Master's thesis

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

Kristoffer Chi Rong Jin

Deep-learning algorithms for estimation of fish-population parameters from video data

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
Co-supervisor: Eleni Kelasidi and Espen Eilertsen
June 2022

NTNU
Norwegian University of Science and Technology

Kristoffer Chi Rong Jin

Deep-learning algorithms for estimation of fish-population parameters from video data

Master's thesis in Cybernetics and Robotics
Supervisor: Annette Stahl
Co-supervisor: Eleni Kelasidi and Espen Eilertsen
June 2022

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

NTNU

Kunnskap for ei betre verd

Preface

This master thesis is written as the final part of the master's programme in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU) in collaboration with SINTEF Ocean. The thesis is a continuation of the pre-project submitted in the fall semester 2021 [1].

I want to thank my supervisor from NTNU, Annette Stahl and my co-supervisors from SINTEF Ocean, Eleni Kelasidi and Espen Eilertsen, for guiding me towards a solution to solve the problem at hand. I also want to thank the engineers at NTNU ITK lab who provided me with the necessary equipment to perform experiments in their lab. Furthermore, I am grateful that SINTEF Ocean let me capture image footage from live fish in their lab.

Kristoffer Chi Rong Jin
Trondheim, 9th June 2022

Abstract

This thesis presents a method to estimate the distance and speed of the fish by using two deep learning methods and a stereo camera in collaboration with SINTEF Ocean. Stereo images are captured by the camera and rectified to establish the epipolar geometry. YOLOv5, an object detection network, is applied to the images to detect the fish. This network is trained on two different datasets. After detecting the fish, a pretrained network, Superglue, which outperforms traditional matching methods such as SIFT and ORB, is utilized to establish point correspondences within the bounding boxes robustly. The point correspondences are used to reconstruct the 3D world point by triangulation. The 3D points are used to compute the Euclidean distance and speed.

An experimental test is created in a controlled environment to validate the distance measurement. It is seen from the validation that the method produces good results. After validating the distance measurements, the method is tested with live fish in a more realistic environment. It shows promising results on fish that swims alone, despite no ground truth available to measure the performance.

This thesis also conducts a literature review focusing visual tracking. A tracking method should be incorporated into the suggested method to track the fish over an image sequence robustly.

Sammendrag

Denne hovedoppgaven presenterer en metode for å estimere distansen og hastigheten til fisk ved å bruke to dyplæring metoder og et stereoskopisk kamera i samarbeid med SINTEF Ocean. Stereobilder er fanget av kameraet og rettet opp for å etablere epipolar geometri. YOLOv5, et objekt deteksjon nettverk, er brukt på bildene for å detektere fisk. Dette nettverket er trent på to forskjellige datasett. Etter å ha detektert fisken blir et forhåndstrent nerual nettverk, Superglue, tatt i bruk for å etablere punktkorrespondanser innenfor avgrensningsboksene på en en robust måte. Superglue er tatt ibruk ettersom den har vist seg å utkonkurerer tradisjonelle metoder som SIFT og ORB. Punktkorrespondansene er brukt til å rekonstruere 3D punktene ved hjelp av triangulering. 3D punktene er brukt for å regne ut den euklidiske distansen og hastigheten.

Et eksperiment er gjennomført i et kontrollert miljø for å validere distanse målingen. I valideringen er det observert at metoden produserer gode resultater. Etter valideringen er metoden testet i et mer realistisk miljø med ekte fisk. Den viser lovende resultater på fisk som svømmer alene, selv om det ikke er noe referansemåling tilgjengelig for å måle ytelsen.

Denne hovedoppgaven gjennomfører også et literaturstudie med fokus på visuell sporing. En sporingsmetode burde bli innarbeidet i den foreslåtte metoden for å spore fisken over en bildesekvens på en robust måte.

CONTENTS

CHAPTER 1
INTRODUCTION

This chapter starts to introduce the motivation behind this thesis. After the motivation is presented, the main objectives for the thesis are listed. The final section in this chapter shows how this thesis is structured.

1.1 Motivation

This master's thesis was done in cooperation with SINTEF Ocean, a department in the Norwegian research institute. SINTEF Ocean conducts research and innovation related to ocean space for national and international industries [1]. The work of this thesis is a contribution to the RACE Fish-Machine Interaction project. RACE is a project to shift production methods from manual operations to a more automated approach using smart sensors in combination with mathematical models and autonomous systems [2]. However, immature technology can lead to fatal consequences, loss and damage in fish farms, which are unacceptable when interacting with live fish. Hence, intelligent solutions like this must be researched and tested properly before being used in production. Figure 1.1.1 shows an illustration of the goal of the project where intelligent systems can operate in the fish farms.

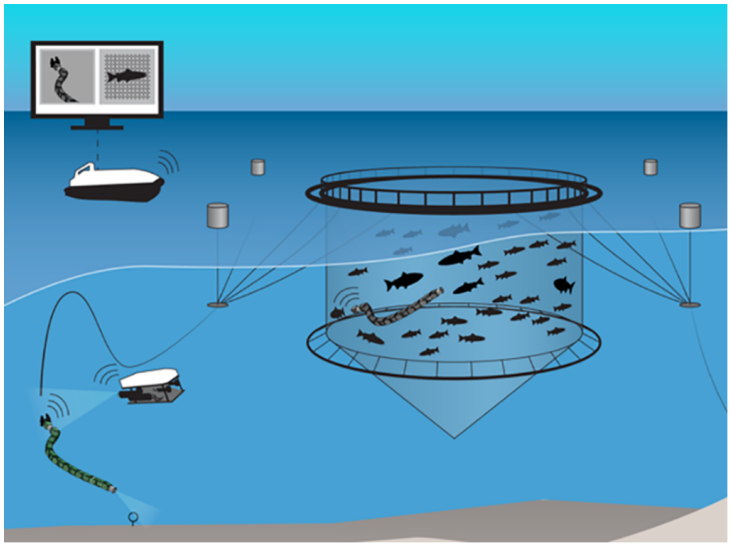


Figure 1.1.1: An illustration of the goal of the RACE project. It suggests how the fish-machine interaction can look like [2].

[1] SINTEF Ocean website

Aquaculture is an essential contributor to the production of seafood for human consumption. In 2016, a record high of 171 million tons of fish was output by the global fishery, where 88 % were consumed directly by people [3]. These numbers make aquaculture extremely important for building a world free from hunger. However, with an increasing population, the demand from the world fisheries will continue to increase [3].

Current fish farms are primarily driven by manual operations and close human interactions with the process and fish cage structures. It may be problematic in situations like exposed areas where the environment can be challenging. Hence, it is essential to explore new technology that can contribute to reducing the need for human interactions [4].

Researchers have proposed a concept called Precision Fish Farming (PFF). The concept aims to improve the farmer's ability to monitor, control and document biological processes in fish farms [5]. By realizing this concept in fish farms, they can become more automated. Before PFF can be achieved, two core areas need to be addressed.

The first area is safety and risk management. Current fish farms are operating at the safety limit with the available technology and management systems [6]. Difficult weather conditions, sub-optimal human-technology interaction, and many organizational aspects are known risks. These factors contribute to a greater demand for better operator skills and the ability to take necessary actions to avoid equipment failure, fish escaping or occupational accidents.

The second area is fish behaviour and welfare. Fish welfare is essential in any aquaculture operation and is the primary focus of the RACE project. To perform operations such as feeding, de-lousing and harvesting/sampling in a way that prioritizes fish welfare, it is necessary to both detect and understand the different behaviours of fish and how they correlate with welfare [7] [4].

Researchers have suggested solutions that incorporate sensor technology, such as Doppler velocity log [8] or laser [9] into remotely operated underwater vehicles. However, these solutions focus on finding holes in the fish cage nets. According to [10], the swim speed of the fish can be connected to multiple welfare issues. Currently, there are no tools available which can measure fish welfare. Cameras already exist in fish cages and are relatively cheap, which can be further developed to incorporate computer vision methods directly to the video stream [5]. Hence, this thesis aims to research a tool to measure the fish welfare by estimating the distance and speed of the fish by using a stereo camera, which incorporates deep learning and computer vision methods.

1.2 Objectives

RACE Fish-Machine Interaction project has the primary object to find methods to identify changes in behaviour of the fish by using smart sensors and to develop new methods for modelling and control for underwater vehicles, which allows autonomous operations in fish farms by also considering interactions of live fish with the used technology. By achieving this, the overall impact on the fish during complex operations can be greatly reduced. This thesis aims to experiment the use of camera sensors and deep learning algorithms to estimate fish behaviour changes. A solution like this can be incorporated in ROVs to according to the measurements provided by the cameras. As this thesis is an continuation of [1], the focus will be on the following objectives:

1. Perform a literature review of tracking method and the possibility to incorporate such solutions into a stereo system
2. Prepare an experimental setup to validate the methods
3. Compute the distance and velocity by using deep learning and computer vision methods on stereo images
4. Validate the accuracy of the proposed method with ground truth data
5. Test the suggested method on live fish data

1.3 Thesis outline

This thesis consists of 8 chapters. Chapter 2 conducts a literature review on tracking methods using camera sensors. Chapter 3 presents general theory about photogrammetry, such as the camera model, camera calibration and epipolar geometry. Chapter 4 shows the procedure behind the data acquisition and the relevant camera specifications. Chapter 5 continues this by presenting the calibration results gathered from the selected environments. Chapter 6 presents deep learning theory related to neural networks along with the deep learning methods that are used in the methodology. The chapter also presents the preparation of training data for the deep learning method. Chapter 7 presents the proposed method to solve this task at hand by showing results from applying the method on the gathered data. Chapter 8 concludes this thesis and suggests future work.

CHAPTER 2
LITERATURE REVIEW

This chapter presents a literature review on methods related to visual tracking. It starts with a short introduction to visual tracking and the challenges in visual tracking. After the introduction, methods that use optical flow for tracking are presented, followed by approaches that utilize indirect methods for tracking. Finally, tracking methods which use deep learning are presented, and then the chapter is concluded with a section which summarizes the advantages and disadvantages behind the three techniques, which should be considered when choosing a tracking method.

2.1 Visual tracking

Object tracking is an essential task within computer vision. It enables video cameras to capture moving objects by tracking them from frame to frame. Tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene in the image plane [11]. It can prove helpful when calculating the speed of objects with video data [12, 13]. Depending on the tracking domain, trackers can provide object-centric information, such as an object's orientation, area, or shape. Object tracking can be a challenging task due to:

- Loss of information
- Noise in the images
- Complex object motion
- Nonrigid or articulated nature of objects
- Occlusion
- Complex object shape
- Illumination changes [14]

2.1.1 Optical flow

A common way to determine the motion between a sequence of images is to compute an independent estimate at each pixel, often called the optical method [15, p.578]. The motion can also be referred to as the displacements of intensity patterns [16]. Optical flow techniques typically consider two constraints for motion estimation, data conservation and spatial coherence. Data conservation is derived from observing that objects in the scene usually persist in time. Hence, the pixel intensity of a small area in two consecutive images remains the same, even though its position changes.

Other successful methods within optical flow were introduced as global and local gradient-based approaches. These can also be categorized as sparse and dense methods. Global methods are focused on calculating the dense vector fields, while local methods calculate the sparse vector fields [17]. One of the first successful methods solved the optical flow problem by introducing a global constraint that applies a soft spatial coherence which forces the partial derivatives of neighbouring motion vectors to be minimal [18]. The method from [19] proposed to use a strong spatial coherence instead, which is a local constraint that expects the motion in a small region to be constant. However, these methods are based on assumptions which may be violated in real-world applications. According to [20], violations of these constraints lead to gross measurement errors that are known as outliers. The two previous methods penalize the minimization in a quadratic way and cannot handle outliers robustly. To robustly handle outliers, [21] proposed a robust estimation framework exploiting the Lorentzian robust norm.

Most state-of-the-art global methods to solve the optical problem utilize robust estimation frameworks. Compared to global methods, most local methods are based on least square optimization. These methods have the advantage of being excellent in terms of runtime. A method proposed by [22] builds on the Lucas-Kanade method [23] and increases its robustness by using the Hampel norm. This extension prevents violations of the constraints established in the optical flow problem mentioned previously and provides excellent feature tracking performance while slightly increasing computational complexity compared to Lucas-Kanade. However, this method proves to be inefficient in illumination changes and/or larger motions. Hence, [24] builds upon these limitations and exploits an illumination model to deal with varying illumination and a prediction step based on a perspective global motion model that increases the robustness of long-range motions. The performance of the method is compared against [22] and pyramidal Lucas Kanade [25], an overall improved version of [23] on popular benchmarking datasets like KITTI [26], Middlebury [27] and SIntel [28]. It is also compared against itself without the global motion model to demonstrate the impact of the global motion model. The former provides excellent results compared to the other methods. Although the method is a local method, it shows excellent results in calculating sparse motion fields but also competes with the global methods when calculating dense motion fields. OpenCV [1], an open-source library for computer vision, has implemented this optical flow method in their database, which enables the possibility to implement this in the programming languages Python or C++. Figure 2.1.1 shows an example of the optical flow with and without the illumination model.

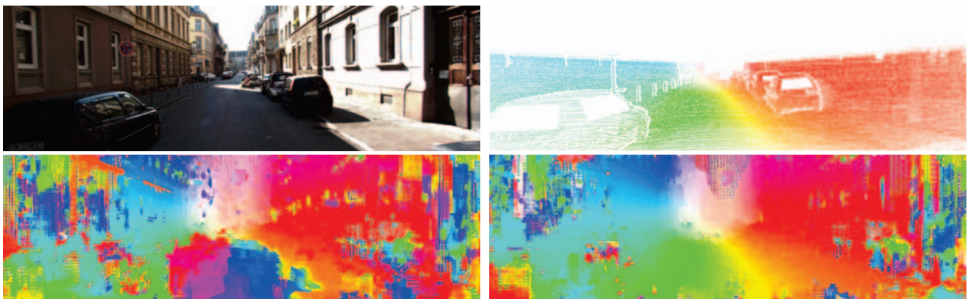


Figure 2.1.1: The two top images is the raw image and the ground truth. While the bottom left and right are the motion field without and with the illumination model [24].

The method presented in [29] incorporates stereo vision with an extended version of Lucas-Kanade to establish 3D trajectories. The classical Lucas-Kanade method is extended by integrating geometric constraints with epipolar constraint, which enforces tracked patches to remain on the epipolar lines, and magnification constrain, which links the disparity of the tracked patches to the apparent size of these patches.

A popular method specifically made for dense motion fields is the method presented by [30]. First, it approximates each neighbourhood of two frames by quadratic polynomials. This is performed efficiently using the polynomial expansion transform. Furthermore, a method to estimate displacement fields is derived and results in a robust algorithm for dense motion fields. It provides significantly better results compared to

[1] OpenCV GitHub

the other methods. However, the main weakness of the algorithm is the slowly varying displacement field, causing discontinues to be smoothed out. This can be improved by combining the algorithm with a simultaneous segmentation procedure [31]. The method by [32] also suggests a dense optical flow method by using dense inverse search. It consists of three parts. First, perform an inverse search for patch correspondences. Secondly, a dense displacement field is established through patch aggregation along with multiple scales, and then a variational refinement is performed. The approach is inspired by inverse compositional image alignment [33] and provides the same accuracy as other state-of-the-art methods, but the inference speed is significantly higher. Another advantage is that OpenCV has incorporated this in their code base, making it possible to implement it in programming languages like C++ or Python. However, this method only works for 2D trajectories, which is unsuitable for a stereo camera. It needs to be extended in the same way as in [29] to be able to establish 3D trajectories.

After the establishment of Flownet [34], deep learning approaches for optical flow became more popular. Flownet consists of two architectural lines, FlowNetS and FlowNetC, which operate like an hourglass-shaped neural network architecture that consists of an encoder and decoder, where the most significant difference is in the encoder part. However, issues with small displacements and noisy artefacts in estimating flow fields motivated [35] to develop Flownet 2.0, which inherited the advantages of the predecessor while improving the limitations with small displacements and noisy artefacts. These improvements gave a significant performance advantage on real-world applications and provided state-of-the-art results.

2.1.2 Indirect methods

Another approach to track objects in a sequence of images is to find point correspondences between the images where the motion between the frames is expected to be small [15, p.452]. The process of establishing point correspondences can be divided into three stages. The first stage establishes the interest points at distinctive locations in the image, such as corners, blobs and T-junctions. In this case, the most valuable information is the repeatability of an interest point. The repeatability describes the reliability of a detector in finding the identical interest points from different viewing perspectives. In the next stage, the neighbourhood of each interest point is expressed by a feature vector. This descriptor must be distinctive and robust to variations in the images. At last, the descriptor vectors are matched between a sequence of images. The matching process is based on the distance between the vectors. Popular examples of these kinds of feature detectors and descriptors are Scale-Invariant Feature Transform (SIFT) [36], Speeded-Up Robust Features (SURF) [37] and oFAST and rBRIEF (ORB) [38].

A method proposed by [39], object tracking is performed by using SIFT to identify matching features between consecutive frames in combination with particle filtering. The purpose of particle filtering is to combine particles at a particular position into an individual particle. A weight is given to the particle, reflecting the number of particles required to form it. The method is tested on surveillance cameras and outperforms the other tested methods, which are found in [39]. However, it suffers from occlusion and great illumination changes, which may be solved by using a multi-view camera setup. A multi-view camera will also enable the possibility of 3D tracking. In [40], a similar approach is also suggested. Instead of using SIFT feature matching, the latter use SURF to find matching features between a sequence of images. The method performs a performance comparison with other methods where SIFT is one of the best competitors. SURF outperforms SIFT on bigger objects, while on smaller objects, the performance is similar for all compared methods due to lack of feature points. The computational time with SURF is about 40ms less than SIFT. The speed makes SURF slightly better than SIFT for real-time applications in terms of speed and accuracy. Figure 2.1.2 shows an example of their tracking method deployed on a sequence of images.

Figure 2.1.2: An example of the tracker from [40] deployed on a sequence of images.

The method proposed by [41] uses SIFT features for multiple vehicle detection and tracking. After finding the SIFT features, matching is conducted, and RANSAC is used to remove outlier matches. This procedure provides an image alignment to search for the region with the highest sum of absolute differences. By following this process for a sequence of images, moving vehicles are successfully tracked with the proposed method. It proves to be significantly faster than the other compared methods [42] in the paper. The method provides robustness in traffic scenes with variations in illumination, shadows and camera dithering. The runtime is the only comparison between the methods, making it suitable for real-time applications. However, there is no evaluation of accuracy between the methods.

There are also methods like [43, 44] that combine both feature matching and optical flow to improve the estimation of dense trajectories. In [43], a combination of feature matching and optical flow makes it possible to add the camera motion into account. The suggested method improves the performance of the trajectory estimation as the camera motion can correct them. The camera motion is calculated by first extracting feature points between consecutive frames with SURF descriptors and optical flow. After matching, RANSAC is used to remove outliers. The proposed method provides excellent results. The approach from [44] uses FAST algorithm to detect corners as the initial position, then matches this with the next frame. After achieving this information, pyramidal Lucas Kanade is exploited to estimate the optical flow.

2.2 Machine learning-based methods

The last approach for visual tracking is the use of machine learning. There are different ways of utilizing machine learning to track objects of interest. Some deploy deep learning in the detection part, and others use deep learning for feature extraction and motion prediction. These subjects will be presented further in this section.

2.2.1 Deep learning for detection

A method suggested by [45] presents a method for deepwater animals where they use machine learning in the detection part. It can be categorized as an algorithm that performs tracking-by-detection. By using RetinaNet [46] to detect objects on both cameras, bounding boxes are drawn around objects of interest. A correlation between the bounding boxes in both cameras is found by calculating the intersection over union (IoU), which represents the overlapping area between the bounding boxes. The higher IoU score, the higher likelihood that a bounding box pair have detected the same object. The score is an input to a 3D tracker, consisting of an unscented Kalman Filter and the Hungarian algorithm and can determine the 3D positions of the detected objects. This information is shared with a supervisor module which determines whether the detected objects match the target class of interest and modifies the vehicle behaviour. The overview of the method is shown in figure 2.2.1 The presented method is promising since it is already deployed in an underwater environment and utilizes a stereo setup to construct 3D positions of detected objects. In addition, behaviour control is also implemented, which is relevant for the RACE project. However, the method does not show results with 3D information and its accuracy. Another method that uses IoU is [47]. It operates with high speed and competitive accuracy results. The method uses Mask R-CNN [48] and CompACT [49] as detectors to perform IoU tracking. However, a problem with regular IoU tracking systems is the number

of track fragmentations and ID switches. Hence, the method proposes to perform single-object tracking when no object detection is available, which provides more robust tracking while maintaining a high speed.

