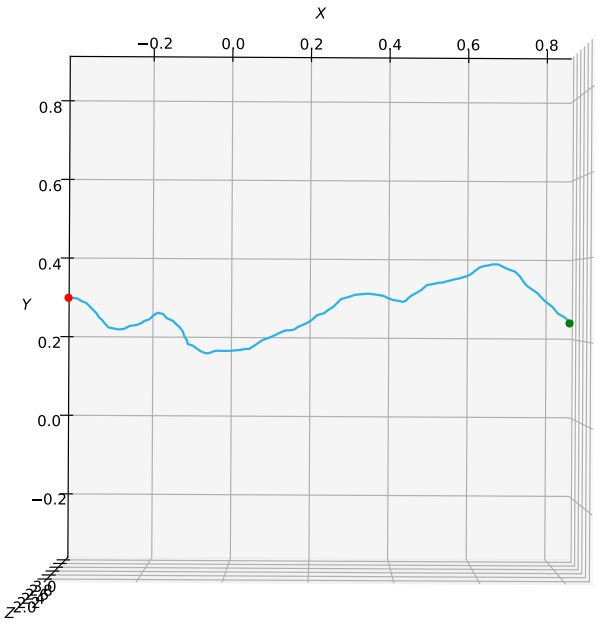
Figure 6.15: Examples of disparity estimates discarded because of too high variance.

6.4.2 3D Tracks

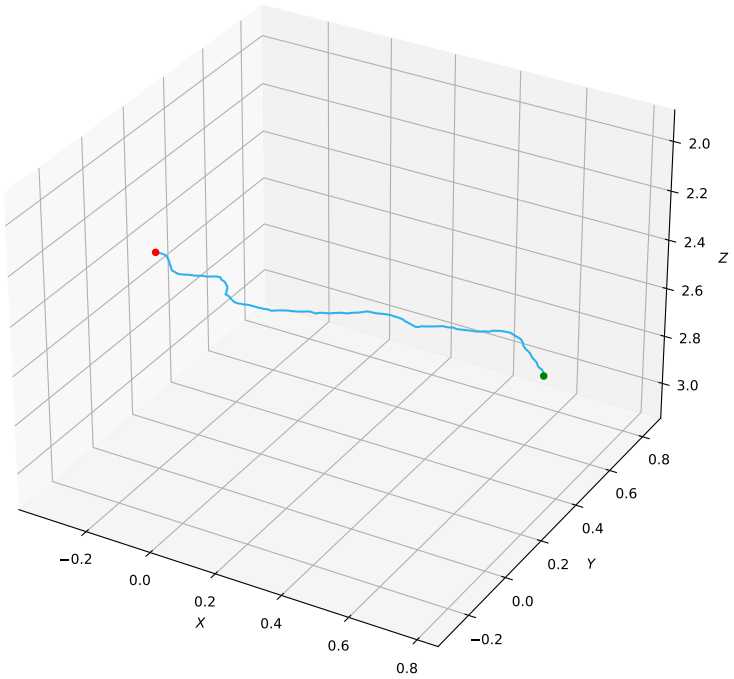
The final 3D tracks were plotted in 3D plots to study if they were reasonable with respect to the actual movement of the fish observed in the test videos. In both test videos, fish are mostly moving from the right to the left, parallel to the camera. Figure 6.12 shows the typical movement of fish in the videos. Note that the caudal fins have an egomotion mainly perpendicular to the motion of the entire fish, i.e., the fin moves back and forth from and towards the camera.



(a) The track seen from the XZ-plane.



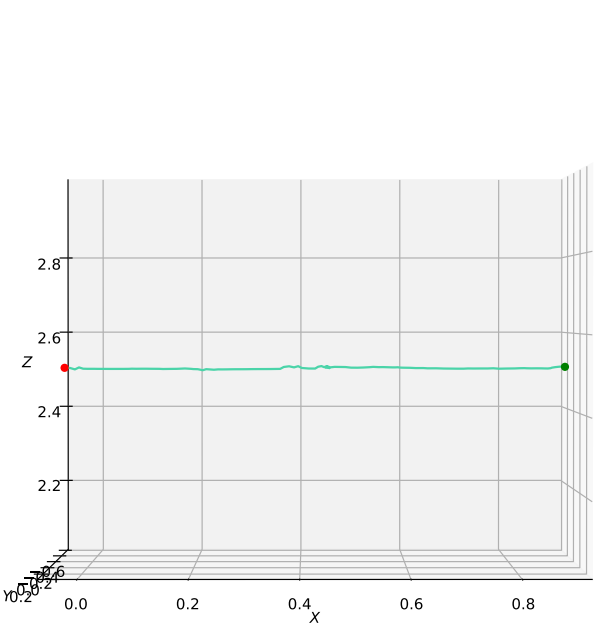
(b) The track seen from the XY-plane.



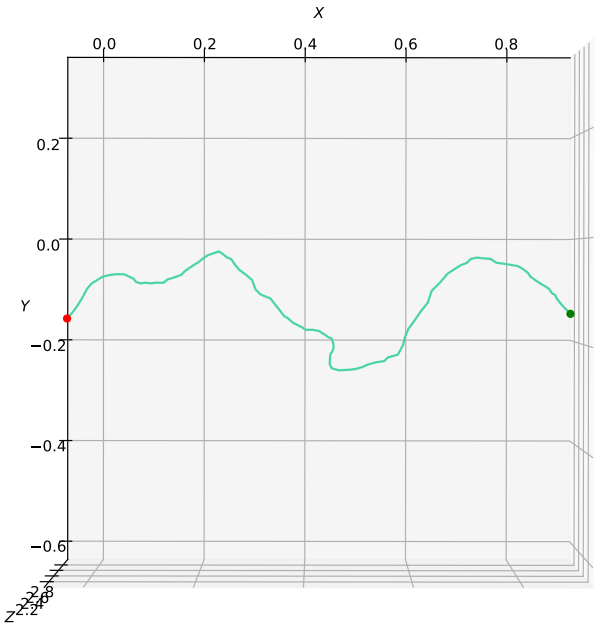
(c) The track seen from the XYZ-plane. Note the dimensions of the axes, and that a larger Z value means larger depth (from the camera).

Figure 6.16: The longest track (128 frames, 5.33 s) of video 1 plotted in 3D coordinates.

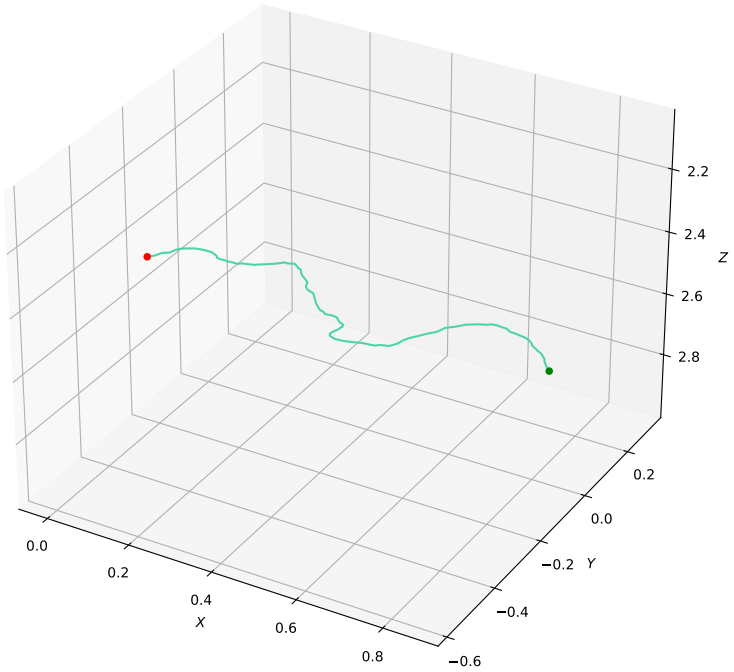
Figure 6.16, Figure 6.17, Figure 6.18, and Figure 6.19 show the 3D path of the longest and second-longest tracks obtained in the two test videos, respectively. The green dot indicates the start point of the track, and the red dot indicates the end point. I.e., from the camera point of view, the fish are swimming from right to left. Unit is meters.