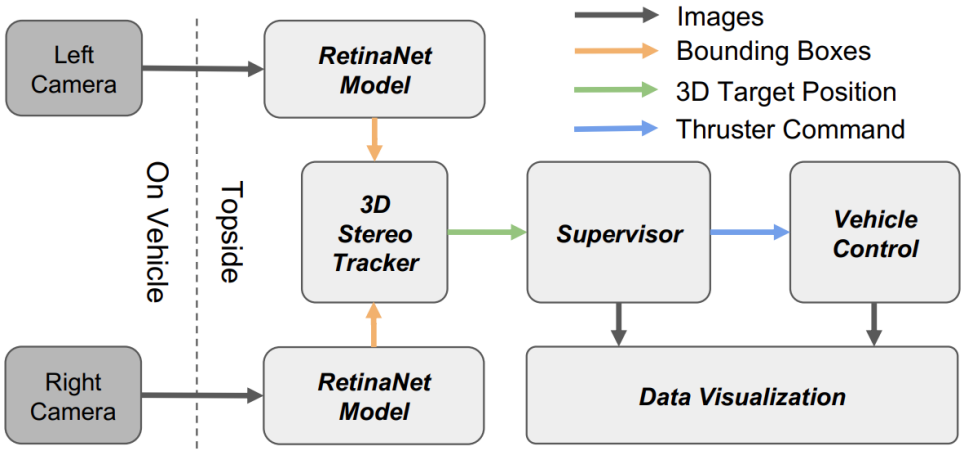


Figure 2.2.1: The flowchart of the proposed method in [45].

The popular method Simple Online and Realtime Tracking (SORT) suggested by [50] exploits the tracking-by-detection principle in combination with Kalman Filter and the Hungarian algorithm. It provides significantly faster performance compared to state-of-the-art trackers. The key factor in the method is detection quality, which can substantially increase performance accuracy. The suggested approach gives the user freedom to choose a detection algorithm suitable for the specific application. An example of this is shown in [51] where SORT is deployed with YOLOv2 to track and estimate the vehicle speed from monocular camera footage.

Convolutional neural networks (CNNs) are widely used for feature extraction. CNNs are based on subtle modifications of these networks. An example is a further improvement of the previously mentioned SORT algorithm. In [52], they extend the SORT algorithm by incorporating a custom residual CNN, which extracts appearance information. By adding appearance information to the algorithm, SORT can track objects for longer periods of occlusions and results in a reduced number of identity changes. Identity changes happen when the objects get occluded for a short time and then reappear as a new object for the model. The extension provides a significantly higher reduction in identity switches but adds an extra layer of complexity. Both tracking methods provide competitive results for real-time applications and may be possible to use in different environments. However, these are built around the benchmark dataset MOT16 [53]. Hence using this algorithm on custom datasets can be time consuming as datasets with the same format need to be created. Figure 2.2.2 shows an example of [52] deployed on the MOT16 dataset.

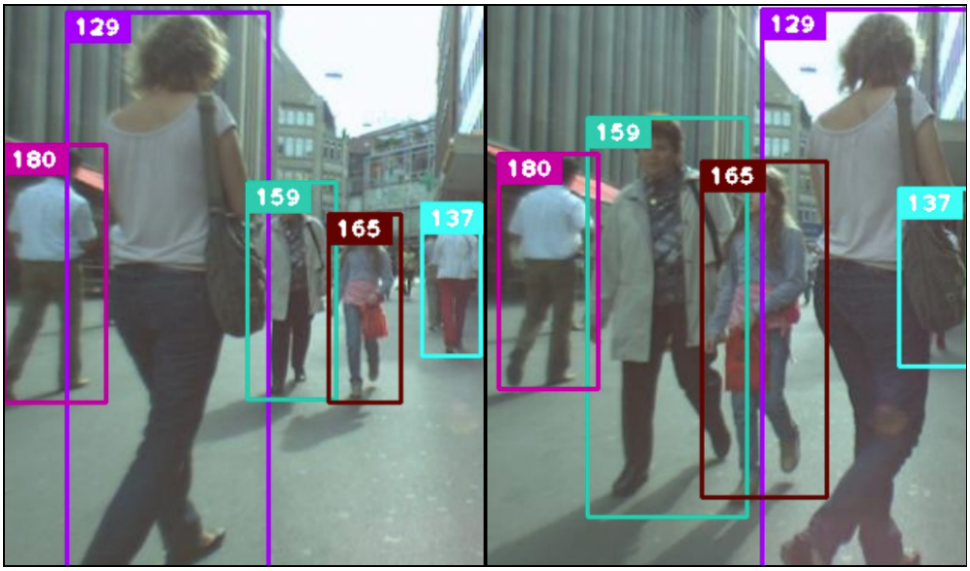


Figure 2.2.2: A sample of [52] on the MOT16 dataset. It provides a bounding box around each object of interest attached with an ID.

2.2.2 Deep learning for feature extraction and motion prediction

Another approach to incorporating deep learning into their framework is using Siamese networks. These are CNNs trained with loss functions that combine information from different images to learn the features that best differentiate examples from various objects. A simple example of a Siamese network is illustrated in figure 2.2.3

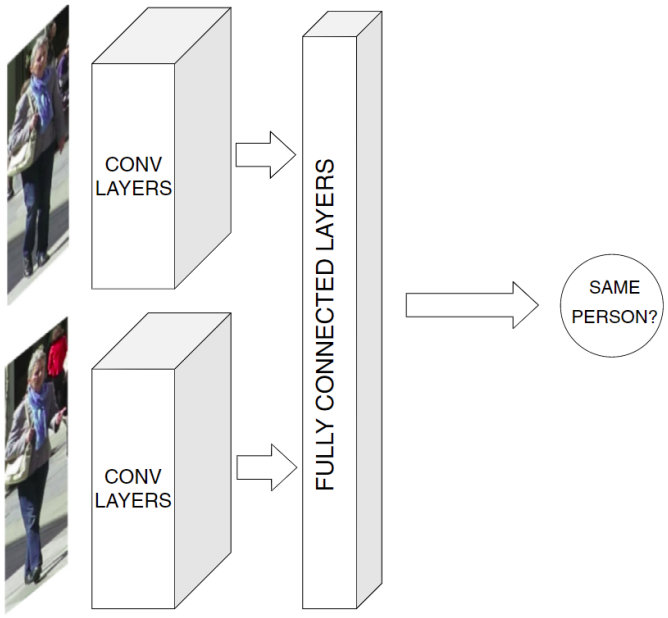


Figure 2.2.3: An illustration of how a siamese network works [54].

The presented method from [55] is based on similarity learning. It learns a function that compares an exemplar image to a candidate image of the same size and returns a high score if the two images represent the same object and a low score otherwise. By utilizing a fully-convolutional network [56], instead of a candidate image of the same size, a candidate image with a higher image can be used as input. This approach allows the algorithm to use a bigger image as a search area and computes the similarity for a more extensive

area. The network is tested against various methods with a competitive accuracy and significantly higher speed. Inspired by the former, [57] extended the method by adding a Region Proposal Network (RPN). This extension includes a regression and classification branch that significantly improves the method. It outperforms the state-of-the-art trackers in VOT2015 [58] in every category. Figure 2.2.4 shows a test of the tracker on samples from a benchmarking dataset.

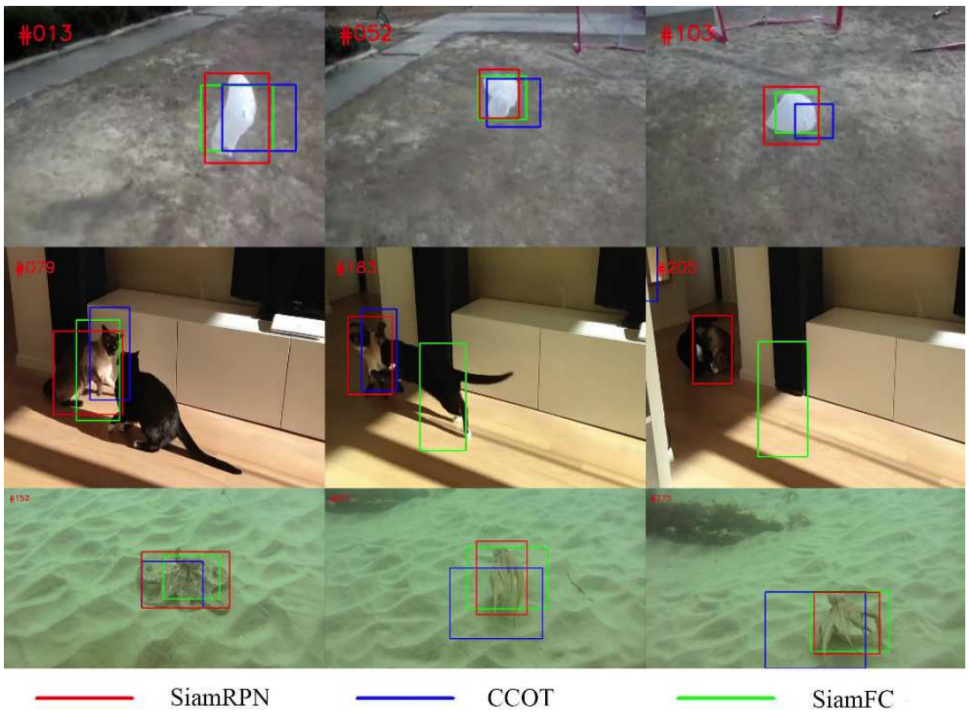


Figure 2.2.4: The proposed method SiamRPN compared against two other approaches [57]

Despite the significant performance of Siamese-based trackers, they underperform during complicated situations, especially when there are distractors. An attempt to solve this issue is suggested by [59]. By adding two efficient modules, a relation detector (RD) and a refinement module (RM), a significant improvement is obtained. The RD performs in a meta learning way to obtain a learning ability to filter distractors from the scene. The RM integrates the RD into the Siamese framework efficiently to generate accurate tracking results. To improve the robustness of the tracker, a training strategy that also learns to distinguish different objects is introduced. The proposed method provides state-of-the-art performance on the benchmarking dataset VOT 2017 [60] and outperforms other state-of-the-art methods.

Siamese networks also suffer from significant limitations. The first issue is that Siamese trackers utilize the target appearance when inferring the model. This approach completely ignores the background appearance information, which is vital for separating the target from similar objects in the scene. An example of this is shown in figure 2.2.5. The second problem is that the similarity measure is not always reliable for objects that are excluded from the training set, resulting in poor generalization. The last thing to mention is the Siamese formulation that does not provide a good model update strategy. Hence, state-of-the-art methods exploit simple template averaging. These limitations are addressed by [61], which introduces an alternative tracking architecture. The method is inspired by discriminative online learning procedures that use a discriminative learning loss by applying an iterative optimization process.

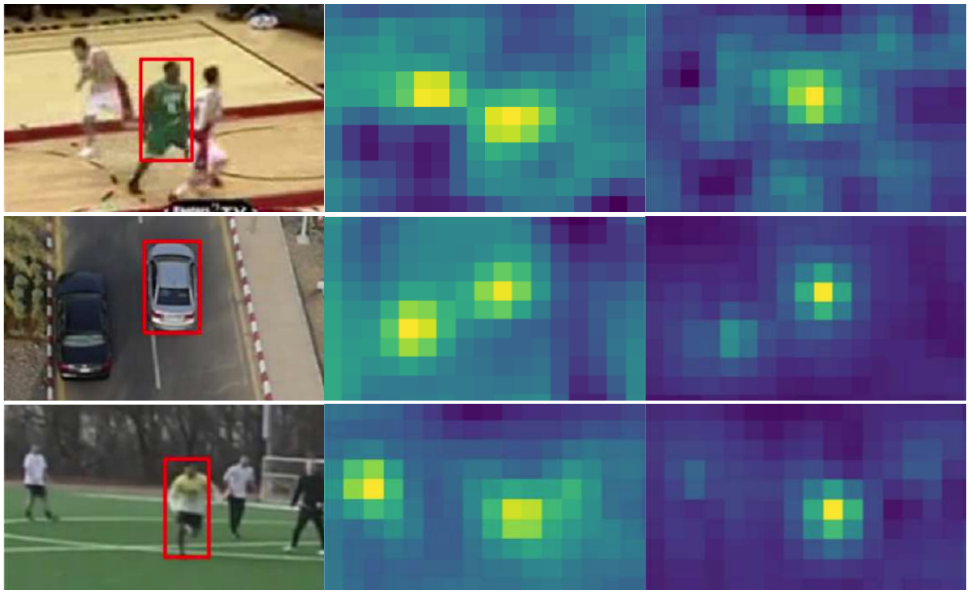


Figure 2.2.5: The middle and right image shows confidence maps of the target object shown in the left image with red box. The red box represent the target model. In the middle image a Siamese approach is used, while the right image shows the suggested approach [61].

Some methods utilize stereo vision in addition to deep learning to track objects. By using a stereo setup, 3D information such as distance and speed can be extracted directly from the objects [62]. The paper [63] proposes a method that performs 3D box inference by using 2D object detection and viewpoint classification that can provide object reprojection contour and occlusion mask for feature extraction. The feature extraction is achieved using ORB and combines this with an epipolar line search for stereo matching. This process serves as the semantic measurement model for the bundle adjustment. The bundle adjustment approach contributes to tightly coupling the semantic and feature measurements to track the objects continuously. Since the method computes the camera pose, it is robust in dynamic environments. It outperforms the state-of-the-art method ORB combined with simultaneous localization and mapping, also called ORB-SLAM2, while providing 3D information such as speed and distance of objects. It also shows competitive results compared to the method suggested by [64]. These advantages make it a good candidate for solving the problems in this thesis. The code is not open-source, which means it can be time-consuming to implement manually and performs tests on the KITTI dataset. The dataset requires a specific setup of the data, such that it can be challenging to obtain the data required to use this suggested method.

The method proposed by [65] combines 2D object detections and stereo depth measurements to improve image-based tracking with a precise 3D localization. The object detections provide object class information, while the stereo depth contributes to locating objects in world coordinates. This information is fed to a Kalman filter which keeps an image and a world space estimate. These estimates are coupled to secure the consistency of a track. This coupling makes it possible to track distant objects and continue these tracks with more precise information in the close range with a smooth transition between the modalities. The method is tested against the KITTI dataset and provides competitive results. It is constructed in a way which makes it easily expandable for other features. This method seems to be a possible candidate as a stereo tracking method but is limited to the KITTI dataset, similar to the previous method.

Another contribution to this task is [66] which performs a fusion of different inputs to perform stereo tracking. Their method, MOTSFusion is a two-stage algorithm and begins with creating short tracklets by utilizing the 2D motion consistency of segmentation masks under an optical flow wrap, then fuses these tracklets using depth and ego-motion estimates into dynamic 3D object reconstructions. After performing these operations, the transformation required for these reconstructions is then used to estimate the 3D motion of the object tracklets. They are merged into longer tracks if they undergo consistent 3D motion and use the extrapolated positions to fill in missing detections. The proposed method outperforms previous

state-of-the-art methods and provides tracking and depth information. Because of this, the method can be extended to provide the speed of each tracked object. The downside of this method is that it is built around the KITTI dataset, just like the previous methods.

2.3 Selecting the best tracking approach

Three different approaches for tracking have now been reviewed. The possible methods to solve the tracking problem are the stereo tracking methods. They have incorporated depth, object detection and tracking together. However, they require a lot of data to give good results. Optical flow is widely used for motion estimation and can be incorporated with object detection methods. Dense and sparse methods are suitable for estimating the motion field between a sequence of images. It is tightly connected to the indirect methods since they are often used to establish initial displacements or object detectors. However, optical flow is 2D-based, which may not work as intended in a stereo setup. An alternative to feature matching is the use of deep-learning detectors. The deep learning methods are currently state-of-the-art but require a high quantity of labelled data to reach the best performance. However, the issues mentioned in section 2.1 can result in bad tracking results. Many of the reviewed methods are applied to cars and may have issues translating to an underwater environment. It would be interesting to see how many of these methods work underwater.

CHAPTER 3

PHOTOGRAMMETRY

Photogrammetry is defined as the art, science and technology of obtaining reliable information about physical objects and the environment through processes of recording, measuring, and interpreting photographic images and patterns of recorded radiant electromagnetic energy and other phenomena [67]. The purpose of image capturing is to project 3D world data into the 2D image plane. In other words, one dimension of information is lost during the capturing. This dimension describes the depth of the scene. The lost dimension can be reconstructed using triangulation, which is explained later in this chapter. This chapter will cover the following topics within photogrammetry: camera model, camera calibration, stereo camera geometry, stereo camera calibration, essential matrix, epipolar geometry, rectification and triangulation. The majority of the theory in this section is based on the two books [15] and [68]. The theory in this chapter is essential for the proposed method, which is shown later in section 7.1.

3.1 Camera model

A common representation of a camera is the pinhole model. It is the most specialized and simplest camera model [68, p.153]. The model describes how objects in the real world are projected onto the image plane. This projection process is called true 3D perspective [15, p.53]. It is most commonly used in computer graphics and computer vision, as it more accurately models the behaviour of real cameras. The centre of the projection is the origin of a Euclidean coordinate system, and let $z = f$, which is called the image plane or focal plane. With the pinhole camera model, a world coordinate $\boldsymbol{X} = (X, Y, Z)^T$ is projected to the point in the image plane where a line joining the point $\boldsymbol{X}$ to the centre of projection meets the image plane. This is illustrated on the left of figure 3.1.1. By using similar triangles, the world point $(X, Y, Z)^T$ is projected onto the image plane as $(f\frac{X}{Z}, f\frac{Y}{Z}, f)$. By ignoring the third dimension, the projection can be given as

$$(X, Y, Z)^T \Longrightarrow (f\frac{X}{Z}, f\frac{Y}{Z}). \tag{3.1.1}$$

With a homogeneous coordinate representation, the projection has a simpler linear form:

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{3.1.2}$$

where f_x and f_y are the focal length in x and y-direction. The matrix can be written as a diagonal matrix $\text{diag}(f_x, f_y, 1)$ multiplied with a 3×4 matrix, which consists of the identity matrix and a zero column vector.

From now on, the notation $\boldsymbol{X}$ represents the world point in homogeneous 4-vector $(X, Y, Z, 1)^T$, $\boldsymbol{x}$ represents the image point in homogeneous 3-vector and $\boldsymbol{P}$ is the 3×4 homogeneous camera projection matrix. Equation (3.1.2) can then be rewritten as

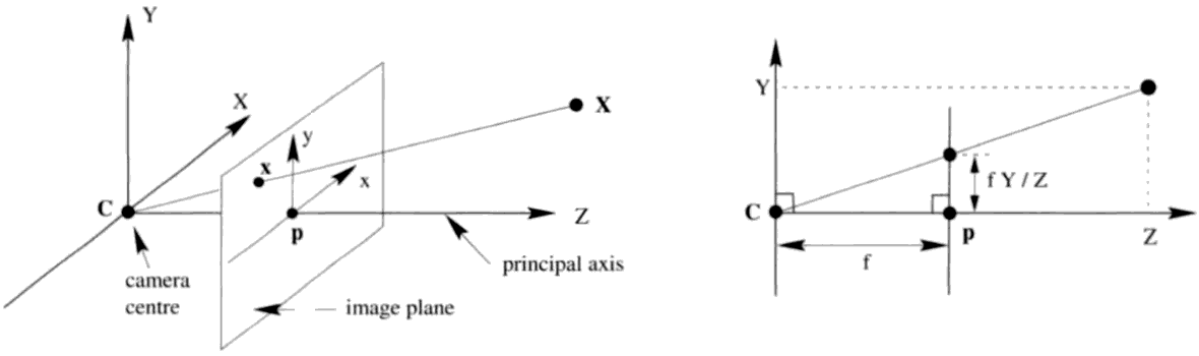


Figure 3.1.1: An illustration of the pinhole camera geometry [68, p.154].

$$\boldsymbol{x} = \boldsymbol{P}\boldsymbol{X}. \tag{3.1.3}$$

Equation (3.1.3) describes the camera matrix for the pinhole model of central projection as

$$\boldsymbol{P} = diag(f, f, 1)[\boldsymbol{I} \mid \boldsymbol{0}]. \tag{3.1.4}$$

Equation (3.1.1) assumes that the origin of coordinates in the image plane is at the principal point. In reality, this may not be the case. A new mapping shown in equation (3.1.5) is introduced to compensate for the offset of the principal point.

$$(X, Y, Z)^T \Longrightarrow (f\frac{X}{Z} + c_x, f\frac{Y}{Z} + c_y)^T \tag{3.1.5}$$

where (c_x, c_y) are the coordinates of the camera. Similar to equation (3.1.2), equation (3.1.5) can be expressed in a simpler form with homogeneous coordinates

$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} fX + Zc_x \\ fY + Zc_y \\ Z \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}. \tag{3.1.6}$$

The matrix in this expression can be referred as the camera intrinsics and can be written as

$$\boldsymbol{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.1.7}$$

This representation is one of several ways to represent the intrinsic matrix. How to find the parameters in the camera matrix is explained later in this chapter. This is done by camera calibration 5. Another way of representing the intrinsic matrix is

$$\boldsymbol{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.1.8}$$

which contains one extra variable s that encodes any possible skew between the sensor axes, due to the sensor not being mounted perpendicular to the optical axis.

By formulating $\boldsymbol{K}$ as in equation 3.1.7, equation (3.1.6) can be expressed in more concise form

$$\boldsymbol{x} = \boldsymbol{K}[\boldsymbol{I} \mid \boldsymbol{0}]\boldsymbol{X}_{\boldsymbol{cam}} \tag{3.1.9}$$

where matrix $\boldsymbol{K}$ is the camera calibration matrix and $\boldsymbol{X}_{\boldsymbol{cam}}$ is the world point. It is assumed that the world point is located at the origin of a Euclidean cooridnate system with principal axis of the camera pointing

straight down the z-axis, and the point $\boldsymbol{X_{cam}}$ is expressed in this coordinate system. This coordinate system can be called the camera coordinate frame.

In general, points in space will be described in a coordinate frame known as the world coordinate space. The two coordinate frames are related by a rotation and translation, which are called the camera extrinsics. This is illustrated in figure 3.1.2. If $\tilde{\boldsymbol{X}}$ represents an inhomogeneous 3-vector that represents a point in the world coordinate frame. Then $\tilde{\boldsymbol{x}}$ represents the same point in the camera coordinate frame. A transformation from world to camera coordinate frame may be expressed as

$$\tilde{\boldsymbol{X}}_{\boldsymbol{cam}} = \boldsymbol{R}(\tilde{\boldsymbol{X}} - \tilde{\boldsymbol{C}}) \tag{3.1.10}$$

where $\tilde{\boldsymbol{C}}$ is the coordinates of the camera centre expressed in the world frame, and $\boldsymbol{R}$ is a rotation matrix representing the orientation of the camera frame. With a homogeneous representation, this equation can be written as

$$\boldsymbol{X}_{cam} = \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{R}\tilde{\boldsymbol{C}} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & -\boldsymbol{R}\tilde{\boldsymbol{C}} \\ 0 & 1 \end{bmatrix} \boldsymbol{X} \tag{3.1.11}$$

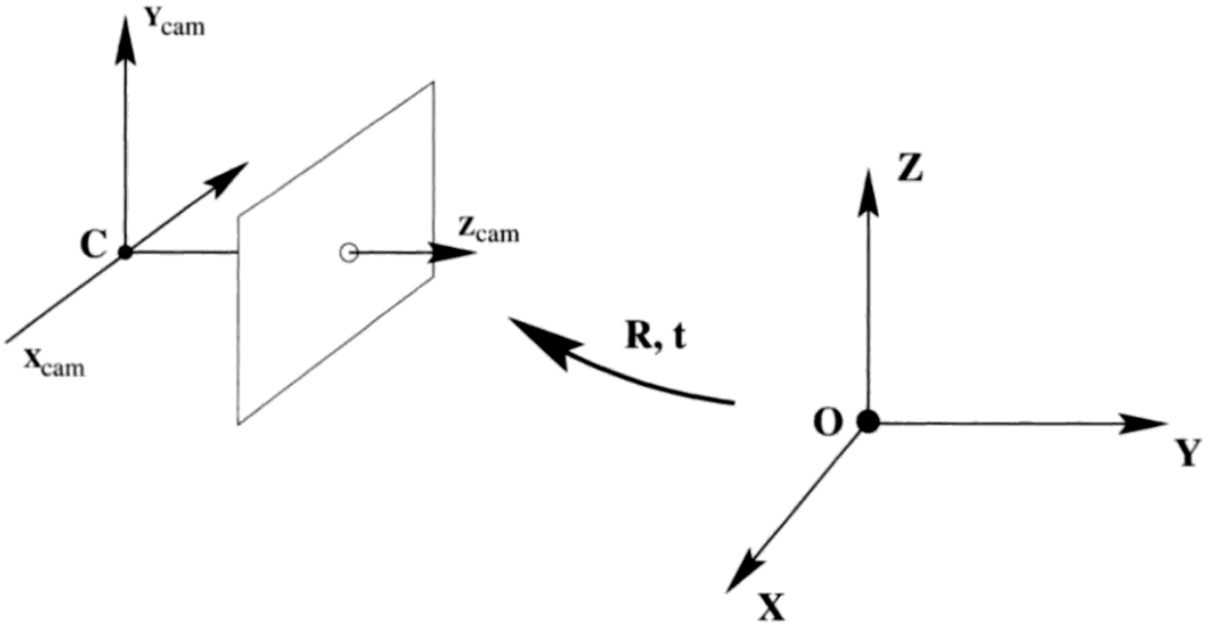


Figure 3.1.2: The transformation between world and camera coordinate frames [68].

Inserting equation (3.1.10) into equation (3.1.11) leads to equation (3.1.12)

$$\boldsymbol{x} = \boldsymbol{KR}[\boldsymbol{I} \mid -\tilde{\boldsymbol{C}}]\boldsymbol{X} \tag{3.1.12}$$

where $\boldsymbol{X}$ is expressed in the world coordinate frame. This is the general mapping given by the pinhole camera.

In general, it is convenient to not make the camera centre explicit, and rather describe the world to image transformation as $\tilde{\boldsymbol{X}}_{cam} = \boldsymbol{R}\tilde{\boldsymbol{X}} + \boldsymbol{t}$. Expressing the transformation like this will give equation (**??**), which expresses the camera matrix as

$$\boldsymbol{P} = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}] \tag{3.1.13}$$

where $\boldsymbol{t} = -\boldsymbol{R}\tilde{\boldsymbol{C}}$.

3.2 Camera calibration

Camera calibration is the process of finding the unknown parameters of the pinhole camera model presented in the previous section. A common approach to performing the calibration process is to use Zhang's method [69] which will be presented in this section. The theory in this section is also taken from [70].

3.2.1 Plane based calibration

The idea behind Zhang's method is to present a set of pictures of a planar surface for the camera. This planar surface is commonly a checkerboard with known square dimensions. During the calibration process, it is important that either the camera or the checkerboard is fixed in space. For the calibration process in this thesis, the camera is chosen to be fixed. The camera characteristics are similar for all images, only the orientation and position, $\boldsymbol{R}$ and $\boldsymbol{t}$ of the checkerboard varies. Since the size and structure of the checkerboard are known, it is possible to define a coordinate system on the checkerboard that describes where each corner is located in the real world. The origin of the world frame is defined as the upper left corner of the plane, the directions of X and Y move right and down, respectively, and Z is orthogonal to the plane. Based on this, Zhang's method assumes that the plane is on $Z = 0$ on the world coordinate system. This shows that any point on the plane has the coordinates

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{K}[\boldsymbol{R} \mid \boldsymbol{t}] \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{3.2.1}$$

where $\boldsymbol{R} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix}$ and λ is an arbitrary scalar. Since Z is always zero, equation (3.2.1) can be simplified as

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{K}[\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{r}_3 \mid \boldsymbol{t}] \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}. \tag{3.2.2}$$

Defining $H = \boldsymbol{K}\ [\boldsymbol{r}_1 \quad \boldsymbol{r}_2 \quad \boldsymbol{r}_3 \mid \boldsymbol{t}]$ gives the following equation

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{3.2.3}$$

where $\boldsymbol{H} \in \mathbb{R}$ describes the homography matrix that maps points from the checkerboard to the corresponding pixel points in the image plane.

3.2.2 Homography estimation

The camera calibration technique is dependent on a homography estimation. Let the homography matrix $\boldsymbol{H}$ be expressed as

$$\boldsymbol{H} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}. \tag{3.2.4}$$

Equation 3.2.3 can be derived as

$$\lambda \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}. \tag{3.2.5}$$

By dividing equation (3.2.5) with λ, the pixel coordinates x and y can be mapped into imhomogeneous coordinates

$$x = \frac{h_{11}X + h_{12}Y + h_{13}}{h_{31}X + h_{32}Y + h_{33}} \tag{3.2.6}$$

$$y = \frac{h_{21}X + h_{22}Y + h_{23}}{h_{31}X + h_{32}Y + h_{33}} \tag{3.2.7}$$

where

$$\lambda = h_{31}X + h_{32}Y + h_{33}. \tag{3.2.8}$$

The goal is to solve for $\boldsymbol{H}$. Arranging equation (3.2.6) and (3.2.7) gives the following equations

$$\boldsymbol{a}_x^T \boldsymbol{h} = 0 \tag{3.2.9}$$

$$\boldsymbol{a}_y^T \boldsymbol{h} = 0 \tag{3.2.10}$$

where

$$\boldsymbol{h} = \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{bmatrix}^T \tag{3.2.11}$$

$$\boldsymbol{a_x} = \begin{bmatrix} -X & -Y & -1 & 0 & 0 & 0 & xX & xY & x \end{bmatrix}^T \tag{3.2.12}$$

$$\boldsymbol{a_y} = \begin{bmatrix} 0 & 0 & 0 & -X & -Y & -1 & yX & yY & y \end{bmatrix}^T. \tag{3.2.13}$$

Given a set of corresponding points, a set of linear equations can be formed and solved as a linear system

$$\boldsymbol{A}\boldsymbol{h} = 0, \qquad \boldsymbol{h} \neq 0 \tag{3.2.14}$$

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_{x1}^T \\ \boldsymbol{a}_{y1}^T \\ \cdot \\ \cdot \\ \boldsymbol{a}_{xm}^T \\ \boldsymbol{a}_{ym}^T \end{bmatrix}, \qquad m \geq 4 \tag{3.2.15}$$

where m is the number of correspondences. The homography matrix $\boldsymbol{H}$ has 8 degrees of freedom and is defined up to scale, given a minimum of 4 correspondences. This linear system can be solved by using linear least squares method and singular value decomposition (SVD). The sum of squared error can be defined as

$$f(\boldsymbol{h}) = \frac{1}{2}\boldsymbol{h}^T \boldsymbol{A}^T \boldsymbol{A}\boldsymbol{h} \tag{3.2.16}$$

$$\frac{df}{d\boldsymbol{h}} = 0 = \frac{1}{2}(\boldsymbol{A}^T\boldsymbol{A} + (\boldsymbol{A}^T\boldsymbol{A})^T)\boldsymbol{h} \tag{3.2.17}$$

$$0 = \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{h}. \tag{3.2.18}$$

The SVD is defined as

$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{V}^T \tag{3.2.19}$$

where $\boldsymbol{A} \in \mathbb{R}^{mxn}$, $\boldsymbol{U} \in \mathbb{R}^{mxm}$, $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}_m$ and the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$ make up the columns of $\boldsymbol{U}$. The columns of $\boldsymbol{U}$ are the left singular vectors of $\boldsymbol{A}$. $\boldsymbol{V} \in \mathbb{R}^{nxn}$, $\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}_n$ and the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$ make up the columns of $\boldsymbol{V}$. $\sum \in \mathbb{R}_{mxn}$ is a rectangular diagonal matrix. It has elements along the diagonal which are square root of the eigenvalues of both $\boldsymbol{A}^T\boldsymbol{A}$ and $\boldsymbol{A}\boldsymbol{A}^T$ in descending order. A more detailed explanation of the SVD can be found in [71, p.66-67].

3.2.3 Intrinsic matrix

Now that the homography matrix is known, the intrinsic matrix $\boldsymbol{K}$ can now be calculated. For every camera position, there is a projection transformation $\boldsymbol{H} = \lambda\boldsymbol{K}\begin{bmatrix}\boldsymbol{R} & \boldsymbol{t}\end{bmatrix}$. The projection transformation can be written in column form, which is shown in equation (3.2.20).

$$\lambda\boldsymbol{K}^{-1}\begin{bmatrix}\boldsymbol{h}_1\\ \boldsymbol{h}_2\\ \boldsymbol{h}_3\end{bmatrix}^T = \begin{bmatrix}\boldsymbol{r}_1\\ \boldsymbol{r}_2\\ \boldsymbol{t}\end{bmatrix}^T \tag{3.2.20}$$

Equation (3.2.20) can be broken up and provide 3 equations

$$\lambda\boldsymbol{K}^{-1}\boldsymbol{h}_1 = \boldsymbol{r}_1 \tag{3.2.21}$$

$$\lambda\boldsymbol{K}^{-1}\boldsymbol{h}_2 = \boldsymbol{r}_2 \tag{3.2.22}$$

$$\lambda\boldsymbol{K}^{-1}\boldsymbol{h}_3 = \boldsymbol{t}. \tag{3.2.23}$$

The rotation matrix $\boldsymbol{R}$ has unit vectors in the columns and it is orthogonal, such that

$$||\boldsymbol{r}_i|| = 1, \quad (\boldsymbol{r}_1)^T\boldsymbol{r}_2 = 0, \quad i = 1, 2. \tag{3.2.24}$$

By exploiting these properties, two constraints for the intrinsic matrix can be introduced given one homography

$$\boldsymbol{h}_1^T(\boldsymbol{K}\boldsymbol{K}^T)^{-1}\boldsymbol{h}_2 = 0 \tag{3.2.25}$$

$$\boldsymbol{h}_1^T(\boldsymbol{K}\boldsymbol{K}^T)^{-1}\boldsymbol{h}_1 = \boldsymbol{h}_2^T(\boldsymbol{K}\boldsymbol{K}^T)^{-1}\boldsymbol{h}_2. \tag{3.2.26}$$

Let

$$\boldsymbol{B} = (\boldsymbol{K}\boldsymbol{K}^T)^{-1} = \begin{bmatrix}B_{11} & B_{12} & B_{13}\\ B_{12} & B_{22} & B_{23}\\ B_{13} & B_{23} & B_{33}\end{bmatrix} \tag{3.2.27}$$

$$\begin{bmatrix}B_{11} & B_{12} & B_{13}\\ B_{12} & B_{22} & B_{23}\\ B_{13} & B_{23} & B_{33}\end{bmatrix} = \begin{bmatrix}\frac{1}{f_x^2} & -\frac{s}{\alpha^2\beta} & \frac{c_y\gamma - c_x f_y}{f_x^2 f_y}\\ -\frac{s}{f_x^2 f_y} & \frac{s^2}{f_x^2 f_y^2} + \frac{1}{f_x^2} & \frac{s(f_y s - c_x f_y)}{f_x^2 f_y^2} - \frac{c_y}{f_y^2}\\ \frac{c_y s - c_x\beta}{f_x^2 f_y} & \frac{s(c_y s - c_x f_y)}{f_x^2 f_y^2} - \frac{c_y}{f_y^2} & \frac{(c_y s - c_x f_y)^2}{f_x^2 f_y^2} + \frac{c_y^2}{f_y^2} + 1\end{bmatrix}. \tag{3.2.28}$$

From the equation, it is observable that $\boldsymbol{B}$ is symmetric and is defined by a 6D vector

$$\begin{bmatrix}B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33}\end{bmatrix}^T \tag{3.2.29}$$

Let the i^{th} column vector of $\boldsymbol{H}$ be $\boldsymbol{h}_i = \begin{bmatrix}h_{i1} & h_{i2} & h_{i3}\end{bmatrix}$. This results in the following equation

$$\boldsymbol{h}_i^T\boldsymbol{B}\boldsymbol{h}_j = \boldsymbol{v}_{ij}^T\boldsymbol{b} \tag{3.2.30}$$

with

$$\boldsymbol{v}_{ij} = \begin{bmatrix} h_{i1}h_{j1} & h_{i1}h_{j2} + h_{i2}h_{j1} & h_{i2}h_{j2} & h_{i3}h_{j1} + h_{i1}h_{j3} & h_{i3}h_{j2} & h_{i2}h_{j3} + h_{i2}h_{j3} & h_{i3}h_{j3} \end{bmatrix}^T. \tag{3.2.31}$$

Hence, the two constraints shown in equation (3.2.25) and (3.2.26) can be rewritten as two homogeneous equations

$$\begin{bmatrix} \boldsymbol{v}_{12}^T \\ (\boldsymbol{v}_{11} - \boldsymbol{v}_{22})^T \end{bmatrix} \boldsymbol{b} = 0. \tag{3.2.32}$$

If there are n images of the checkerboard taken, then equation (3.2.32) can be stacked to form a matrix

$$\boldsymbol{V}\boldsymbol{b} = 0, \quad \boldsymbol{V} \in \mathbb{R}^{2n \times 6}. \tag{3.2.33}$$

When $n \geq 3$, a unique solution for $\boldsymbol{b}$ can be found. This is solved the same way as for the homography matrix.