(a) The track seen from the XZ-plane.

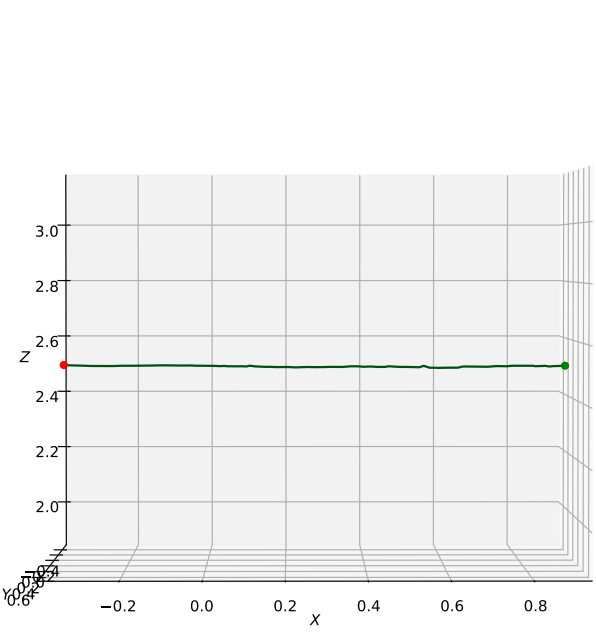


(b) The track seen from the XY-plane.

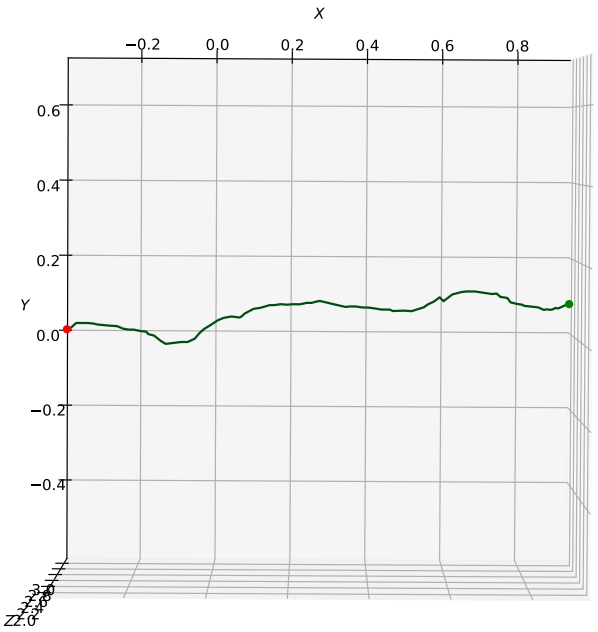


(c) The track seen from the XYZ-plane. Note the dimensions of the axes, and that a larger Z value means larger depth (from the camera).

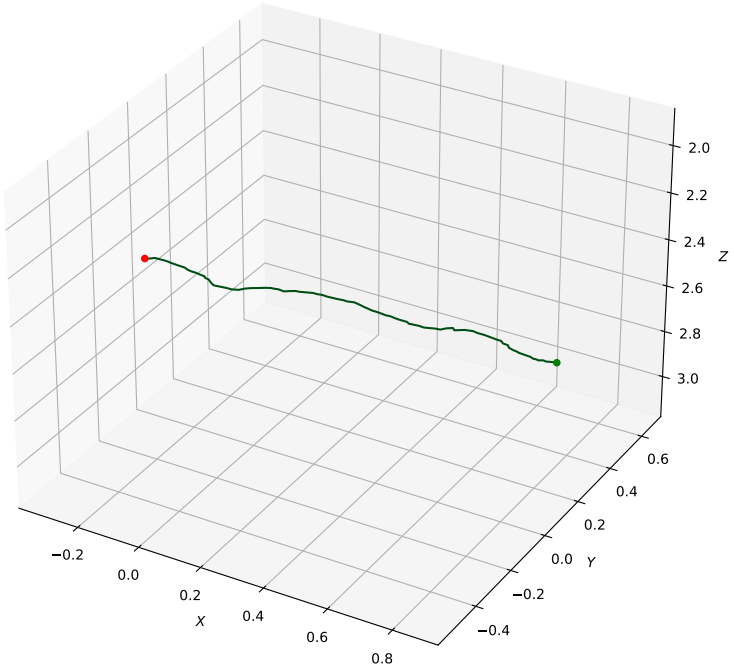
Figure 6.17: The second longest track (108 frames, 4.50 s) of video 1 plotted in 3D coordinates.



(a) The track seen from the XZ-plane.

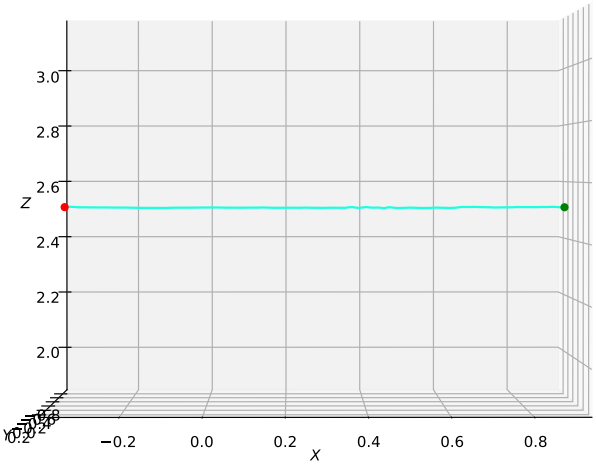


(b) The track seen from the XY-plane.

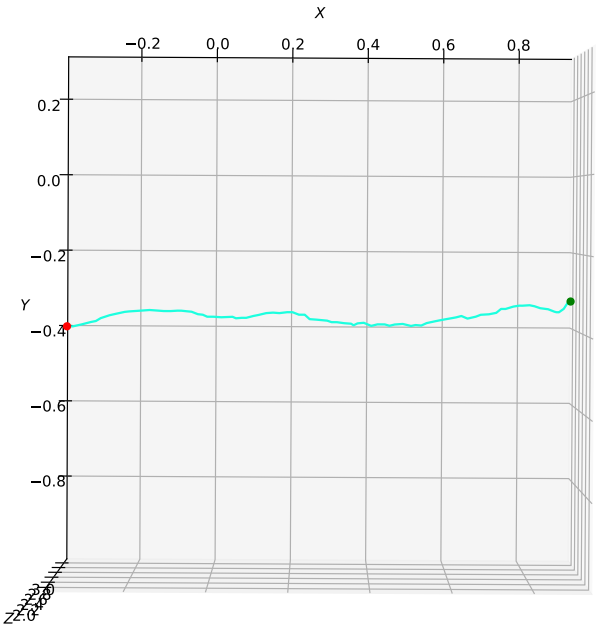


(c) The track seen from the XYZ-plane. Note the dimensions of the axes, and that a larger Z value means larger depth (from the camera).