Once $\boldsymbol{b}$ is estimated, the intrinsics parameters for the camera matrix shown in equation (3.1.8) can be uniquely extracted from matrix $\boldsymbol{B}$.

$$c_y = \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B_{22} - B_{12}^2} \tag{3.2.34}$$

$$\lambda = \frac{B_{33} - B_{13}^2 + c_x(B_{12}B_{13} - B_{11}B_{23})}{B_{11}} \tag{3.2.35}$$

$$f_x = \sqrt{\frac{\lambda}{B_{11}}} \tag{3.2.36}$$

$$f_y = \sqrt{\frac{\lambda B_{11}}{B_{11}B_{22} - B_{12}^2}} \tag{3.2.37}$$

$$s = -\frac{B_{12}f_x^2 f_y}{\lambda} \tag{3.2.38}$$

$$c_x = \lambda\frac{c_y}{f_x} - \frac{B_{13}f_x^2}{\lambda} \tag{3.2.39}$$

3.2.4 Extrinsic matrix

When the camera matrix $\boldsymbol{K}$ is found, the extrinsic parameters can be computed for each homography. From equations (3.2.21) - (3.2.23), the following equations can be derived

$$\boldsymbol{r}_1 = \lambda\boldsymbol{A}^{-1}h_1 \tag{3.2.40}$$

$$\boldsymbol{r}_2 = \lambda\boldsymbol{A}^{-1}h_2 \tag{3.2.41}$$

$$\boldsymbol{r}_3 = \boldsymbol{r}_1 \times \boldsymbol{r}_2 \tag{3.2.42}$$

$$\boldsymbol{t} = \lambda\boldsymbol{A}^{-1}h_3 \tag{3.2.43}$$

where

$$\lambda = \frac{1}{||\boldsymbol{A}^{-1}h_1||} = \frac{1}{||\boldsymbol{A}^{-1}h_2||} \tag{3.2.44}$$

and the extrinsic matrix is expressed as

$$\begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \end{bmatrix} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 & \boldsymbol{t} \end{bmatrix}. \tag{3.2.45}$$

3.2.5 Rotation matrix approximation

Noise can appear, which can cause the estimated parameters of the rotation matrix $\boldsymbol{R}$ to not satisfy the properties of a rotation matrix. Hence, $\boldsymbol{R}$ needs to be estimated such that it meets the conditions of a rotation matrix. Let

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{r}_1 & \boldsymbol{r}_2 & \boldsymbol{r}_3 \end{bmatrix} \tag{3.2.46}$$

The goal is to find the best $\boldsymbol{R}$ to approximate $\boldsymbol{Q}$. This can be accomplished by minimizing the Frobenius norm of the difference $\boldsymbol{R} - \boldsymbol{Q}$.

$$\min_{\boldsymbol{R}} \; ||\boldsymbol{R} - \boldsymbol{Q}||_F^2 \quad s.t \quad \boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{I} \tag{3.2.47}$$

$$||\boldsymbol{R} - \boldsymbol{Q}||_F^2 = Tr((\boldsymbol{R} - \boldsymbol{Q})^T(\boldsymbol{R} - \boldsymbol{Q})) \tag{3.2.48}$$

$$||\boldsymbol{R} - \boldsymbol{Q}||_F^2 = Tr(\boldsymbol{R}^T\boldsymbol{R}) + Tr(\boldsymbol{Q}^T\boldsymbol{Q}) - 2Tr(\boldsymbol{R}^T\boldsymbol{Q}) \tag{3.2.49}$$

$$||\boldsymbol{R} - \boldsymbol{Q}||_F^2 = 3 + Tr(\boldsymbol{Q}^T\boldsymbol{Q}) - 2Tr(\boldsymbol{R}^T\boldsymbol{Q}) \tag{3.2.50}$$

where $\Sigma = diag(\sigma_1, \sigma_2, \sigma_3)$. Minimizing equation (3.2.47) is equivalent to maximizing $Tr(\boldsymbol{R}^T\boldsymbol{Q})$. By computing the SVD of $\boldsymbol{Q} = \boldsymbol{U}\Sigma\boldsymbol{V}^T$, the following expression is obtained

$$Tr(\boldsymbol{R}^T\boldsymbol{Q}) = Tr(\boldsymbol{R}^T\boldsymbol{U}\Sigma\boldsymbol{V}^T) = Tr(\boldsymbol{V}^T\boldsymbol{R}^T\boldsymbol{U}\Sigma). \tag{3.2.51}$$

The manipulation of equation (3.2.51) is possible because the trace is invariant under cyclic permutations. Let $\boldsymbol{Z} = \boldsymbol{V}^T\boldsymbol{R}^T\boldsymbol{U}$. This can be inserted into equation (3.2.51).

$$Tr(\boldsymbol{V}^T\boldsymbol{R}^T\boldsymbol{U}\Sigma) = Tr(\boldsymbol{Z}\Sigma = \sum_{i=1}^{3} z_{ii}\sigma_i \leq \sum_{i=1}^{3} \sigma_i \tag{3.2.52}$$

From equation (3.2.52) it can be seen that the optimal solution is $\boldsymbol{Z} = \boldsymbol{I}$, which gives a rotation matrix defined as

$$\boldsymbol{R} = \boldsymbol{U}\boldsymbol{V}^T \tag{3.2.53}$$

which is the solution of equation (3.2.47).

3.3 Stereo camera geometry

By introducing a stereo vision system, it is possible to reconstruct the third dimension, which represents the depth. The concept of stereo vision systems is to simulate the human eyes, looking at the same points from different angles. It is also possible to use a monocular camera setup to solve the depth problem. However, monocular cameras suffer from a single depth-translation scale ambiguity issue, which is a problem for objects in motion [72]. Figure 3.3.1 shows an example of how a stereo camera setup can look like. It consists of two cameras with a known distance between them, which is called the baseline.

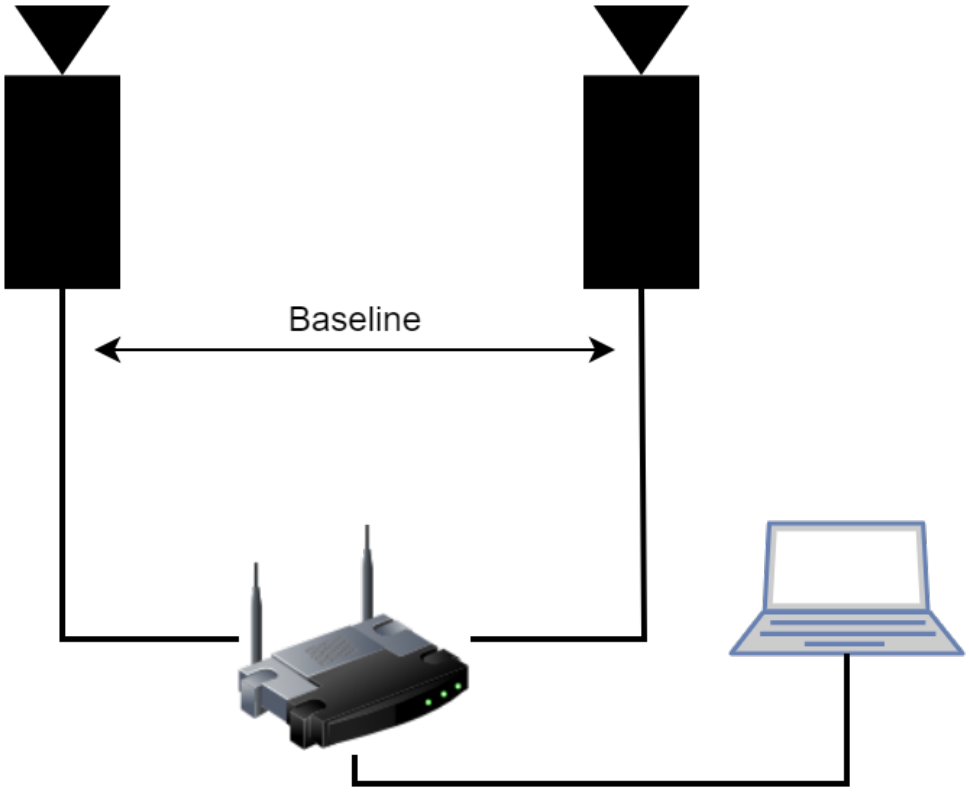


Figure 3.3.1: An illustration of the stereo camera setup used in this thesis. It has two cameras which are synchronized and powered through a router which is connected to a PC.

3.4 Calibration a stereo camera setup

Two cameras are mounted in parallel in a stereo camera setup, both fixed in space relative to each other. This can be seen in figure 3.3.1. The calibration process of a stereo camera is similar to section 3.2. When calibrating the stereo camera, the checkerboard used for calibration must be visible for both cameras. The checkerboard is used to estimate the rigid transformation between the cameras. For a stereo camera, the set of corresponding points for the left and the right camera can be defined as

$$\boldsymbol{x}_{ij}^{l} = \boldsymbol{K}_{i}^{l} \begin{bmatrix} \boldsymbol{R}_{i}^{l} & \boldsymbol{t}_{i}^{l} \end{bmatrix} \boldsymbol{X}_{i}^{l} \tag{3.4.1}$$

$$\boldsymbol{x}_{ij}^{r} = \boldsymbol{K}_{i}^{r} \begin{bmatrix} {}^{r}\boldsymbol{R}_{i}^{r} & \boldsymbol{t}_{i}^{r} \end{bmatrix} \boldsymbol{X}_{i}^{r}, \quad i, j \in m, n \tag{3.4.2}$$

where m is the number of test images and n is the number of correspondences. Equation (3.4.1) and (3.4.2) are the parameters and image points for left and right camera. In figure 3.4.1 an illustration of the stereo camera geometry is shown.

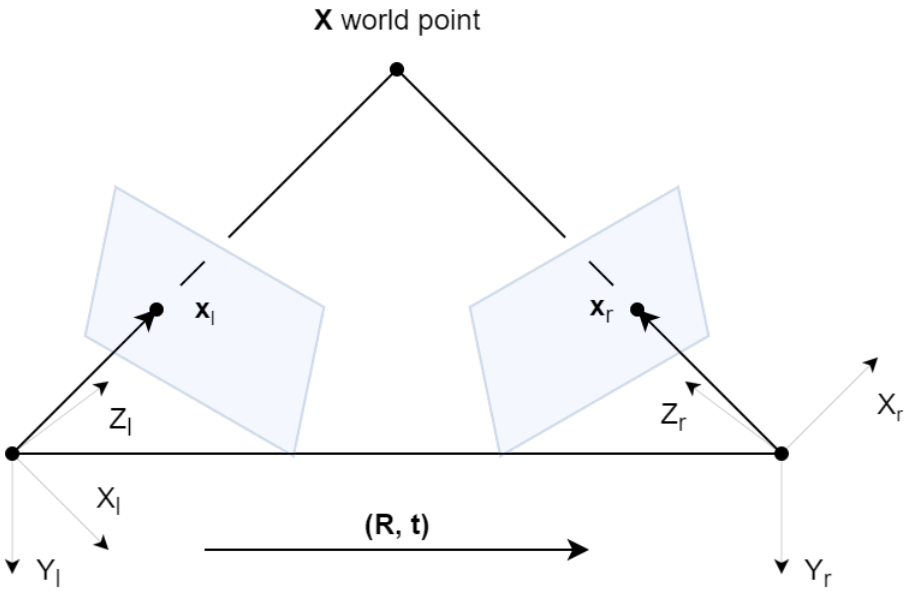


Figure 3.4.1: An illustration of a typical stereo camera geometry where two cameras are looking at the same point.

3.5 Essential matrix

Consider figure 3.4.1, which illustrates a 3D world point $\boldsymbol{X}$ being viewed from two cameras where the relative position can be transformed by a rotation $\boldsymbol{R}$ and a translation $\boldsymbol{t}$. Since the camera positions are unknown, without loss of generality, the left camera can be set to have a rotation $\boldsymbol{R} = \boldsymbol{I}$ and $\boldsymbol{t} = 0$.

A 3D point $\boldsymbol{X}_0 = d_0 \hat{x}_0$ is observed in the left image at location $\hat{x}_0$ with a distance d_0, which is mapped into the right image by the transformation

$$d_1 \hat{\boldsymbol{x}}_1 = X_1 = \boldsymbol{R}\boldsymbol{X}_0 + t = \boldsymbol{R}(d_0 \hat{\boldsymbol{x}}_0) + t \tag{3.5.1}$$

where $\hat{x}_j = \boldsymbol{K}_j^{-1} x_j$ are the ray direction vectors. Taking the cross product of both sides with $\boldsymbol{t}$ to remove it gives the following equation

$$d_1 [\boldsymbol{t}]_{\times} \hat{\boldsymbol{x}}_1 = d_0 [\boldsymbol{t}]_{\times} \boldsymbol{R} \hat{\boldsymbol{x}}_0. \tag{3.5.2}$$

Taking the dot product of both sides with $\hat{x}_1$ results in

$$d_0 \hat{x}_1^T ([\boldsymbol{t}]_\times \boldsymbol{R}) \hat{\boldsymbol{x}}_0 = d_1 \hat{x}_1^T [\boldsymbol{t}]_\times \hat{\boldsymbol{x}}_1 = 0, \tag{3.5.3}$$

since the $[t]_\times$ is skew symmetric and returns 0 when multiplying with the same vector.

This gives the epipolar constraint

$$\hat{\boldsymbol{x}}_1^T \boldsymbol{E} \hat{\boldsymbol{x}}_0 = 0 \tag{3.5.4}$$

where

$$\boldsymbol{E} = [\boldsymbol{t}]_\times \boldsymbol{R} \tag{3.5.5}$$

is defined as the essential matrix.

The essential matrix can be estimated by using the eight-point algorithm. Given $n \geq 8$ correspondences, it is possible to calculate an estimate of $\boldsymbol{E}$ up to scale for the values in $\boldsymbol{E}$ by using SVD. Recall equation (3.2.19). Assuming matrix $\boldsymbol{A}$ is solved with SVD, $\boldsymbol{E}$ can be found by extracting the elements in the last column of $\boldsymbol{V}^T$.

3.6 Epipolar geometry

The epipolar geometry is often motivated when searching for corresponding points between two frames. This geometry reduces the number of potential correspondences and speeds up the correspondence search by reducing the search problem from being 2D to 1D [15, p.754] [68, p.239]. Figure 3.6.1 illustrates how pixel $\boldsymbol{x}_l$ in one image projects to an epipolar line segment in the other image. The segment is bounded at one end by the world point $\boldsymbol{X}$ and at the other end by the projection of the origin of the left camera into the right camera. This is known as the epipole $\boldsymbol{e}_1$. If the epipolar line is projected in the second image back to the reference image, another epipolar line segment is obtained and bounded by the other corresponding epipole $\boldsymbol{e}_0$. By extending these two epipoles to infinity, a pair of corresponding epipolar lines construct the epipolar plane shown in figure 3.6.1, which is the yellow area in the figure. The epipolar plane passes through both camera origins and the world point $\boldsymbol{X}$.

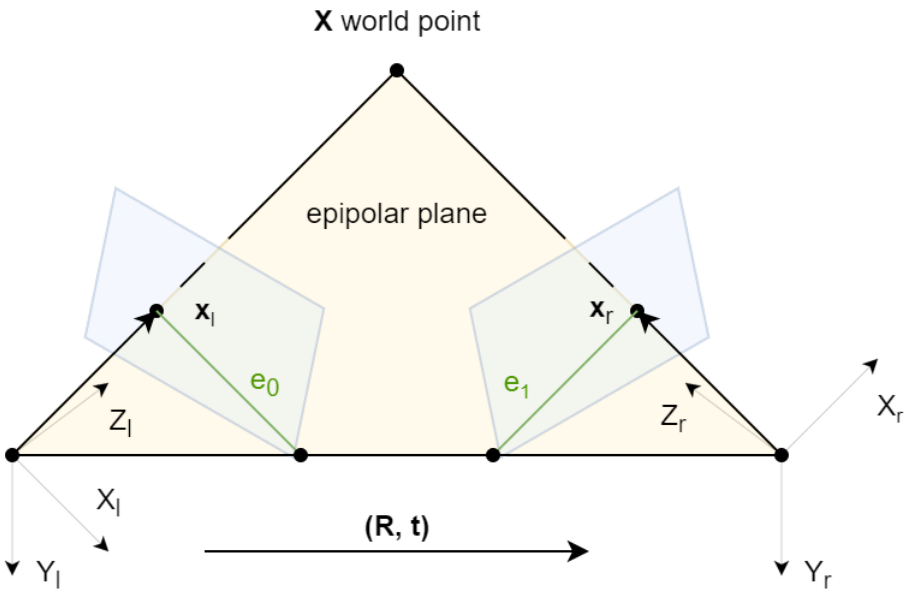


Figure 3.6.1: The epipolar geometry of a stereo camera setup. It illustrates the two corresponding epipolar lines and the epipolar plane.

3.7 Rectification

As shown in section 3.5, the epipolar geometry can implicitly be computed by using the eight-point algorithm to estimate the essential matrix. After this geometry has been calculated, the epipolar line corresponding to one pixel in one image can be used to limit the search for point correspondences in the other image.

A more efficient method to establish the epipolar geometry is first to rectify the image pair so that the corresponding horizontal scanlines are epipolar lines. This is obtained by transforming the image pair such that corresponding points lie on the same horizontal line, which reduces the point correspondence search from 2D to 1D since the points should lie on the y-coordinate. An example of a rectified image pair is shown in figure 3.7.1.

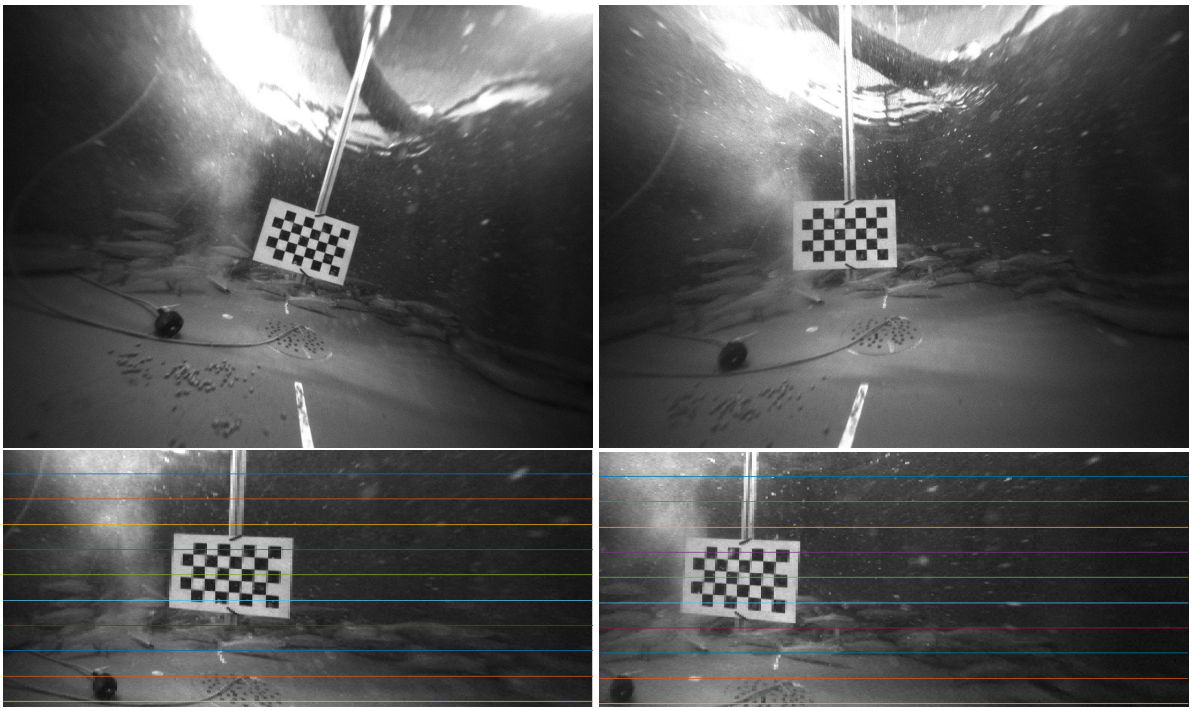


Figure 3.7.1: The original unrectified image pair is shown on the top. The two bottom images shows the rectified image pair with the epipolar lines displayed in different colors.

Rectifying two images can be done by first rotating both cameras such that they are looking perpendicular to the line joining the origins of the two cameras. Since there is a degree of freedom in the tilt, the least number of rotations should be used. The next step is to determine the twist around the camera axes. This is achieved when the camera y-axis is perpendicular to the camera centre line. It ensures that corresponding epipolar lines are horizontal and that the disparity for points further away from the camera is closer to zero. The final step is to re-scale the images to compensate for different focal lengths. However, in this thesis, this is not required as the two cameras used are identical.

3.8 Triangulation

Triangulation can be used to reconstruct the 3D point from point correspondences. With the point correspondences available, the disparity can be calculated. The baseline and focal length of a stereo camera setup are assumed to be known. This information can be combined with the disparity to reconstruct the 3D world point with the equations

$$X = \frac{x_l - x_r}{f} * Z, \tag{3.8.1}$$

$$Y = \frac{y_l - y_r}{f} * Z, \tag{3.8.2}$$

$$Z = \frac{f * b}{x_l - x_r}. \tag{3.8.3}$$

where b is the baseline between the cameras and (x_l, y_l), (x_r, y_r) are the pixel coordinates of left and right image.

After reconstructing the world points from the point correspondences, the 3D points can be used to determine the distance to an object by calculating the Euclidean distance

$$distance = \sqrt{X^2 + Y^2 + Z^2}. \tag{3.8.4}$$

With the distance established, the velocity can be computed by

$$velocity = (distance_{curr} - distance_{prev}) * FPS \tag{3.8.5}$$

where $distance_{curr}$ and $distance_{prev}$ are the distances of the object in current and previous image. FPS is how many frames the camera can capture in one second. This is found in the camera specifications.

CHAPTER 4

DATA ACQUISITION

In this thesis, there were no stereo images available for testing. Data acquisition was necessary to gather stereo images from different environment. In this section the camera used in the thesis is presented along with the necessary software to capture images simultaneously.

4.1 Camera specifications

The experimental data was gathered with a stereoscopic camera built by previous master students [73]. It is equipped with two FLIR Blackfly GigE cameras with a baseline of 230 mm and an inertial measurement unit (IMU). The IMU is not relevant for this thesis. An image of the Flir Blackfly GigE camera [1] and the technical specifications are shown in figure 4.1.1 and table 4.1.1. The whole setup of the stereo camera is displayed in figure 4.1.2. It is powered by an Ethernet connection from the attached router. It has global shutter which gives complete control over highly sensitive sensors and the possibility for external hardware triggering or software triggering. The benefit of global shutter is that the whole image will be taken at once, while rolling shutter cameras capture one line at a time. Two GPIO pins are attached with the cameras to enable synchronized image capturing.

Figure 4.1.1: The Flir Blackfly GigE used in the camera setup [1].

[1] Link to camera
[2] Software for the cameras

Parameter	**Value**
Product number	BFLY-PGE-13S2C-CS
Resolution	1288 x 964
Frame rate	30
Readout Method	Global shutter
Triggering	External and Software [2]
Synchronization	GPIO pins
Meta-data	Timestamp, image count, image parameters
Sensor	Sony ICX445
Sensor Format	1/3"
Lens Focal Length	1.28 mm
Field of View	125 °
Distortion	< 3 %
Pixel size	3.75 μm

Table 4.1.1: Specifications for Blackfly GigE [73]

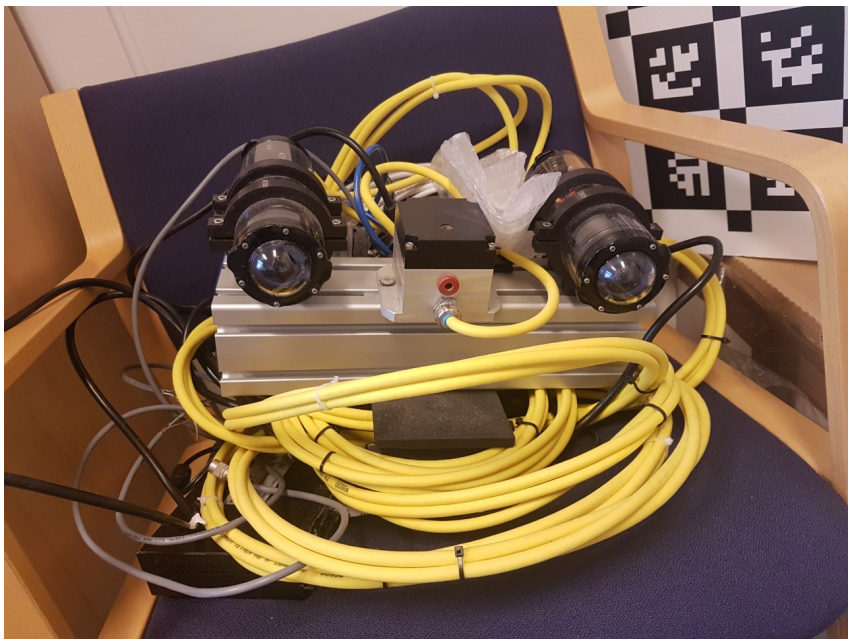


Figure 4.1.2: An image of the stereo setup.

To capture the images, software triggering was used. The producers have created a software development kit (SDK) that allows freedom for the user. It is possible to use their own program Spinnaker [2], which provides a graphical user interface to control the cameras. In addition to their program, documented Python scripts were also created which allowed users to incorporate data acquisition into their own Python program. This script can be found in Appendix A.

Image data was captured from two different environments, one in the NTNU ITK Lab, and one at SINTEF Ocean. The images from NTNU ITK lab represent a controlled environment, more suitable for validation purposes. The data from SINTEF represent a more realistic scene of the fish cages that can be used to evaluate the performance of the method in such environments. In figure 4.1.3, image samples can be seen from left and right camera.

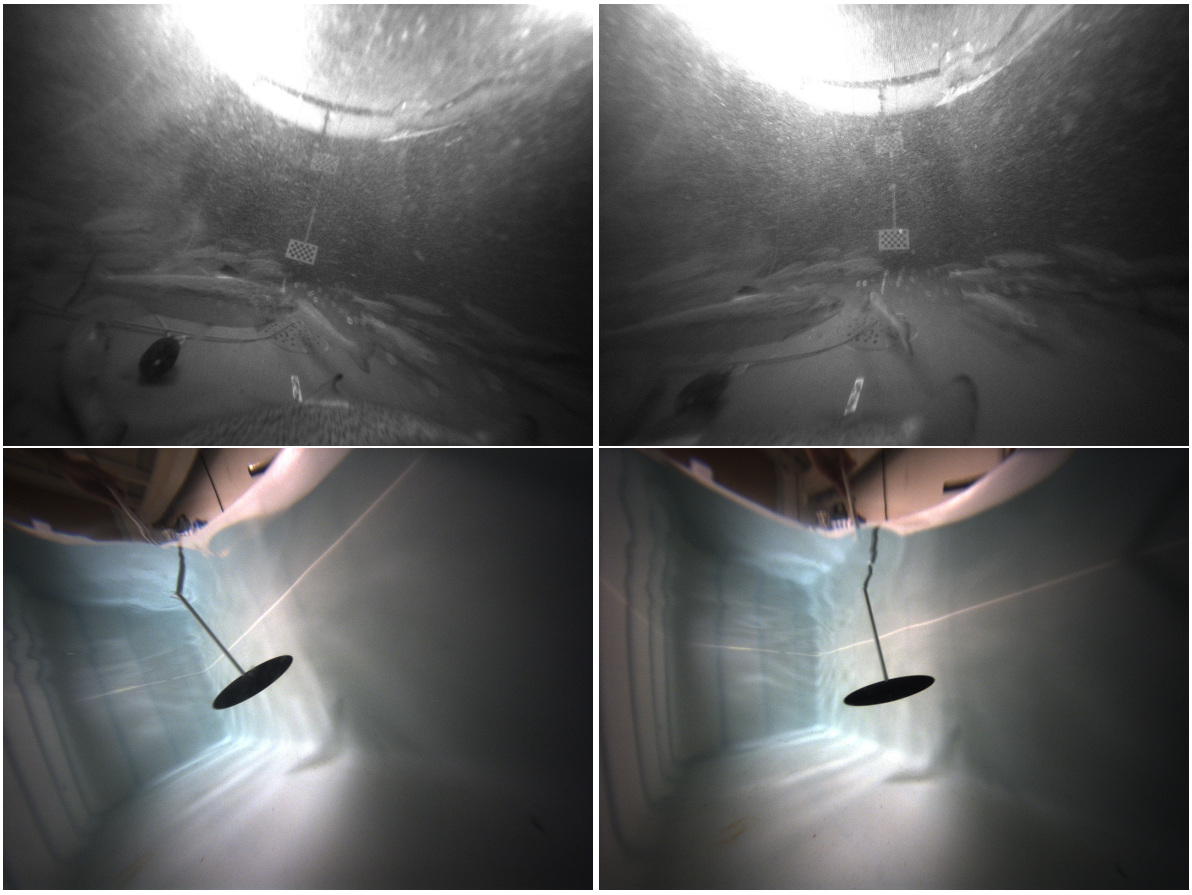


Figure 4.1.3: The images on top shows an example of an image pair taken at SINTEF Ocean. On the bottom, an image pair from the NTNU ITK Lab is shown.

CHAPTER 5

CALIBRATION RESULTS

Camera calibration must be done to extract camera parameters and to minimize the reprojection error. Since data was gathered in different environments, camera calibration also needs to be performed in the same environments. This chapter will present the calibration process and results, which uses theory related to sections 3.2 and 3.4.

To calibrate the camera, calibration images with a checkerboard needs to be taken in the different environments. These images can be processed in a calibration tool in order to extract the camera parameters. OpenCV and MATLAB are two options that can be used to perform camera calibration. In this thesis MATLAB's calibration tool was used. MATLAB provide a user friendly interface and a great visualization tool to show the calibration results. They have two similar toolboxes for camera calibration, one for single camera and one for stereo camera. The calibration process started with the stereo camera to find the accepted calibration image pairs. After finding the image pairs, they were split into left and right images to calibrate the left and right camera individually. Figure 5.0.1 shows an image of the calibration tool.

Figure 5.0.1: MATLAB calibration tool.

An advantage of using MATLAB is that it has a function to remove outliers depending on the threshold of mean error in pixels. The threshold can be adjusted, and MATLAB will automatically select images with

a higher mean error than the threshold. These images can be removed, and perform a recalibration of the camera if desired. A figure of this function is shown in figure 5.0.2. After adjusting the threshold and recalibrating the camera, MATLAB generates a script to extract the camera intrinsics.

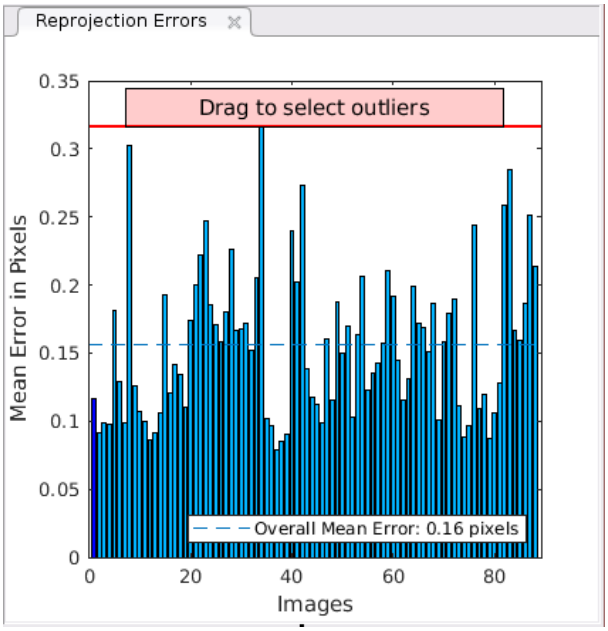


Figure 5.0.2: Tool to remove outliers depending on which threshold is set.

After calibrating the cameras individually, the image pairs were used in the stereo calibration tool. This would calibrate the camera to provide the extrinsic matrix. The left camera is assumed to be the reference camera in these results. This means that the left camera will have $\boldsymbol{R} = \boldsymbol{I}$ and $\boldsymbol{t} = 0$. The calibration results are listed in sections 5.1 and 5.2

5.1 Calibration results from NTNU ITK lab

At the NTNU ITK lab, 54 images of a checkerboard with 24mm square size were used to calibrate the cameras. Equation (5.1.1) and (5.1.2) shows the left and right intrinsic matrix, and equations (5.1.3) and (5.1.4) denotes the rotation and translation from the reference camera.

$$\boldsymbol{K}_{left} = \begin{bmatrix} 338.20 & 0.00 & 634.71 \\ 0.00 & 337.85 & 511.30 \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{5.1.1}$$

$$\boldsymbol{K}_{right} = \begin{bmatrix} 352.03 & 0.00 & 632.25 \\ 0.00 & 352.35 & 489.13 \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{5.1.2}$$

$$\boldsymbol{R} = \begin{bmatrix} 0.9473 & -0.3133 & -0.0673 \\ 0.3139 & 0.9494 & -0.0022 \\ 0.0646 & -0.0191 & 0.9977 \end{bmatrix} \tag{5.1.3}$$

$$\boldsymbol{t} = \begin{bmatrix} -239.78 \\ 34.30 \\ -51.54 \end{bmatrix} \tag{5.1.4}$$

5.2 Calibration results from SINTEF

In the calibration process at SINTEF, 51 images were used along with a checkerboard with 30 mm square size to calibrate the cameras. Equations (5.2.1) and (5.2.2) shows the left and right intrinsic matrix, and equations (5.2.3) and (5.2.4) denotes the rotation and translation from the reference camera.

$$\boldsymbol{K}_{left} = \begin{bmatrix} 335.02 & 0.00 & 634.23 \\ 0.00 & 334.61 & 513.80 \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{5.2.1}$$

$$\boldsymbol{K}_{right} = \begin{bmatrix} 363.83 & 0.00 & 628.77 \\ 0.00 & 364.17 & 488.85 \\ 0.00 & 0.00 & 1.00 \end{bmatrix} \tag{5.2.2}$$

$$\boldsymbol{R} = \begin{bmatrix} 0.9471 & -0.3105 & -0.0811 \\ 0.3139 & 0.9494 & -0.0024 \\ 0.0779 & -0.0230 & 0.9967 \end{bmatrix} \tag{5.2.3}$$

$$\boldsymbol{t} = \begin{bmatrix} -213.15 \\ 34.79 \\ -51.37 \end{bmatrix} \tag{5.2.4}$$

In [15, p.58], it is suggested that the focal length and camera center is calculated as equation (5.2.5)

$$f = \frac{W}{2}[tan(\frac{\theta_H}{2})]^{-1}, \quad (c_x, c_y) = (\frac{W}{2}, \frac{H}{2}). \tag{5.2.5}$$

where θ_H is the field of view, W is the width of the image and H is the height of the image.

Inserting the camera values from table 4.1.1 gives equations (5.2.6) and (5.2.7).

$$f = \frac{1288}{2}[tan(\frac{125^\circ}{2})]^{-1}, \quad (c_x, c_y) = (\frac{1288}{2}, \frac{964}{2}), \tag{5.2.6}$$

$$f = 335.24, \quad (c_x, c_y) = (644, 482). \tag{5.2.7}$$

There are some differences between these values and the calibration results. The focal length of the left camera seems to be quite similar, while the right camera has a slight difference in the focal length. This can result from the right camera being calibrated slightly worse than the reference camera, or it can be a natural issue in the product itself. The error is not significant, which makes it sufficient for experimental tests. It is expected to have some offset since it is an estimation of the camera parameters. A similar case can be seen in the camera centre.

CHAPTER 6

DEEP LEARNING

6.1 Neural networks

Artificial Neural Networks (ANNs), which are also called neural networks, have been widely used in areas of human interests [74]. They are based on the biological neural network of the human brain. The core idea behind neural networks came from cognitive science, where a considerable number of computational units are connected to show intelligent behaviours [75]. One type of such network is the multilayered perceptron network. They consist of a system of neurons or nodes [76]. Figure 6.1.1 shows an example of a neural network with one hidden layer.

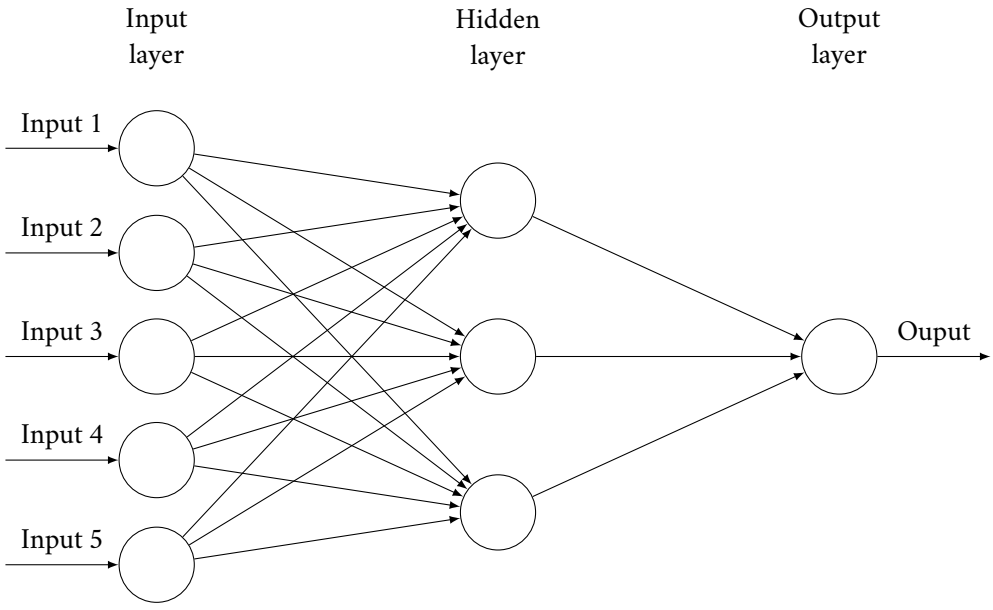


Figure 6.1.1: An example of a neural network.

These networks are modelled to represent a nonlinear mapping between an input vector and an output vector. The neurons are connected by weights and output signals, consisting of the sum of inputs to the neuron. These are modified by a simple nonlinear transfer function, also known as an activation function. A commonly used activation function is the Rectified Linear Units [15, p.272].

With a suitable set of weights and activation functions, a neural network is able to approximate any smooth, measurable function between the input layer and output layer [77]. These weights can be obtained

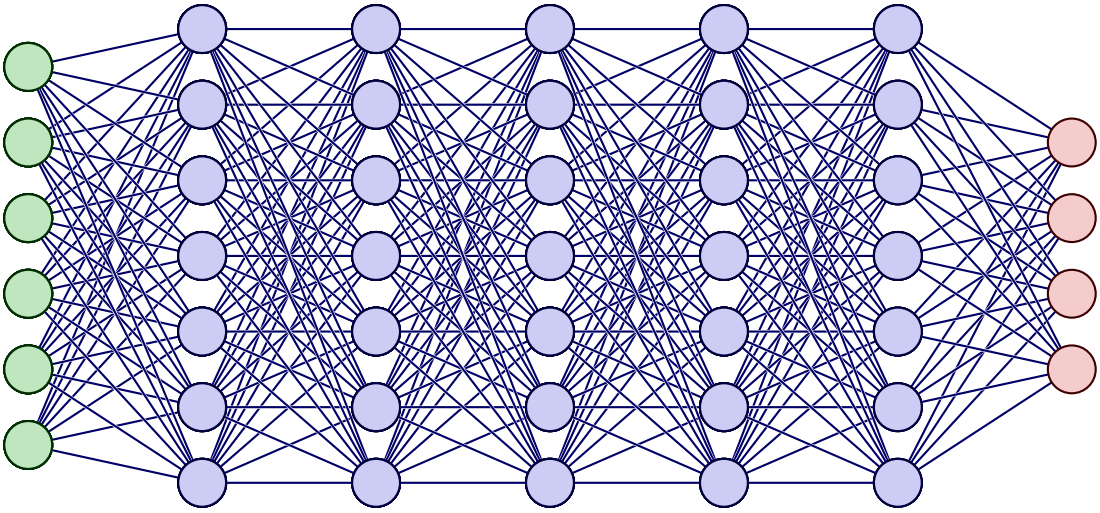


Figure 6.1.2: A figure of a fully-connected deep neural network with 5 hidden layers [79].

through training. A set of training data is required to train a neural network. It consists of a series of input and associated output vectors. When training a neural network, it is repeatedly presented with training data and adjusts the weights accordingly until the desired input-output is obtained. A common way to train a network is supervised learning. In the training process, the output for a given input may not equal the desired output. An error signal is established as the difference between the expected and actual output. Supervised training uses this error signal to adjust the weights in the network such that the overall error is minimized.

A deep neural network (DNN) is an extension of the neural network. They consist of multiple hidden layers, which increases the complexity of the model. It allows the network to transform the input data using different functions that represent the data in a hierarchical way [78]. Figure 6.1.2 shows an example of a fully-connected deep neural network. A fully-connected network can be recognized when every node is connected in the previous and next layer.

One advantage of utilizing DNNs is the automatic feature extraction from raw data. In a DNN, the numerous layers are connected to each other. The connection allows the extraction of high level features that are formed by the composition of lower level features [78]. Figure 6.1.3 shows how the DNN extracts features from an image to detect faces.

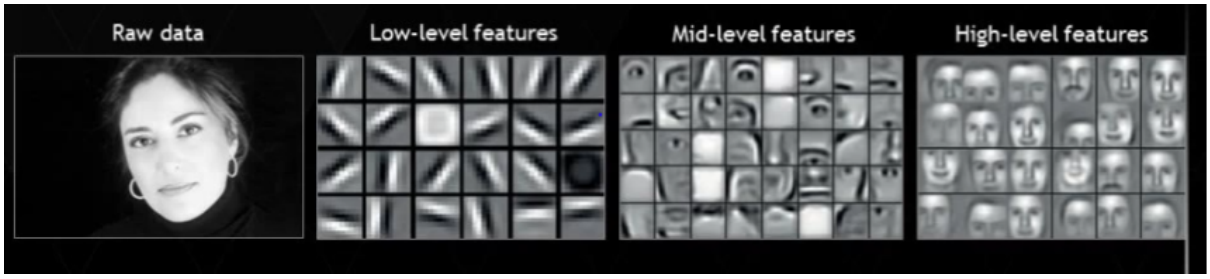


Figure 6.1.3: An example of how features from an image are found [80].