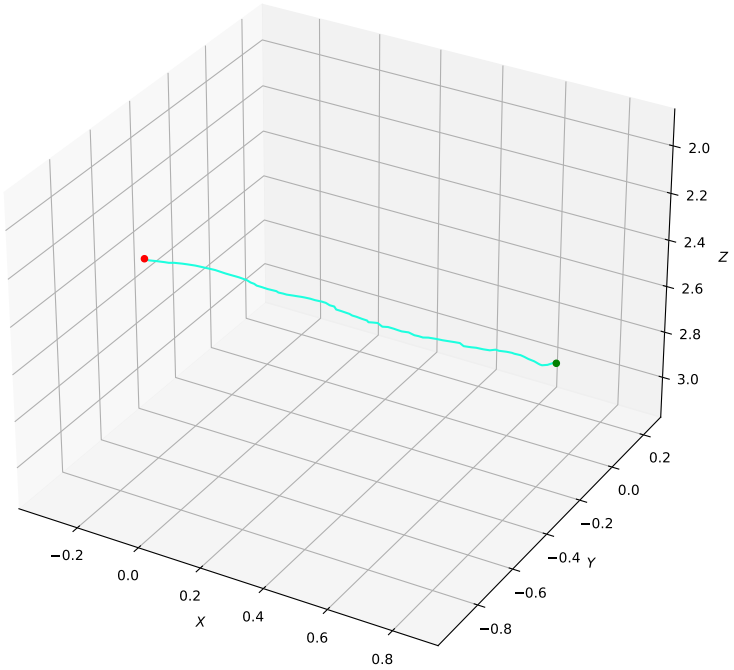
Figure 6.18: The longest track (108 frames, 4.50 s) of video 2 plotted in 3D coordinates.



(a) The track seen from the XZ-plane.



(b) The track seen from the XY-plane.



(c) The track seen from the XYZ-plane. Note the dimensions of the axes, and that a larger Z value means larger depth (from the camera).

Figure 6.19: The second longest track (83 frames, 3.46 s) of video 2 plotted in 3D coordinates.

6.4.3 Mean Velocity and Standard Deviation

Table 6.5 shows the mean velocity, velocity standard deviation, maximum velocity, and minimum velocity calculated over all detected tracks in two test videos.

Table 6.5: Statistics for two test videos.

	Video 1	**Video 2**
Mean velocity [ms^{-1}]	0.5479	0.6561
Velocity standard deviation [ms^{-1}]	0.3175	0.3149
Minimum velocity [ms^{-1}]	0.0707	0.1662
Maximum velocity [ms^{-1}]	2.1427	2.4141

Chapter 7

Discussion

The discussion is divided into five parts. We address issues regarding the dataset and annotation, and analyse the results of each individual component in the system. Finally, the overall pipeline estimating salmon swimming velocity is discussed.

7.1 Dataset and Annotation

7.1.1 Dataset

To reduce the challenges of detection model overfitting without annotating more images, each data point was augmented, resulting in a larger and more variable dataset. Additionally, the (approximate) early stopping point was found as explained in section 2.1.2.3. Nevertheless, since the validation and test data were retrieved from the same video sources as the training data, one cannot be confident that the results are not affected by overfitting. Also, the small size of the validation set (only 26 image frames) could affect the choice of the final model since this decision is based on validation mAP only. Hence, there is a possibility that another model, in reality, performs better.

Another factor affecting the performance of the detection model is the varying appearance of a caudal fin. Its rotation relative to the camera determines its perceived shape, which can be interpreted as triangular (seen from the side), rectangular (seen from the front or back), or something in between depending on which side is exposed. The lighting conditions affect the color and exposure of the fins. This might be another reason for obtaining false positives. Since multiple moving fish are present in the detection volume, caudal fins are occasionally occluded, partially or totally. Partially occluded fins are more difficult to detect as their features will deviate from non-occluded fins.

The underwater imagery used for the experiment was governed by typical issues like reduced visibility, blur, low contrast, noise, and diminished color. This affected all parts of the process, and each method had to be tuned to cope with this. Visibility was limited because of light attenuation. This was a problem when training the detection model to recognize caudal fins in specific as they had a variety of appearances. For the depth estimation part, this lead to water incorrectly reconstructed as surfaces on one hand, and actual objects not being fully reconstructed on the other. Motion blur and low contrast reduced the detail level and thus affected the number of matched pixels.

The dataset was a part of the experimental pipeline and reflects natural challenges with underwater imaging. However, the system developed must be trained, tuned and verified on other datasets before it could be applied for practical purposes.

7.1.2 Annotation

Caudal fins were detectable in the area closest to the camera and the concentration of fins resulted in a certain degree of overlap and shadow zones. As fish (and thus, the caudal fins) get further away from the camera lens, they get smaller and more blurry, and there can possibly be a number of fins in an image frame without them being possible to distinguish from other objects. Also, at its smallest size, a caudal fin can appear to be only a few pixels in size. Labeling too small instances can increase the likelihood of the detector finding other, similar patterns in the image that are not caudal fins, especially in noisy images like in this case. Mistakenly labeled non-fins and non-labeled actual fins can possibly explain why the results were affected by such a large number of false positives and false negatives. The size of the fins was also challenging in terms of creating consistent and well-fitted bounding boxes. A deviation in the margin of the bounding boxes of only a few pixels will have a higher impact on small objects as the relative deviation will be much larger. In addition, fins appearing at multiple distances from the camera makes the annotation task more complex in terms of achieving well-fitted and consistent bounding boxes at all scales. Typically, the bounding box margin of small objects will turn out larger than for large objects.

7.2 Performance of Detection Model

The YOLOv4 detection model was running up to 52.3 FPS, which is more than the video speed (24 FPS) and more than sufficient for real-time performance. We must however take the hardware into consideration. The test was run on the GPU described in section 5.2, which is not state-of-the-art but can perform much faster computations than a CPU.

When looking at performance at different IOU thresholds the AP of YOLOv4 remains stable up to and around a threshold of 0.50. It is expected that AP will decrease as the threshold increase, but in this case, we see a drastic drop in the performance between 0.50 and 0.75 IOU. This might be a result of inadequately fitted bounding boxes, as described in section 7.1, causing the model to learn inconsistently. The model is not obtaining more FP and FN at IOU 0.25 than at 0.50. This indicates that most IOU scores are between 0.50 and 0.75.

When increasing the confidence score threshold keeping a fixed IOU threshold, the amount of detections (TP+FP) clearly declines, as expected. The drop in detections with increased confidence suggests that confidence scores are distributed on all confidence scores. Further, we see that the number of false detections is not far from the number of true detections at confidence threshold 0.10, meaning we obtain a too high object count. The number of FN is however relatively small at this threshold, so the model is able to detect most actual fins.

In Figure 6.3 we see that the model shows good detections across scales, but it would be advantageous to do further tests on where the visibility limit for annotating a fin should be. In the upper right part of Figure 6.2 a fin far away is detected with confidence 76%, while other fins at the same distance are not detected. This is a typical problem showing that one should be consistent in the annotations.

7.3 Stereo Matching and 3D Reconstruction

7.3.1 Camera Calibration

The initial total reprojection error was measured to 0.2517 pixels, which, isolated, is satisfactory. However, as shown in Figure 6.4, there was one significant outlier: image number eight from the right camera. This image pair and the three other image pairs with the highest reprojection errors were removed. The mean reprojection error after removal was 0.2234 pixels, as shown in Figure 6.5, which can be considered as highly satisfactory. The low reprojection error is probably a result of properly mounted cameras not moving significantly relative to each other, correct timing of imaging, and identical camera settings in the two cameras.

7.3.2 Stereo Matching Algorithm

The semi-global block matching algorithm performed reasonably well given the challenges of the underwater dataset. The algorithm achieves dense disparity maps and provides a

trade-off between performance and efficiency suitable for many practical cases. However, the depth estimation part was the greatest challenge of the entire system. It would be desirable to achieve depth estimations of larger areas of the images. Even though enough surfaces were properly matched for this proof-of-concept, a system for practical purposes would need more reliable results. A larger amount of successful depth estimations would give more datapoints to rely on, and thus contribute to more accurate swimming velocity estimates.

7.3.3 Disparity Post-Processing

The raw disparity maps were cleary affected by speckles (regions with large variance between disparities), seen as "holes" in the disparity maps in Figure 6.6c and Figure 6.7c. To a certain extent, the speckles were decreased by adjustment of the speckleWindowSize and speckleRange parameters of the stereo matching algorithm. By filtering the disparity maps the speckles were nearly eliminated, and the disparities turned out smoother. However, this introduced a new problem. As seen in Figure 6.6d and Figure 6.7d, not only the surfaces are smoother, but also the edges of the fish. E.g., this is prominent on the caudal fin of the middle fish in Figure 6.7. Figure 6.8 shows how the smooth edges appear as noise in the pointclouds. Additional filtering could might reduce this noise and make edges sharper. However, this noise was not considerable issue for this application, as we only look at a small area in the middle of the caudal fin surface determined by the detection bounding box. Naturally, this requires the assumption that the bounding boxes are reasonably accurate.

7.3.4 3D Reconstruction

Constructing pointclouds from the calculated disparity maps is a good way to validate the stereo matching. The calibration images of checkerboards were reconstructed in order to measure the square size of the checkerboard squares. Even though the checkerboard pointclouds are rarely reconstructed in full due to their non-structured and homogeneous surfaces, the corners are distinctive and the distance between them should correspond to the real square size. Table 6.3 shows the measured distances of some selected examples of checkerboard square sizes shown in Figure 6.9. They deviate from the ground truth in the order of 1-2 millimeters. As the square sizes were measured "by hand", a measurement error is introduced. As the squares were not fully reconstructed in the sense that there was a gradual transition between white and black squares, the results are approximate. However, they show that the distances of the 3D reconstruction are reasonable.

The 3D reconstructions directly reflect the goodness of the disparity estimations, but when

3D points are obtained the results can be associated to the real world. Measurements from the pointclouds showed reasonable salmon lengths. These are however difficult to validate as the ground truth and the age (and thus, the expected size) of the fish are unknown, meaning the measurements of the checkerboard squares are more dependable. Salmon body surfaces with a valid disparity/depth were mostly complete and uniform, as seen in Figure 6.10. The same figure also shows that only a small amount of the fish in an image actually achieve an entire surface of valid disparity values. Especially for fish close to the camera (meaning small depth). We attempted to solve this problem by tuning the disparity thresholds in the StereoSGBM algorithm (by increasing the numDisparities parameter), but this resulted in more noisy disparity maps and pointclouds. Also, the other previously smooth surfaces were exacerbated and the disparity maps were affected by holes (invalid disparities). It must be emphasized that the goal of the 3D reconstruction was never to achieve depth estimates of all fish in an image or a video sequence. Because of the nature of the underwater images, this was not realistic. The main priority was to achieve some satisfactory depth estimates that could serve as a part of the final pipeline. In that context, results as in Figure 6.10 were satisfactory.

7.4 Tracking Algorithm

The tracking algorithm was implemented to investigate if it was possible to use detection-based tracking on the results from the detection model, and if adequate results could be obtained to identify a salmon over several frames. A sufficient tracking result was to be used for estimating the salmon swimming velocity. As argued in section 5.5 the IOU tracking algorithm was chosen as a starting point because it was by far the simplest algorithm, and it required no image or motion information. Although the results are difficult to assess quantitatively as no formal method is used, there are several aspects to discuss.

7.4.1 Comparison of Algorithms

From here on, we will denote the original tracking algorithm as a_{orig} and the modified algorithm that allows gaps as a_{mod}. In the (informal) tests, a_{mod} performed better in terms of the lengths of the tracks. The maximum track length increased in both tests. The results are substantiated by the average track length which was larger with the modification in both tests. With gaps allowed, the number of tracks was also reduced. These results are probably obtained because tracks that were split due to gaps in a_{orig} were now merged. Hence, the modification indeed provided an improvement in the test cases.

7.4.2 Implementation and Tuning

The implementation and tuning of the tracking algorithm also need some comments. Since the first bounding box after a gap must overlap with the previous certain bounding box to recover the track, the possibility of a match will in general decrease for every frame of the gap. The confidence threshold σ_l was set to a medium/high value to decrease the number of false tracks, but a natural consequence is that actual detections (with small confidence) will be excluded and possibly split tracks. This is a trade-off that must be considered for each specific application if detection results are not flawless.

In Figure 6.11, two false positive bounding boxes create a false track, since they fulfill the IOU overlap minimum requirement. This could be avoided by choosing a higher confidence threshold and/or a higher IOU overlap threshold. Figure 6.12 shows how the original track with ID 806 is split due to overlap below the threshold, whereas the track reappears in the next frame with a new ID. A lower IOU overlap threshold would reduce this problem.

7.4.3 Influencing Factors

The results from tracking were highly impacted by the detection results and by the camera movements. Since the IOU tracking algorithm is highly dependent on the object location to be stable to get an IOU match, the maximum length of a track is directly limited by the time the camera is able to retain its position and orientation. In the short test video analysis, the a_{mod} algorithm was able to track sequences of about 3.8 to 5 seconds. This corresponds to the maximum time fish are present in the camera FoV, if we consider fish in the area where depth estimations were most reliable (i.e., in the "middle" of the camera FoV). This indicates that there is promise in using this simple tracking algorithm to gain insight about the behavior of the salmon, e.g., to find an average velocity vector over all tracks in an image sequence.

A benefit of the IOU tracking algorithm is that it requires no prior knowledge about image semantics or object motion. In our case this also introduces a considerable drawback. Since the IOU tracker is not able to predict or recognize non-detected positions of the caudal fins in case of gap or occlusion, tracks will be terminated if the next bounding box after gap/occlusion does not overlap with the previous one. On the other hand, it would be challenging to use an algorithm that requires a motion model, since the sensor setups vary and the fish behave differently from a camera point of view depending on what relative angle their motion has to the camera lens. Also, the movement of the camera during video recordings would be challenging to capture in a motion model.