6.2 Convolutional neural networks

Convolutional neural networks are one of the most successful types of neural networks [75]. They are primarily used to solve challenging image-driven recognition problems [81]. According to [82], CNNs have the advantage of being able to merge the feature extraction and feature classification processes into a single learning body. Classification is a computer vision problem to decide if a class is present in an image. This helps CNNs to learn to optimize features during the training phase directly from the raw input. CNNs have

a network structure with sparsely-connected nodes with tied weights, which means that they can process large inputs in a computationally efficient way. Another advantage is that CNNs can adapt to different input sizes and are immune to small transformations in the input data, including distortion, translation, scaling and skewing.

Each neuron in a CNN is built up by 2D planes for weights, called the kernel, and input and output, known as the feature map. The neurons have 2D planes because images have two dimensions. Figure 6.2.1 shows an illustration of the basic building blocks of a CNN that classifies a 24×24 grayscale image into two categories. It starts with the image being fed to the input layer of the CNN. An interconnection feeds the first convolutional layer assigned by a weighting filter w with a kernel size $(K_x, K_y) = (4, 4)$. Each neuron in the first convolution layer performs a linear convolution between the image and the corresponding filter to create the input feature map of the neuron. This feature map is passed through the activation function to generate the output feature map of the neuron of the convolution neuron. The size of the output feature map is decided by subsampling factors chosen to be $S_x, S_y = 3$ in this example. Each neuron's feature map is established in the pooling layer by breaking down the output feature map of the previous neuron of the convolutional layer. In the figure, 7×7 feature maps are created in the first pooling layer. After this, the same process is repeated, and the outputs of the second pooling layer are the inputs of the fully-connected layers. In the second convolutional and pooling layer, the kernel size is chosen to be identical, while the subsample factors are $S_x = S_y = 4$. The scalar outputs are forwarded through the fully-connected and output layers to produce the classification results [82].

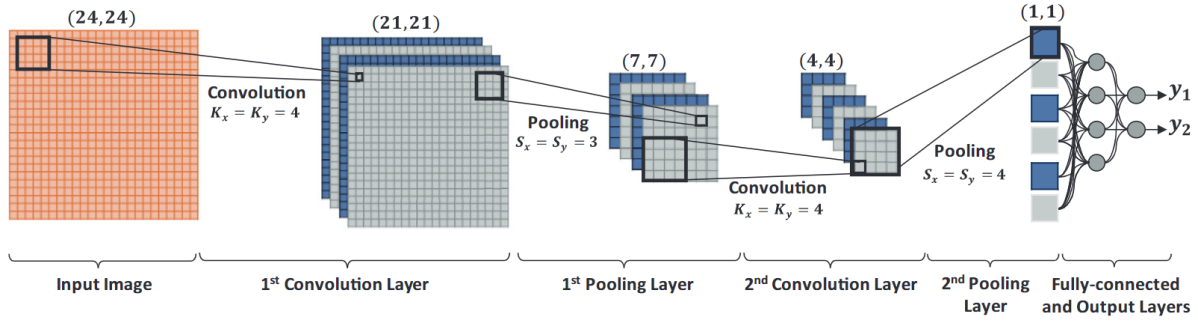


Figure 6.2.1: The illustration of the process in a CNN architecture [82].

6.3 Object detection

Classification is one of the computer vision tasks that deep learning tries to solve. Another task, which is essential in this thesis, is the detection problem. The idea behind object detection is to locate and classify objects. Rectangular bounding boxes are drawn around the detected object and classifies the object. It is closely related to the classification problem. An example of classification and object detection can be seen in figure 6.3.1b.

Object detection methods can be divided into two types of detectors: single-stage and two-stage. The two-stage detectors typically incorporate a Region Proposal Network (RPN) in the first stage. It generates region of interests, which are forwarded to the second stage for object classification and bounding box regression. Examples of state-of-the-art two-stage detectors are region-based CNN, such as Faster region-based R-CNN [84] or Mask R-CNN [48]. Such models can reach the highest accuracy rates because of the RPN, with the downside of being slower. The single-stage detectors, such as Scaled-YOLOv4 [85], and RetinaNet [46], treat object detection as a regression task by taking an image as input and learn the class probabilities and bounding box coordinates. Compared to the two-stage detectors, these models reach lower accuracy rates but significantly faster computational speed. Single-stage detectors are more suitable for real-time applications because of their speed [86].

(a) Classification (b) Object detection

Figure 6.3.1: 6.3.1a shows an example of classification, while object detection is shown in 6.3.1b. The difference is that in object detection draws a bounding box for each object with the corresponding class, while classification does not account for the amount of objects [83].

6.4 YOLOv5

YOLOv5 [1], a model from the You only look once (YOLO) network family will be used to solve the detection problem in this thesis. It is a state-of-the-art object detection architecture. The network has enormous community support, making it easy to use and incorporate. YOLOv5 is not an extension of Scaled-YOLOv4 [85], which is created by another author, but an improvement of YOLOv3 [87], which outperformed state-of-the-art methods like RetinaNet [46], Single-shotSSD [88] and Region-based fully convolutional networks (R-FCN) [89].

6.5 Creating the datasets

A training set from both environments needed to be created to test YOLOv5 on the data acquired from the two environments. The network is trained in a supervised manner, which requires the training set to have labels to tell how correct the prediction of the network is. A simple labelling tool [2] was used to label the data. The training data for YOLOv5 needs to be structured in a way such that the images have a corresponding text file, which contains the information about the object class and the bounding box coordinates. Figure 6.5.1 shows the labelling tool, figure 6.5.2 shows how the file structure needs to be setup and table 6.5.1 shows an example of the text file that is generated from the labelling tool.

class	x-coordinate	y-coordinate	width	height
0	0.244922	0.447226	0.107031	0.228687

Table 6.5.1: Content of the textfiles. The first number indicates which class it is, in this example there is only 1 object class. The rest of the numbers are the x- and y coordinates, and the width and height of the bounding box

[1] YOLOv5 GitHub
[2] Labelling tool

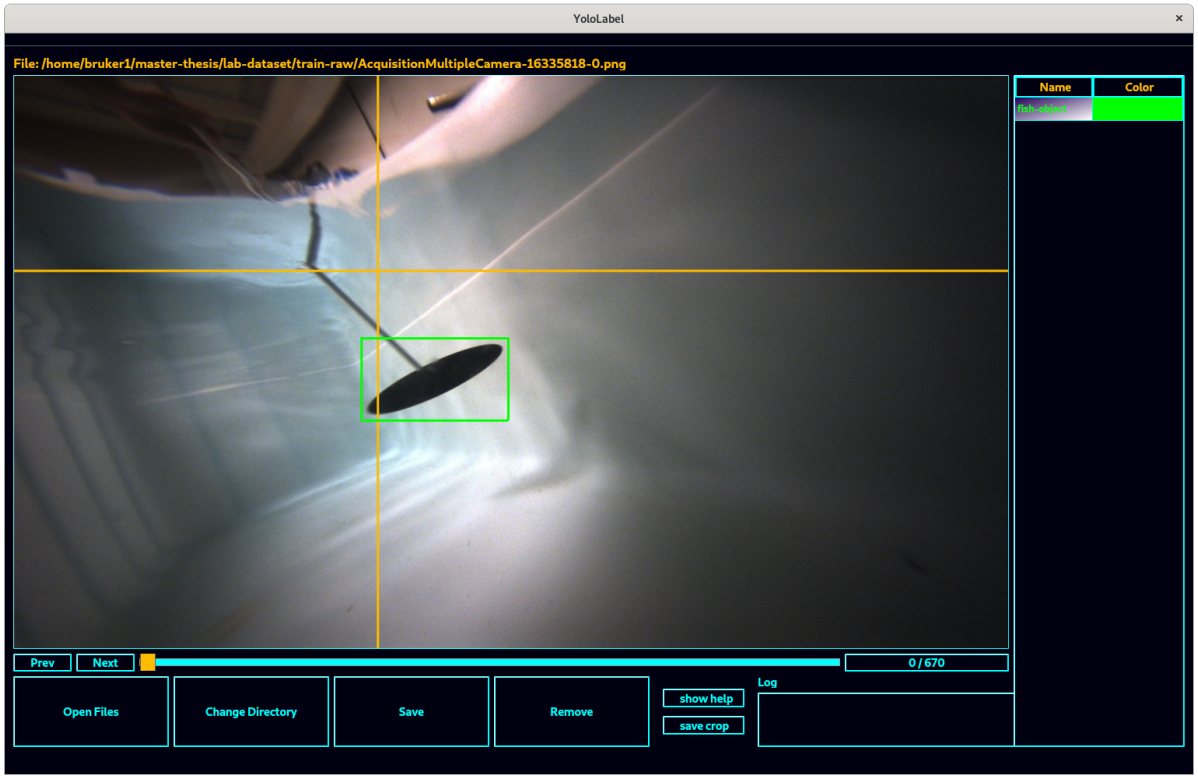


Figure 6.5.1: The labelling tool used to label the training data. It draws bounding box around the objects of interests and generates a textfile information about the class and location of the bounding box.

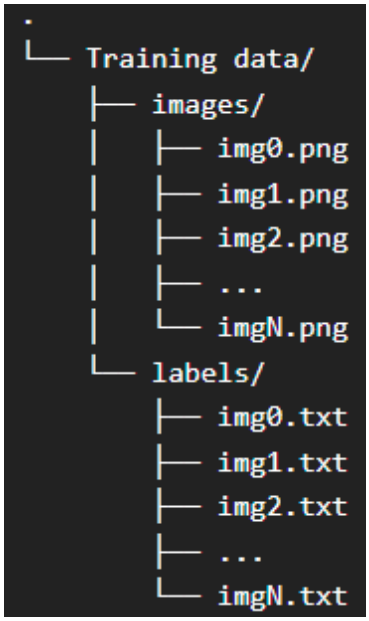
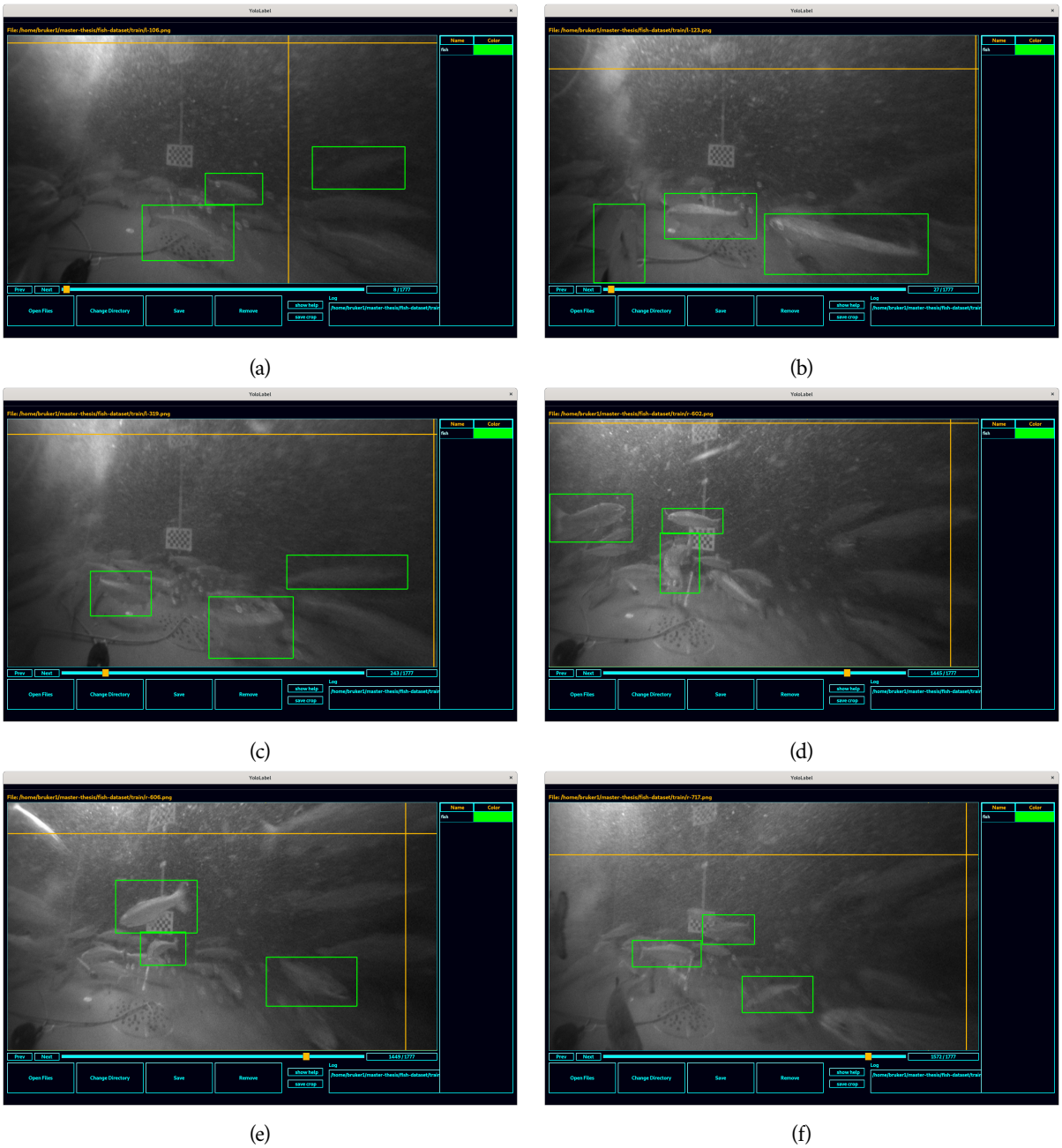


Figure 6.5.2: The file structure of the training data.

6.5.1 SINTEF dataset

The data from SINTEF contained multiple fish swimming around in a pool. Because of time limit, a number between one and five fish was labelled. Manual data labelling can be very time consuming. Figure 6.5.3 shows samples from the labelled training data. Fish that were "alone" were the focus during the labelling process. Fish with other fish close to them were avoided because it would create labels overlapping with other fish. The overlapping may cause the detection network to learn from incorrect features and perform poorly. Examples of fish being close to each other can be seen on the left side of figure 6.5.3a - 6.5.3c. In addition to the fish being alone, labelling different orientations and distances of the fish makes the detection network more robust. A total of 1778 images were first rectified to establish the epipolar geometry, then labelled as training data in the SINTEF dataset.



(a) (b)

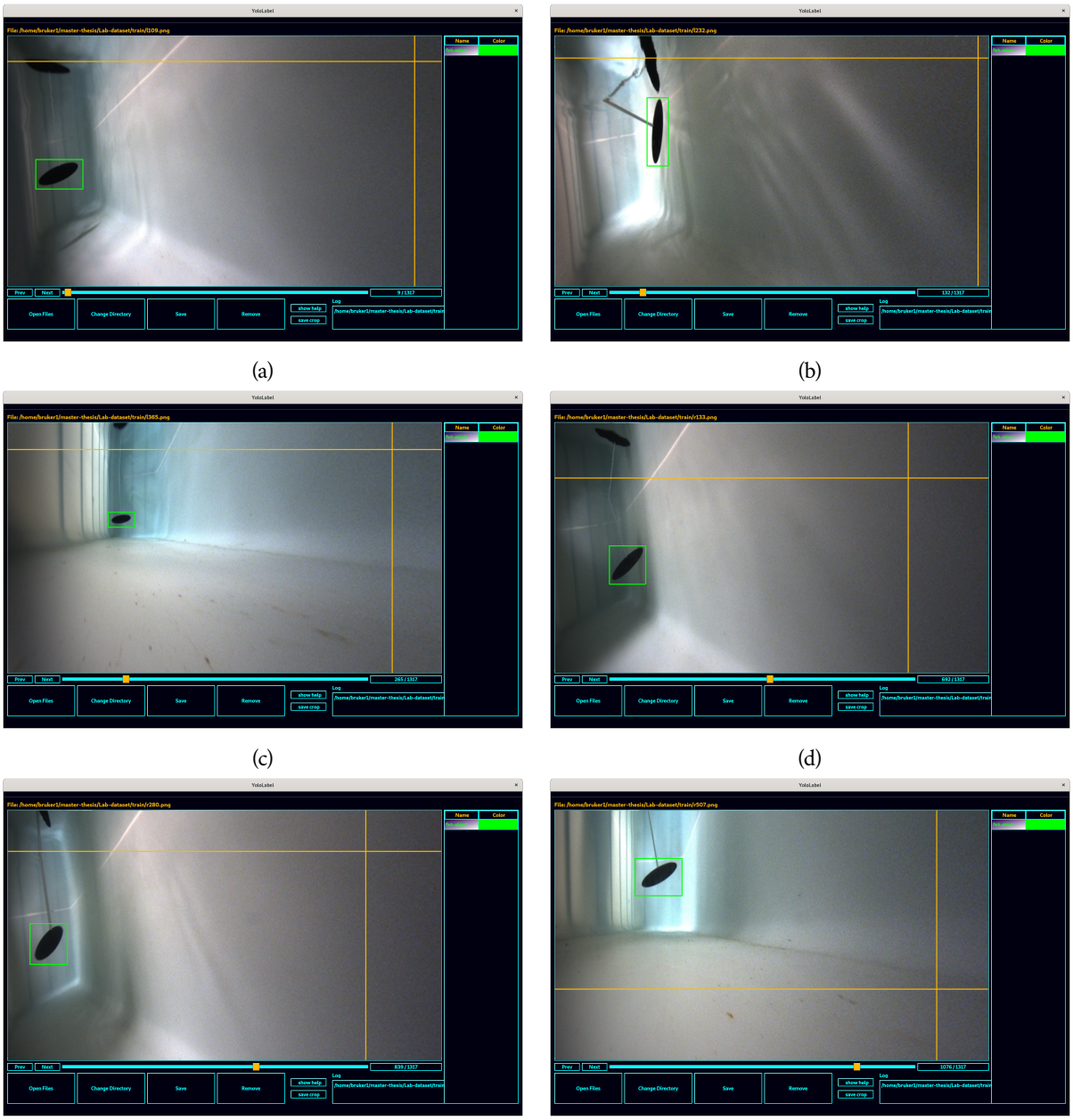
(c) (d)

(e) (f)

Figure 6.5.3: 6.5.3a - 6.5.3c and 6.5.3d - 6.5.3f shows samples of labelled images from left and right.

6.5.2 NTNU ITK Lab dataset

Labelling the data from the NTNU ITK Lab was a more manageable task as there was only one object in the scene. The object was a 3D printed ellipse, which was supposed to resemble a fish. A similar procedure was followed when labelling the data. Different orientations and distances of the object were labelled to make it able to detect the object in different situations. In this dataset, a total of 1317 images were rectified and labelled.



(a) (b)

(c) (d)

(e) (f)

Figure 6.5.4: 6.5.3a - 6.5.3c and 6.5.3d - 6.5.3f shows samples of labelled images from left and right.

6.6 Training the network

YOLOv5 offers five different model sizes for the user to choose from. A larger model size means better accuracy with the drawback of being more complex and having lower speed. A smaller model size provides

higher speed, lower complexity and lower accuracy. Figure 6.6.1 compares the speed and accuracy of the different model sizes. With a Graphical Processing Unit (GPU) available, the difference in speed is reduced to only a millisecond, which is acceptable in a real-time application. GEFORCE RTX 3090 was used to train and test YOLOv5l, the second largest model size. The GPU significantly reduces the training time and inference time. YOLOv5l was chosen as it performed faster than the largest model while maintaining a comparable accuracy.

The network's performance is evaluated with the metric mean average precision (mAP) [90], which is a popular metric to evaluate the performance of object detection networks. From the labelled data, 1778 images were used to train the network in the SINTEF environment, while 1317 images were used in the NTNU lab environment.