7.5 Swimming Velocity Estimation

7.5.1 Detections and Disparities

As discussed in section 7.3, the depth estimations were of variable quality. This was the motivation behind filtering the depth estimations based on the variance of the disparity values. Figure 6.13 shows the combination of detections and depth estimations that are desirable, namely, tight bounding boxes and distinct and smooth disparities on the caudal fins. The lower variance threshold was introduced to filter out caudal fin detections with invalid disparities, e.g., detections outside the valid area as in Figure 6.14b, and incorrect disparity estimations as in Figure 6.14c and Figure 6.14d. Cases like Figure 6.14b were also discarded because of the low threshold. The disparity values are close to the maximum disparity over the entire fin. This might be a good detection, and one could argue that it should be included. This is a trade off encountered when determining the threshold value. However, looking closer at the magnitudes of the variances in Figure 6.14a and Figure 6.14d, an attempt to adjust the threshold to include the former would also include the latter. The high threshold was introduced to filter out non-smooth surfaces, like shown in Figure 6.15. As for the low threshold it is difficult to find the perfect value, filtering out all insufficient depth estimations and keeping all sufficient estimations. The final threshold values were found to be satisfactory for the test videos, but might need adjustments for other datasets or videos.

The size of the inner area used to determine the mean and the variance of the disparity values also needs some comments. Figure 5.14 shows the real ratio between the size of the bounding box and the inner area. There could exist cases where this exact part of the surface is smooth, but the disparity estimation in reality (and outside the inner area) is faulty. There could also exist false positive caudal fin detections where the disparity estimation inside the inner area is incidentally smooth (could be another type of object or just an incorrect disparity estimation). Increasing the inner area would likely reduce these problems. However, an increased area also increases the risk of including a larger area than the actual fin, including the "background" in the mean and variance calculations. This could exclude good fin surface estimates. In other words, we encounter an additional trade off choosing a good size for the inner area.

7.5.2 3D Tracks

Since we chose the caudal fin as an identifier and tracking object, their egomotion affected the depth estimates and thus the 3D position estimates. Overall, the 3D plots of the tracks corresponded to observations of the tracks in the original videos. Video 1 was

more influenced by the camera egomotion. From the observations, the distinct jumps in Figure 6.16b and Figure 6.17b seem to be mainly caused by large movements of the camera. Another factor is that the fish position relative to the camera in this video is such that the caudal fin egomotion also contributes to noticeable motion in the vertical and horizontal directions. The former directly influences the vertical positions of the tracks. The latter affects the vertical direction indirectly because the fin changes shape from the camera point of view, meaning the bounding boxes vary in size and thus the middle point used as tracking point moves relative to the real fin. In the second video, the egomotion of the camera is less significant. This corresponds to Figure 6.18b and Figure 6.19b, where the positions are more stable in the vertical direction. The depth calculations seen in Figure 6.16a, Figure 6.17a, Figure 6.18a, and Figure 6.19a suggest that the system tends to get the best (i.e., longest) tracks from fish not too far and not too close to the camera.

By tracking the fin motion, we wanted to assess whether the system could also be sufficient for more detailed analyses of the caudal fin egomotion. Studies of the depth of the tracks indicate that the system (including the dataset) might not be sensitive and accurate enough to capture this.

7.5.3 Mean Velocity and Standard Deviation

Measurements of mean velocity over all tracks in the test videos are shown in Table 6.5. According to Berge (2016), farmed salmon swim with a velocity of 0.7 bodylengths s^{-1} at daytime. Assuming a grown salmon length of around 0.8 m, the average swimming velocity should be around 0.56 ms^{-1}. This is a very rough approximation, but indicates that the measured mean velocities are within a reasonable range. The measured standard deviations show large variations in the mean velocities of the individual tracks. This is expected, as each individual method combined into the total pipeline introduce errors that affect the final velocity estimation. The maximum and minimum velocities substantiate this as they deviate extensively from the mean value. By reducing the error in each individual component, the total error could also be reduced.

7.5.4 Pipeline Implementation

The pipeline was implemented as an offline solution. The individual components are all able to run in real-time, but effort was not put in implementing an online pipeline because the work was conducted as a proof-of-concept. It is yet to be investigated whether the combined system can run in real-time. As a part of a larger system monitoring fish welfare, this can be an important factor. Performance of computer hardware and software is constantly improving, making the forecasts optimistic to be able to implement a real-

time system for estimating fish swimming velocity with the methods proposed in this thesis. Another comment is that the implementation is somewhat aggravating, requiring some manual actions in between the different components. With small changes the system will run as an automatic pipeline.

Chapter 8

Conclusions and Further Work

8.1 Conclusions

An experimental pipeline has been established from well-known computer-vision and deep-learning based methods for estimating farmed salmon swimming velocity and verified using industrial sea cage video. The work has involved studying fish welfare indicators, examining and implementing vision-based methods, and developing a system for estimation of salmon swimming velocity. Examinations of welfare indicators, the nature of the dataset, and considerations of relevant methods indicated that swimming velocity would be feasible to estimate by vision-based techniques. The results from image scene depth estimation and considerations regarding object detection and tracking indicated that tracking of fish caudal fins would be more feasible than tracking entire fish or other parts of the fish. The YOLOv4 detection model was used to detect caudal fins, semi-global block matching was used for 3D reconstruction, and a simple IOU tracking algorithm was applied to finally provide an experimental pipeline for estimation of swimming velocity.

The work presented serves as a contribution to solve the challenges of fish welfare in industrial fish farms by vision-based techniques. This was obtained by investigating how well-known methods could be tuned, combined and applied for this purpose. The results can provide useful insights to the limitations regarding today's data quality, sensors, and methods for object detection, 3D reconstruction, and object tracking, and provide suggestions to how these aspects can be improved on the path towards a more sustainable and autonomous fish farming industry.

The issues of underwater imagery affected all parts of the process, and each method had to be tuned to cope with this. Welfare indicators like skin patterns and gill rate were discarded directly because of reduced visibility and lack of details in the imagery. Swimming velocity, however, did not require specific details, just an object or point of

interest to track, located on the fish. Object detection was affected by the visibility decreasing with the distance from the camera, meaning at a certain point it is hard to distinguish caudal fins from other objects or the background. In low visibility conditions the annotation of the fins could not be performed due to missing definition of the imagery, which affected the performance of the object detector that was trained and tested on these annotations. Also for the depth estimation/3D reconstruction part, the same challenges were prominent. Not all fish were properly reconstructed and parts of the water was reconstructed incorrectly as surfaces. The former is typically a result of motion blur and the latter a result of turbidity in the water and attenuation.

Considering the trade-off between speed and accuracy, the performance results of the object detector showed that the speed was sufficient for real-time detection. Obtaining a mAP@0.50 of 87.79%, we show that it is possible to detect salmon caudal fins by applying off-the-shelf deep learning methods with minimal customization on industrial sea cage imagery.

The largest issues concerning depth estimation appeared to be non-fully reconstructed surfaces. The application of the WLS filter as a post-processing step for the raw disparity maps showed an improvement, and the extra noise of smoothed edges added by the filter was not a large concern for this application as we only use the surface of an inner area of the caudal fins. The missing fish surface reconstructions are mainly a consequence of the image quality. This was not crucial for a proof-of-concept, but for practical applications one might consider using higher-resolution cameras, different lighting sources, and additional measures to decrease the effect of attenuation and other issues caused by the nature of underwater imaging.

Camera movements and detection results are the main issues regarding tracking of the fins. The camera egomotion prevented consistency in the tracks, and gaps in the detection results induced problems with relating bounding boxes to each other in subsequent frames. From assessing the results from the tracking, the IOU tracking algorithm by itself performs sufficiently for practical purposes. Additionally, we can see a slight improvement when implementing a modification to allow gaps in detections between frames.

The performance of the final pipeline was directly affected by the performance in the individual methods. Filtering out insufficient disparity estimations worked as expected, and most erroneous disparities were discarded while keeping the majority of sufficient disparities. Comparison of plotted 3D tracks and observations of the original videos showed that the obtained tracks corresponded with the real motion of the fish relative to the camera. The egomotion of the caudal fin in the depth direction did not appear to affect the tracks directly, only indirectly by varying sizes of the bounding boxes in consecutive frames. This also means that the system with the current stereo matching technique will not be suitable for detailed analyses of caudal fin egomotion. The mean velocities com-

puted from the test videos are reasonable compared to the numbers one would expect for salmon swimming in sea cages at daytime. The standard deviations are of considerable size, which is expected considering the contributions of errors in the individual components. Since the fish in a school would be expected to have a uniform swimming velocity, this suggests the estimate is suitable for considering the overall swimming velocity at a certain time, but for detailed analyses it would be advantageous to reduce the standard deviation.

Keeping in mind that this study was carried out as a proof-of-concept to investigate the possibility of applying object detection and tracking to underwater image data and with limited resources, we have gained important insight about the limitations of underwater imaging with current sensors, which methods can be used for analyzing them, and found promise in experiments estimating swimming velocity with well-known vision-based methods. We have thus fulfilled the purpose. Object detection tests showed that sufficient results could be obtained for industrial purposes. We have witnessed that tracking of caudal fins in the current circumstances is feasible and can be achieved by applying simple methods, but also that depth estimation is challenging in underwater imagery. Computer vision and deep learning methods show promise for monitoring farmed salmon welfare, but they are directly limited by image quality and precision of camera calibration.

8.2 Further Work

The thesis covers a broad spectre of research fields, including fish welfare, underwater imaging, object detection and tracking, and stereo vision. As expected, several challenges have appeared through the work of analyzing underwater images. In the following we present some suggestions and recommendations for further work.

- Improve the depth estimation, which is the bottle neck of the system. One could improve the method suggested in this thesis by, e.g., adjusting the parameters or adding additional filters for post-processing. The recent advancements in deep-learning based methods for disparity estimation could also be explored.
- Adjust the pipeline to study the egomotion and frequency of the caudal fins. This will require improvements in the depth estimations as the current estimations are not significantly affected by fin egomotion.
- Investigate how camera egomotion can be modeled or detected to achieve swimming velocity estimations independent of camera motion.
- Consider tracking other parts of the fish, e.g., the eye or the entire fish. From the results in this proof-of-concept this was not feasible, but with better depth estimation this might give more accurate results as the eye is a smaller object and only moves with the fish's total movement.

- The tracking part may be improved by replacing the IOU tracking algorithm with a more sophisticated algorithm considering image information. Optical flow could be a feasible method.

Bibliography

Ahmed, M., Boudhir, A., Santos, D., Aroussi, M. & Karas, İ. (2020), *Innovations in Smart Cities Applications Edition 3: The Proceedings of the 4th International Conference on Smart City Applications*, Lecture Notes in Intelligent Transportation and Infrastructure, Springer International Publishing.

Berge, A. (2016), 'Laks kan svømme fra lakselus'. Available from: https://ilaks.no/laks-kan-svomme-fra-lakselus/. (Accessed: 2021-06-02).

Bochinski, E., Eiselein, V. & Sikora, T. (2017), High-Speed Tracking-by-Detection Without Using Image Information. doi: 10.1109/AVSS.2017.8078516.

Bochkovskiy, A. (2020*a*), 'AlexeyAB/darknet: YOLOv4 pre-release'. doi: 10.5281/zenodo.3829035.

Bochkovskiy, A. (2020*b*), 'How to use angle=... in YOLO .cfg files'. [GitHub issue]. Available from: https://github.com/AlexeyAB/darknet/issues/4626. (Accessed: 2020-12-01).

Bochkovskiy, A., Wang, C.-Y. & Liao, H.-Y. (2020), 'YOLOv4: Optimal Speed and Accuracy of Object Detection', *ArXiv* **abs/2004.10934**. Available from: https://arxiv.org/pdf/2004.10934.pdf. (Accessed: 2020-09-20).

Brown, M., Burschka, D. & Hager, G. (2003), 'Advances in Computational Stereo', *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **25**, 993– 1008. doi: 10.1109/TPAMI.2003.1217603.

Brownlee, J. (2019), 'A Gentle Introduction to Pooling Layers for Convolutional Neural Networks'. Available from: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/. (Accessed: 2020-11-14).

Chen, T., Kornblith, S., Norouzi, M. & Hinton, G. E. (2020), 'A Simple Framework for Contrastive Learning of Visual Representations', *CoRR* **abs/2002.05709**.

CNN Edge detection (2018). Available from: http://datahacker.rs/edge-detection/. (Accessed: 2020-11-13).

Cutting, J. (2000), 'Images, Imagination, and Movement: Pictorial Representations and Their Development in the Work of James Gibson', *Perception* **29**, 635–48. doi: 10.1068/p2976.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K. & Fei-Fei, L. (2009), ImageNet: A Large-Scale Hierarchical Image Database, *in* 'CVPR09'.

Dollár, P., Lin, T., Girshick, R. B., He, K., Hariharan, B. & Belongie, S. J. (2017), 'Feature Pyramid Networks for Object Detection', *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* pp. 936–944. doi: 10.1109/CVPR.2017.106.

Ernst & Young (2019), 'The Norwegian Aquaculture Analysis 2019'.

Everingham, M., Eslami, S., Van Gool, L., Williams, C., Winn, J. & Zisserman, A. (2014), 'The Pascal Visual Object Classes Challenge: A Retrospective', *International Journal of Computer Vision* **111**. doi: 10.1007/s11263-014-0733-5.

Fukushima, K. (1980), 'Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position', *Biological Cybernetics* **36**, 193–202.

Fulton, M., Hong, J., Islam, M. J. & Sattar, J. (2018), 'Robotic Detection of Marine Litter Using Deep Visual Detection Models', *CoRR* **abs/1804.01079**.

Funk, C., Bryant, S. B. & Heckman, P. (1972), Handbook of Underwater Imaging System Design.

Gençay, R. & Qi, M. (2001), 'Pricing and hedging derivative securities with neural networks: Bayesian regularization, early stopping, and bagging', *Neural Networks, IEEE Transactions on Neural Networks and Learning Systems* **12**, 726–734. doi: 10.1109/72.935086.

Girshick, R. B. (2015), 'Fast R-CNN', *CoRR* **abs/1504.08083**.

Girshick, R. B., Donahue, J., Darrell, T. & Malik, J. (2013), 'Rich feature hierarchies for accurate object detection and semantic segmentation', *CoRR* **abs/1311.2524**.

Gismervik, K., Tørud, B., Kristiansen, T. S., Osmundsen, T., Størkersen, K. V., Medaas, C., Lien, M. E. & Stien, L. H. (2020), 'Comparison of Norwegian health and welfare regulatory frameworks in salmon and chicken production', *Reviews in Aquaculture* **12**(4), 2396–2410. doi: 10.1111/raq.12440.

Goodfellow, I., Bengio, Y. & Courville, A. (2016), *Deep Learning*, MIT Press. Url: http://www.deeplearningbook.org.

Goyal, P., Lin, T., Girshick, R. B., He, K. & Dollár, P. (2017), 'Focal Loss for Dense Object Detection', *CoRR* **abs/1708.02002**.