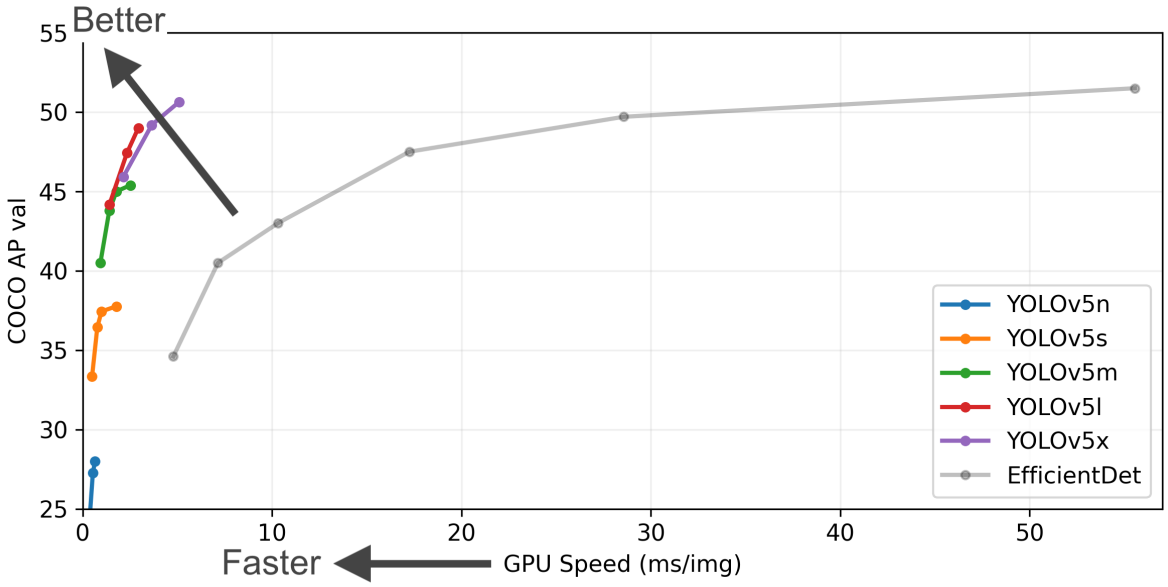
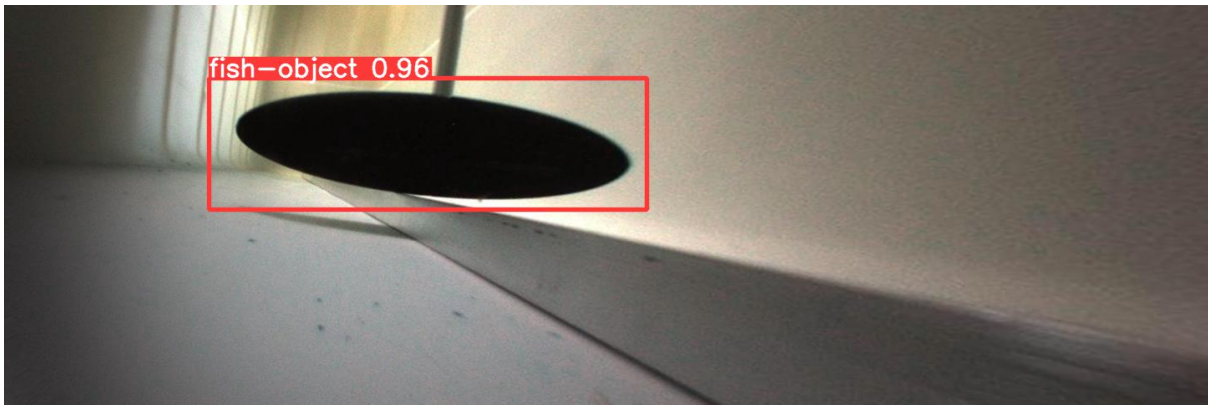


Figure 6.6.1: Comparison of YOLOv5 model sizes and EfficientDet [91].

6.7 Detection results

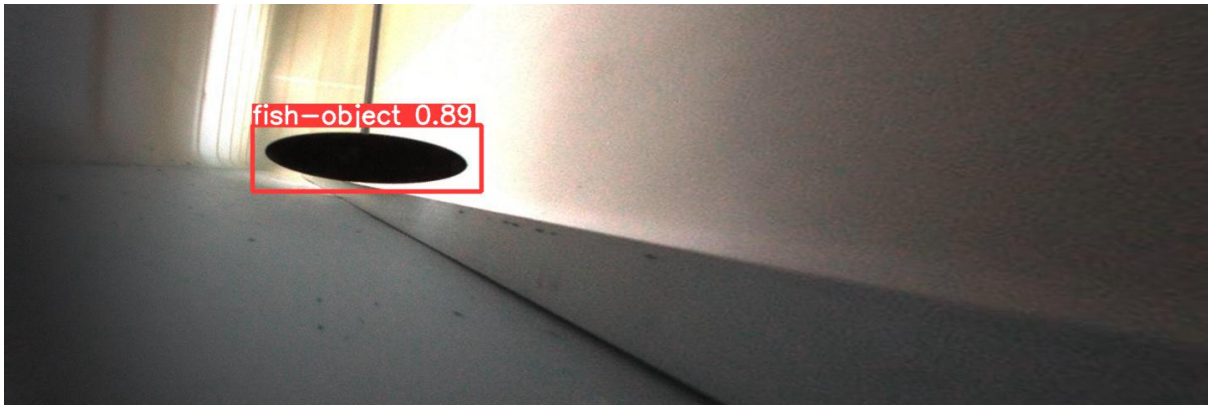
Figure 6.7.1 - 6.7.2 shows the detection results on rectified images from the data gathered in section 4. They are presented with the left image to make the results easier to see for the reader. The left image is presented first in the image pair, then the right image. YOLOv5 draws a bounding box around the object with a confidence. In figure 6.7.1 and 6.7.1, the network is able to detect the object in different distances. There is only one object in the scene, making it less complicated for the network to detect the object with a high confidence. The results reflect that the number of images in the training set is sufficient for this scenario.

Figure 6.7.2 and 6.7.2 shows detection results in an environment with multiple objects. Out of the many objects, only a couple of them is detected. The detection reflects the data labelling process since a maximum of five fish were labelled, focusing on fish that were "alone". Despite the limitation, the network can detect most fish with varying confidence. This means that the training set was not large enough to make the network produce results with high confidence. More representations of the fish should have been included in the training set. One example where the network struggles to distinguish between two fish can be seen in figure 6.7.2c. The network manages to draw an individual bounding box around the two fish. However, it also draws an additional bounding box around both fish since it believes it is one fish. It is worth noting that the two correct bounding boxes have a confidence of 0.70 and 0.68, however, the extra bounding box has a confidence of 0.43. It does not perform as well as on the previous dataset because of the extra objects in the scene and the fish have a more complex behavior.

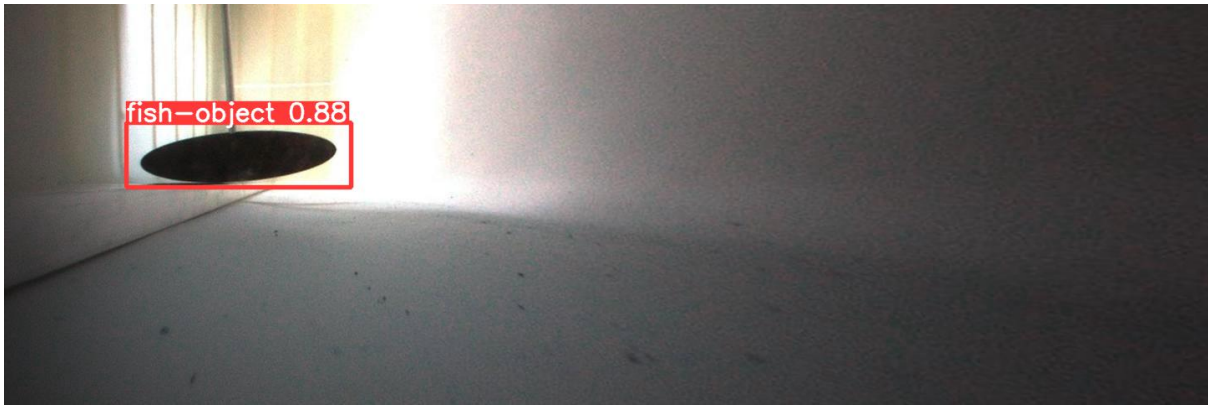


(a)

(b)



(c)



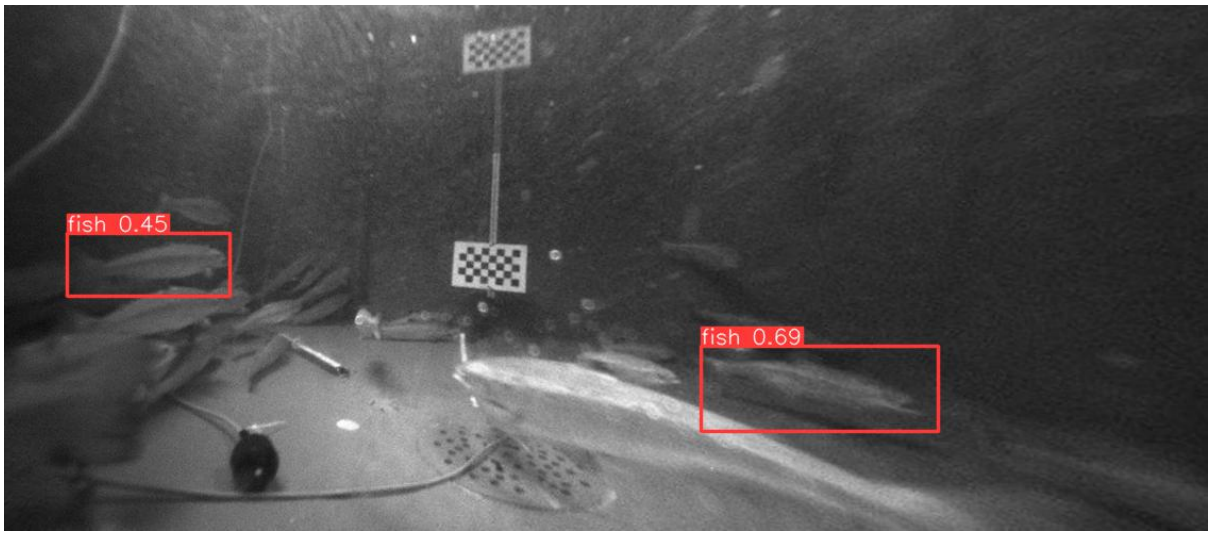
(d)

Figure 6.7.1: YOLOv5 applied on dataset from the lab at NTNU.

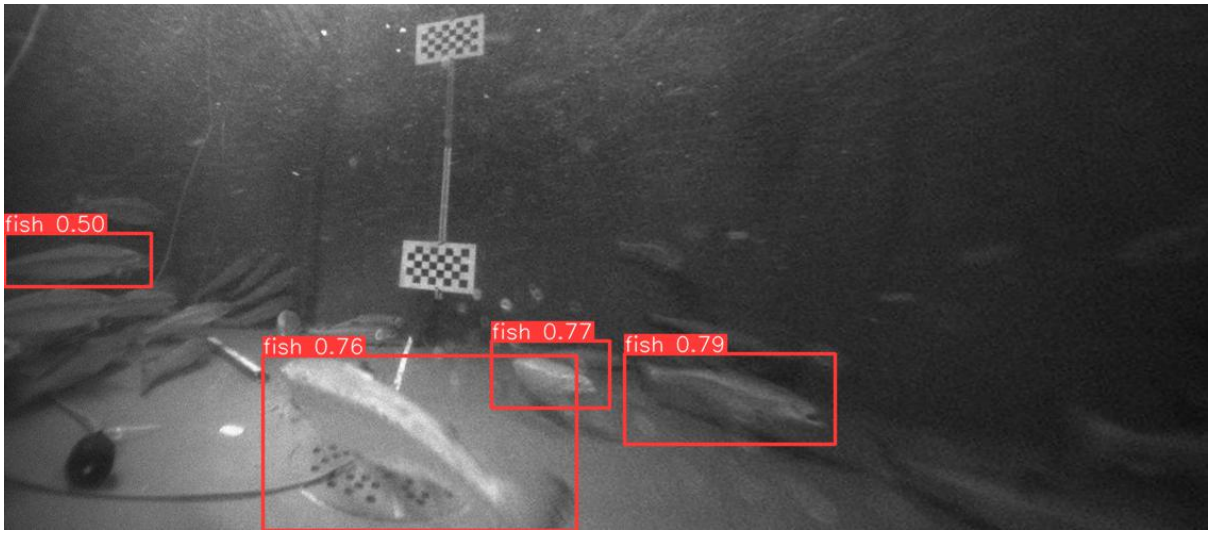
(e)

(f)

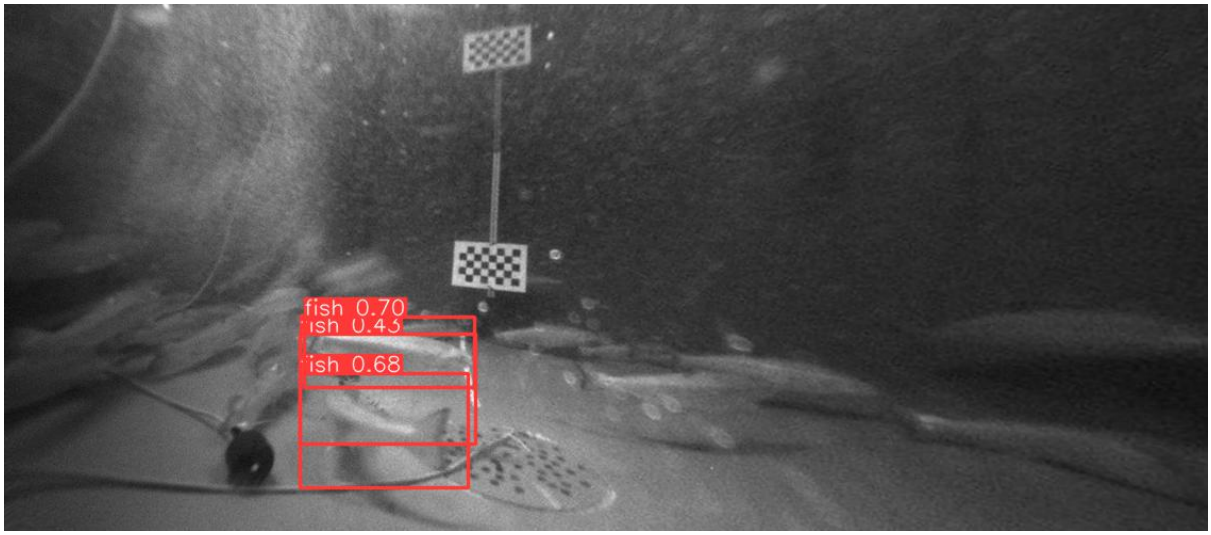
Figure 6.7.1: Detection results on dataset from the lab at NTNU.



(a)

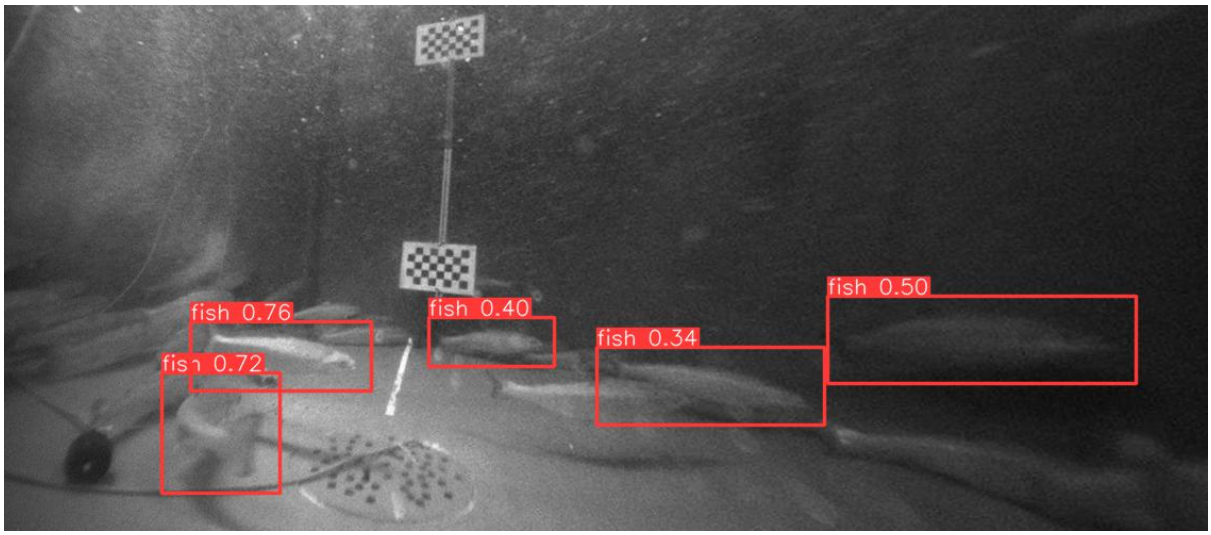


(b)

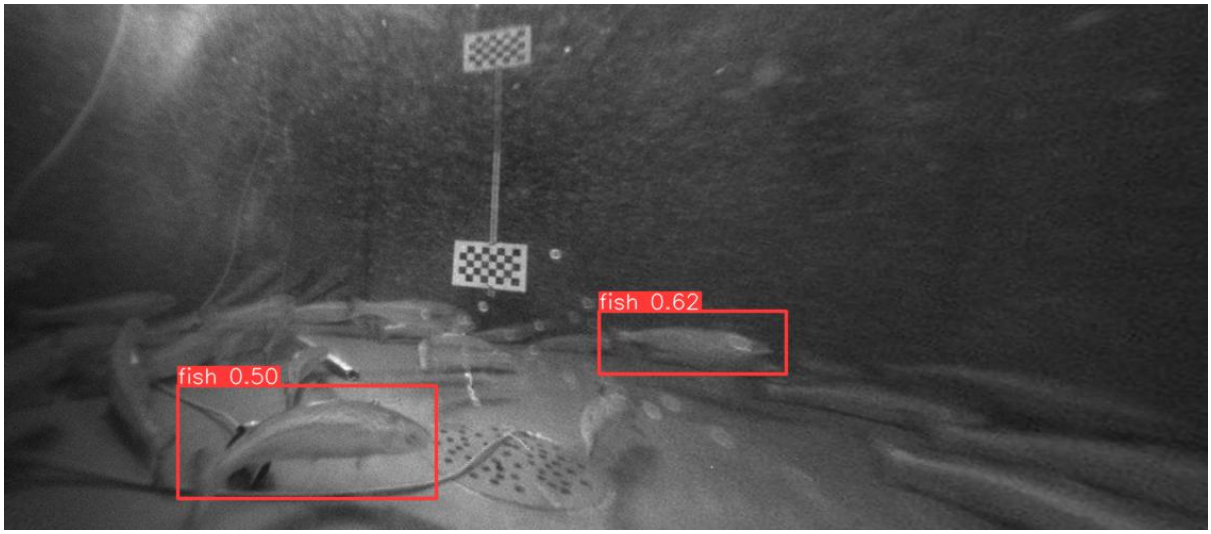


(c)

Figure 6.7.2: Detection results on dataset from SINTEF.



(d)



(e)

(f)

Figure 6.7.2: Detection results on dataset from SINTEF.