Hämäläinen, T., Pimentel, A., Takala, J. & Vassiliadis, S. (2005), *Embedded Computer Systems: Architectures, Modeling, and Simulation*, Lecture Notes in Computer Science, Springer Berlin Heidelberg.

Han, F., Yao, J., Zhu, H. & Wang, C. (2020), 'Marine Organism Detection and Classification from Underwater Vision Based on the Deep CNN Method', *Mathematical Problems in Engineering* **2020**, 1–11.

Hartley, R. & Zisserman, A. (2003), *Multiple View Geometry in Computer Vision*, Cambridge books online, Cambridge University Press.

Haykin, S. (1998), *Neural Networks: A Comprehensive Foundation*, 2nd edn, Prentice Hall PTR, USA.

He, K., Gkioxari, G., Dollár, P. & Girshick, R. B. (2017), 'Mask R-CNN', *CoRR* **abs/1703.06870**.

Hirschmuller, H. (2008), 'Stereo Processing by Semiglobal Matching and Mutual Information', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **30**(2), 328–341. doi: 10.1109/TPAMI.2007.1166.

Huang, G., Liu, Z. & Weinberger, K. Q. (2017), 'Densely Connected Convolutional Networks', *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* . doi: 10.1109/CVPR.2017.243.

Hubel, D. H. & Wiesel, T. N. (1959), 'Receptive fields of single neurones in the cat's striate cortex', *The Journal of physiology* **148**, 574–591.

Krizhevsky, A., Sutskever, I. & Hinton, G. E. (2017), ImageNet classification with deep convolutional neural networks, *in* 'CACM'.

Kurimo, E., Kunttu, L., Nikkanen, J., Grén, J., Kunttu, I. & Laaksonen, J. (2009), The Effect of Motion Blur and Signal Noise on Image Quality in Low Light Imaging, pp. 81–90. doi: 10.1007/978-3-642-02230-2_9.

Lawrence, S. & Giles, C. (2000), Overfitting and neural networks: Conjugate gradient and backpropagation, Vol. 1, pp. 114–119. doi: 10.1109/IJCNN.2000.857823.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. & Jackel, L. D. (1989), 'Backpropagation Applied to Handwritten Zip Code Recognition', *Neural Computation* **1**(4), 541–551. doi: 10.1162/neco.1989.1.4.541.

Lecun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998), 'Gradient-based learning applied to document recognition', *Proceedings of the IEEE* **86**(11), 2278–2324. doi: 10.1109/5.726791.

Li, X., Liu, S., Gao, M., Cai, Y., Nian, R., Li, P., Yan, T. & Lendasse, A. (2018), 'Embedded Online Fish Detection and Tracking System via YOLOv3 and Parallel Correlation Filter'. doi: 10.1109/OCEANS.2018.8604658.

Lien, A. M., Schellewald, C., Stahl, A., Frank, K., Skøien, K. R. & Tjølsen, J. I. (2019), 'Determining spatial feed distribution in sea cage aquaculture using an aerial camera platform', *Aquacultural Engineering* **87**, 102018. doi: 10.1016/j.aquaeng.2019.102018.

Lin, T., Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., Perona, P., Ramanan, D., Dollár, P. & Zitnick, C. L. (2014), 'Microsoft COCO: Common Objects in Context', *CoRR* **abs/1405.0312**.

Liu, S., Qi, L., Qin, H., Shi, J. & Jia, J. (2018), 'Path Aggregation Network for Instance Segmentation', *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* pp. 8759–8768. doi: 10.1109/CVPR.2018.00913.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S. E., Fu, C. & Berg, A. C. (2015), 'SSD: Single Shot MultiBox Detector', *CoRR* **abs/1512.02325**.

Liu, Y. & Aggarwal, J. (2005), *Local and Global Stereo Methods*, pp. 297–308. doi: 10.1016/B978-012119792-6/50081-4.

Lodi Rizzini, D., Kallasi, F., Oleari, F. & Caselli, S. (2015), 'Investigation of Vision-based Underwater Object Detection with Multiple Datasets', *International Journal of Advanced Robotic Systems* **12**, 1–13. doi: 10.5772/60526.

Ludvigsen, M., Hewage, T. D. M., Klingan, K., Holven, E. B., Sture, & Mo-Bjørklund, T. (2020), *Lecture Notes TMR4120 Underwater Engineering*, 1 edn, NTNU - Department of Marine Technology.

Luo, W., Zhao, X. & Kim, T. (2014), 'Multiple Object Tracking: A Review', *CoRR* **abs/1409.7618**.

McCormick, C. (2014), 'Stereo Vision Tutorial - Part I'. Available from: http://mccormickml.com/2014/01/10/stereo-vision-tutorial-part-i/. (Accessed: 2021-04-11).

Min, D., Choi, S., Lu, J., Ham, B., Sohn, K. & Do, M. N. (2014), 'Fast global image smoothing based on weighted least squares', *Image Processing, IEEE Transactions on* **23**(12), 5638–5653.

Nielsen, M. (2015), *Neural Networks and Deep Learning*, Determination Press.

Nofima (2018), 'Welfare Indicators for farmed Atlantic salmon: tools for assessing fish welfare', Report.

Norwegian Seafood Federation (2011), 'Aquaculture in Norway', Report.

Norwegian Seafood Federation (2012), 'Seafood 2025', Report.

OpenCV (2021), 'cv::StereoSGBM Class Reference'. Available from: https://docs.opencv.org/4.5.2/d2/d85/classcv_1_1StereoSGBM.html. (Accessed: 2021-05-11).

Overton, K., Dempster, T., Oppedal, F., Kristiansen, T. S., Gismervik, K. & Stien, L. H. (2019), 'Salmon lice treatments and salmon mortality in Norwegian aquaculture: a review', *Reviews in Aquaculture* **11**(4), 1398–1417. doi: 10.1111/raq.12299.

Ozcakir, E. (2020), 'Camera Calibration with OpenCV'. Available from: https://medium.com/@elifozcakiir/camera-calibration-with-opencv-9fb104fdf879. (Accessed: 2021-06-02).

Padilla, R. (2019), 'Metrics for object detection'. doi: 10.5281/zenodo.2554189.

Praveen, S. (2019), *Efficient Depth Estimation Using Sparse Stereo-Vision with Other Perception Techniques*. doi: 10.5772/intechopen.86303.

Pérez, D., Ferrero, F. J., Alvarez, I., Valledor, M. & Campo, J. C. (2018), Automatic measurement of fish size using stereo vision, *in* '2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)', pp. 1–6. doi: 10.1109/I2MTC.2018.8409687.

Rawat, W. & Wang, Z. (2017), 'Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review', *Neural Computation* **29**, 1–98. doi: 10.1162/NECO_a_00990.

Redmon, J. (2013–2016), 'Darknet: Open Source Neural Networks in C'. Available from: http://pjreddie.com/darknet/. (Accessed: 2020-10-02).

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. (2016), You only look once: Unified, real-time object detection, *in* 'Proceedings of the IEEE conference on computer vision and pattern recognition', pp. 779–788. Available from: https://arxiv.org/pdf/1506.02640.pdf (Accessed: 2020-10-25).

Redmon, J. & Farhadi, A. (2016), 'YOLO9000: Better, Faster, Stronger'. Available from: https://arxiv.org/pdf/1612.08242.pdf. (Accessed: 2020-10-25).

Redmon, J. & Farhadi, A. (2018), 'YOLOv3: An Incremental Improvement', *ArXiv* **abs/1804.02767**. Available from: http://arxiv.org/abs/1804.02767. (Accessed: 2020-10-16).

Ren, S., He, K., Girshick, R. B. & Sun, J. (2015), 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks', *CoRR* **abs/1506.01497**.
URL: *http://arxiv.org/abs/1506.01497*

Schettini, R. & Corchs, S. (2010), 'Underwater Image Processing: State of the Art of Restoration and Image Enhancement Methods', *EURASIP Journal on Advances in Signal Processing* **2010**. doi: 10.1155/2010/746052.

Serna, C. & Ollero, A. (2001), 'A Stereo Vision System for the Estimation of Biomass in Fish Farms', *IFAC Proceedings Volumes* **34**(29), 185–191. doi: 10.1016/S1474-6670(17)32814-8.

Shantaiya, S., Verma, K. & Mehta, K. (2015), 'Multiple Object Tracking using Kalman Filter and Optical Flow', *European Journal of Advances in Engineering and Technology* **2**, 34–39.

Stien, L. H., Bratland, S., Austevoll, I., Oppedal, F. & Kristiansen, T. S. (2007), 'A video analysis procedure for assessing vertical fish distribution in aquaculture tanks', *Aquacultural Engineering* **37**(2), 115–124.

Tan, M., Pang, R. & Le, Q. V. (2019), 'EfficientDet: Scalable and Efficient Object Detection', *CoRR* **abs/1911.09070**.

The MathWorks, Inc. (n.d.), 'Stereo Disparity using Semi-Global Block Matching'. Available from: https://www.mathworks.com/help//visionhdl/ug/stereoscopic-disparity.html. (Accessed: 2021-06-10).

van den Boomgaard, R. (2017), 'Tracking motion features – optical flow', Lecture notes in *Image Processing and Computer Vision*. University of Amsterdam.

Wang, C.-Y., Liao, H., Yeh, I.-H., Wu, Y.-H., Chen, P.-Y. & Hsieh, J.-W. (2020), 'CSPNet: A New Backbone that can Enhance Learning Capability of CNN', *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)* pp. 1571–1580. doi: 10.1109/CVPRW50498.2020.00203.

Weng, J. J., Ahuja, N. & Huang, T. S. (1993), Learning recognition and segmentation of 3-D objects from 2-D images, *in* '1993 (4th) International Conference on Computer Vision', pp. 121–128. doi: 10.1109/ICCV.1993.378228.

Wu, B. (1992), 'An introduction to neural networks and their applications in manufacturing', *Journal of Intelligent Manufacturing* **3**(6), 391—403. doi: 10.1007/bf01473534.

Xu, W. & Matzner, S. (2018), 'Underwater Fish Detection using Deep Learning for Water Power Applications', *CoRR* **abs/1811.01494**.

Zheng, Z., Wang, P., Liu, W., Li, J., Ye, R. & Ren, D. (2019), 'Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression', *CoRR* **abs/1911.08287**.

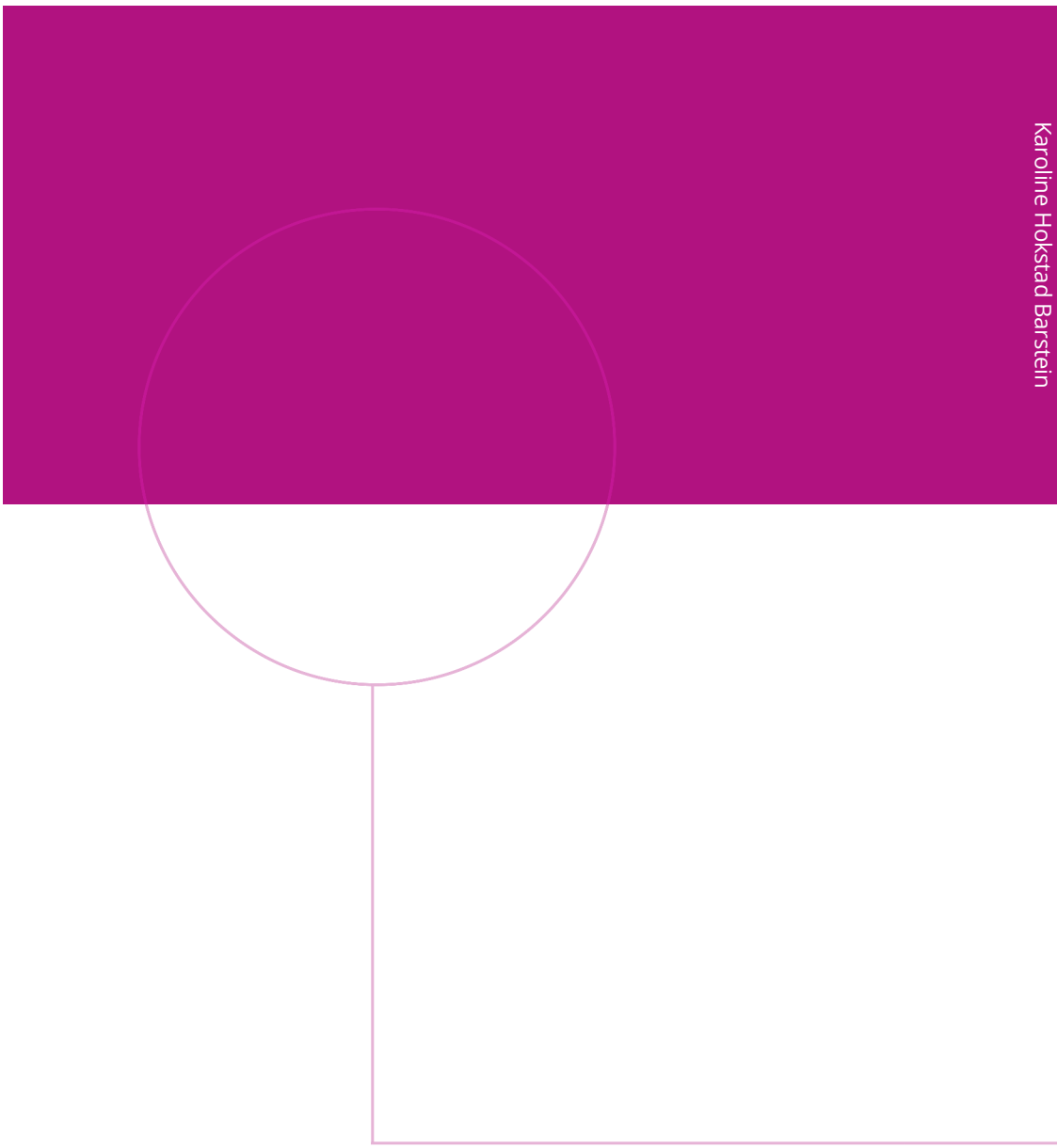
Ziyi, L., Xian, L., Liangzhong, F., Huanda, L., Li, L. & Ying, L. (2014), 'Measuring feeding activity of fish in RAS using computer vision', *Aquacultural Engineering* **60**. doi: 10.1016/j.aquaeng.2014.03.005.

Appendix

A Source Code

Source code for the object detection model, the depth estimation, the tracking algorithms, and the final pipeline can be found at:

https://github.com/karolhb/fish-detection-depth-tracking



Karoline Hokstad Barstein

NTNU
Norwegian University of
Science and Technology