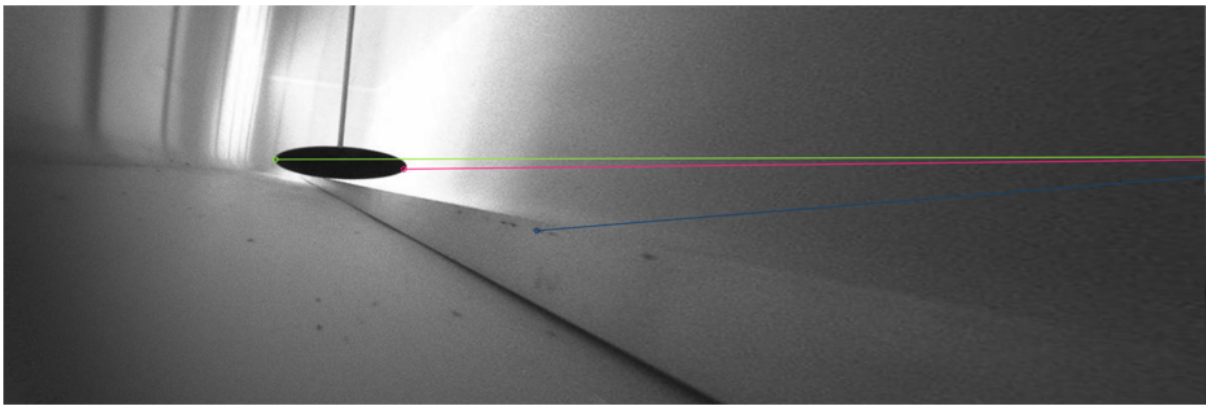
6.8 Superglue

The object detection network solves the detection problem. Point correspondences must be established to perform triangulation and calculate the distance and the velocity of the detected fish. These correspondences were established by using a pretrained model of Superglue [3] [92]. Superglue is a neural network which matches two sets of local features. It finds point correspondences while rejecting non-matchable points at the same time. Superglue assigns the matches by estimating a differentiable optimal transport problem, whose costs are predicted by a graph neural network [93].

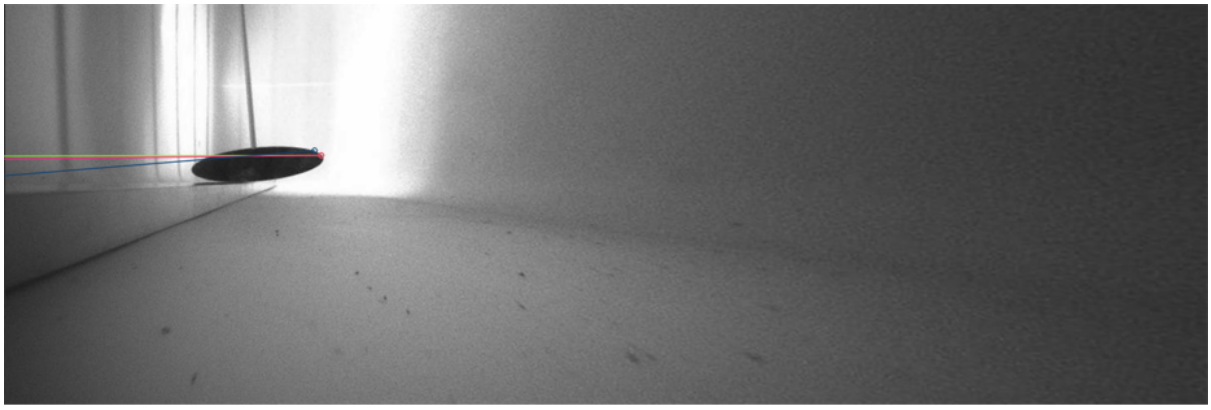
Superglue was pretrained on a dataset provided by the authors, such that training data was unnecessary for the model. It was compared against two popular traditional feature matching methods, ORB [38] and SIFT [36]. Figure 6.8.1 - 6.8.3 shows examples that demonstrate the performance of ORB, SIFT and Superglue. These examples are taken from different distances in the NTNU ITK lab to see how the methods perform when the object becomes smaller. The images are divided into left and right, where the first is left and the second is right, to make the feature matching more visible for the reader. The matching results also show matches from the whole scene. However, the main focus is the matching results on the ellipse object.

From the results, it is observable that Superglue outperforms the two selected methods. ORB struggles to find many matches, but it finds one correct match in all scenarios. SIFT finds significantly more matches but fails to find correct matches reliably. Similar to ORB, it finds one matching point on the left side of the object in figure 6.8.1 and 6.8.2. However, it seems that it mostly matches points outside the object. This observation is most noticeable in figure 6.8.2c where a light-blue point looks like it is located on the edge of the object, while figure 6.8.2d shows that it is a match to the environment. In figure 6.8.3, SIFT fails to match any point within the object, which makes it unreliable to use as a matching method. Looking at the results from Superglue, it not only manages to establish multiple matches but also matches them correctly, including the ones outside the object. In every scenario, Superglue robustly finds multiple matches on the ellipse object, proving that it is the best candidate.

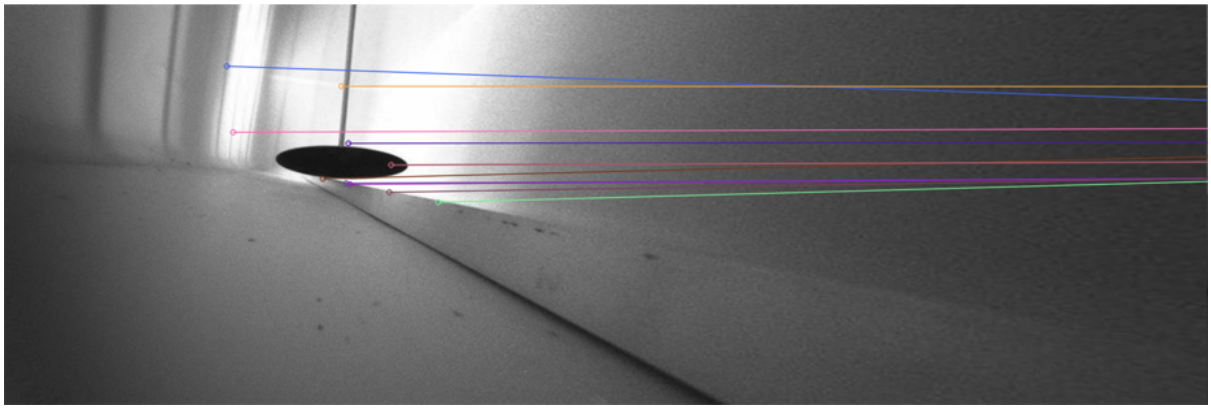
[3] SuperGlue GitHub



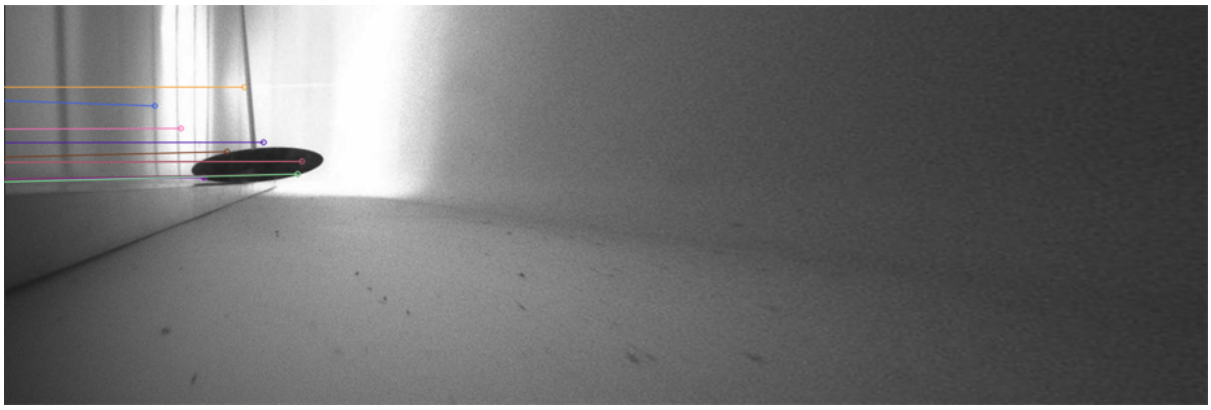
(a) ORB left



(b) ORB right

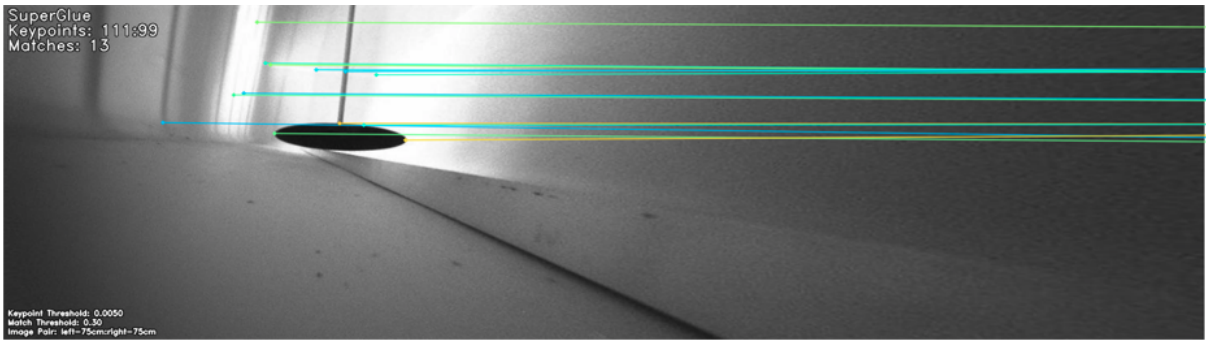


(c) SIFT left

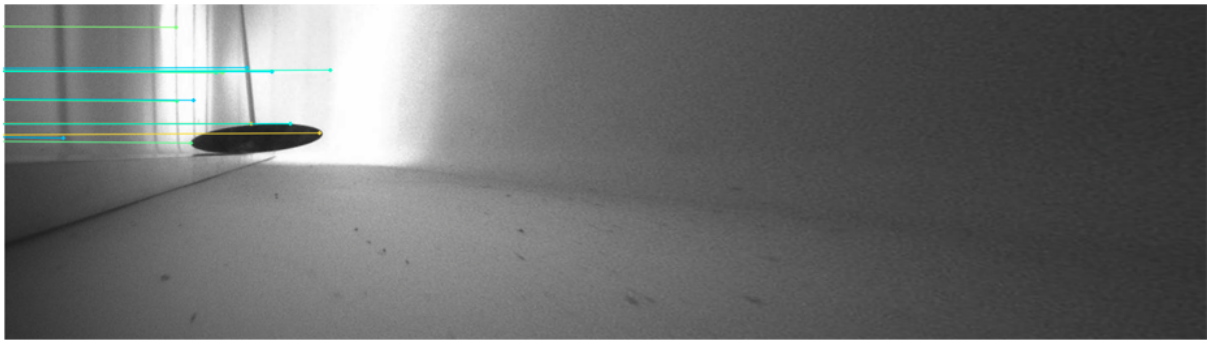


(d) SIFT right

Figure 6.8.1: Starting from the top, the images are illustrating matching results from ORB and SIFT on a 75cm distance.

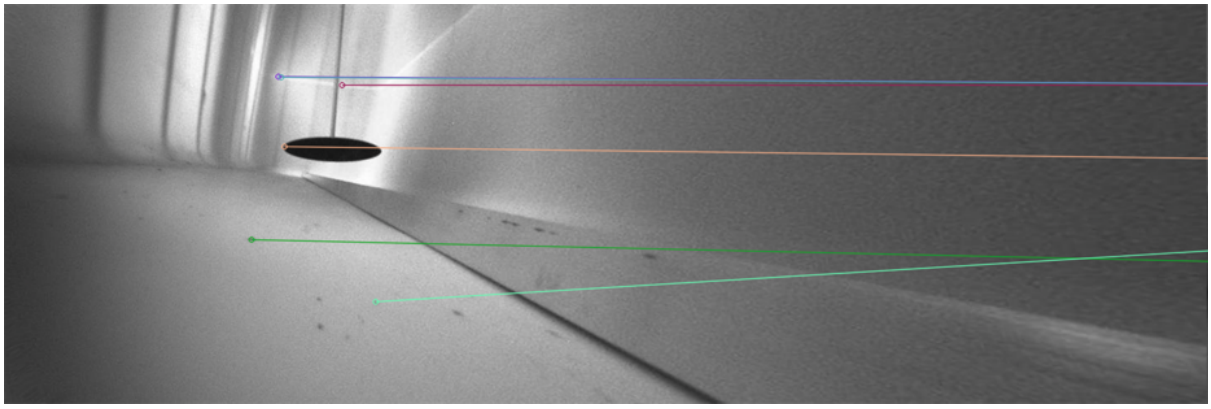


(e) Superglue left

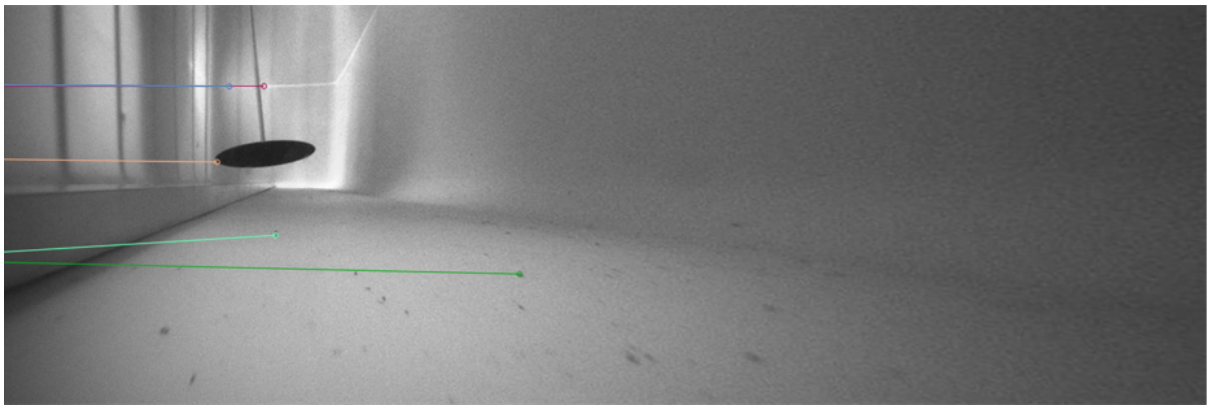


(f) Superglue right

Figure 6.8.1: Results from Superglue matching on 75 cm distance.

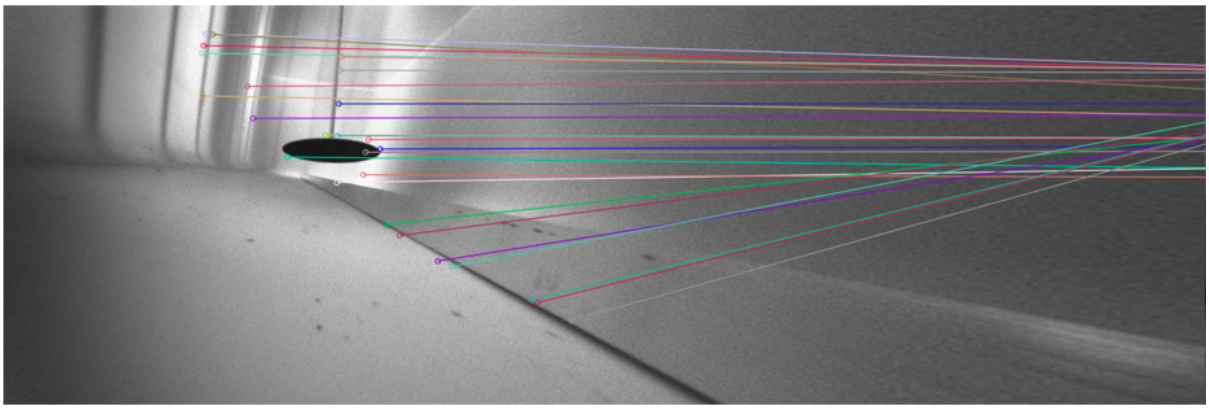


(a) ORB left



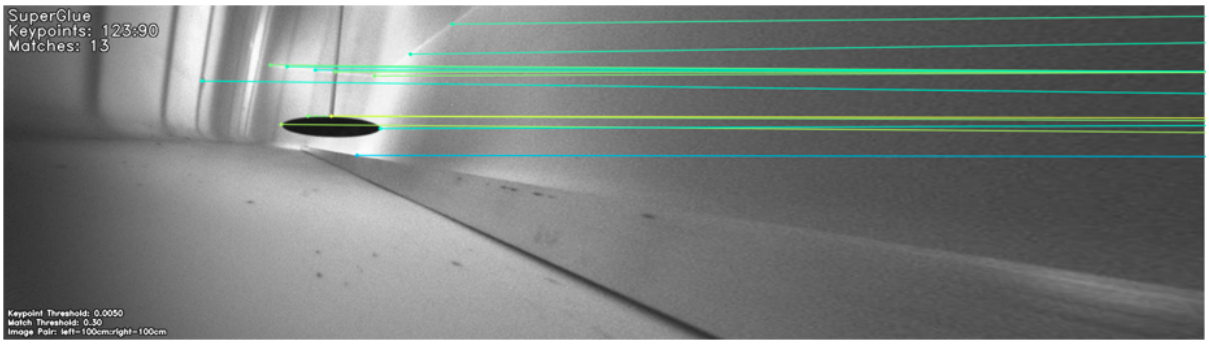
(b) ORB right

Figure 6.8.2: Results from ORB matching on a 100 cm distance.



(c) SIFT left

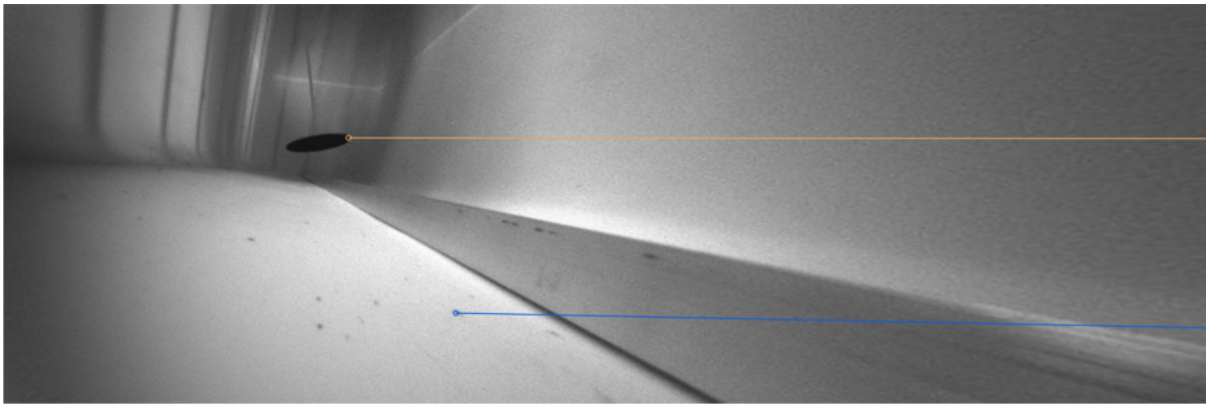
(d) SIFT right



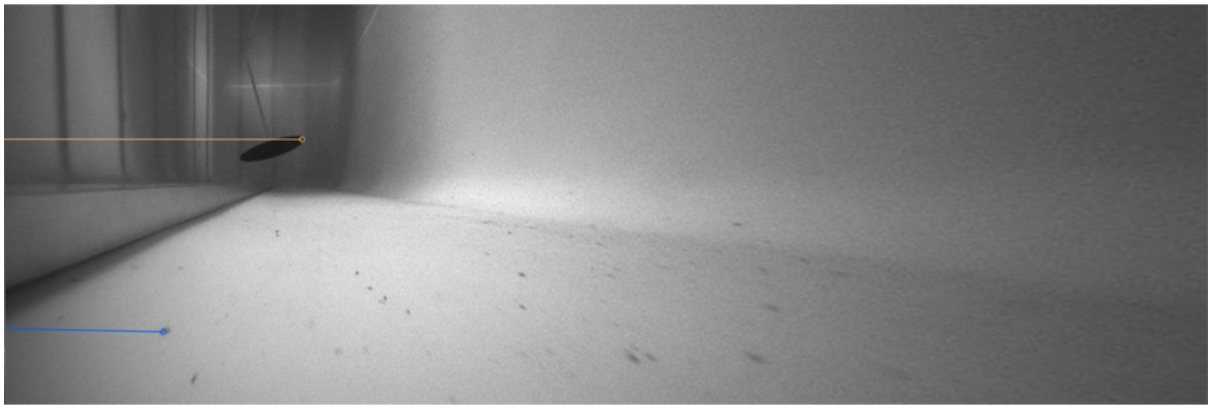
(e) Superglue left

(f) Superglue right

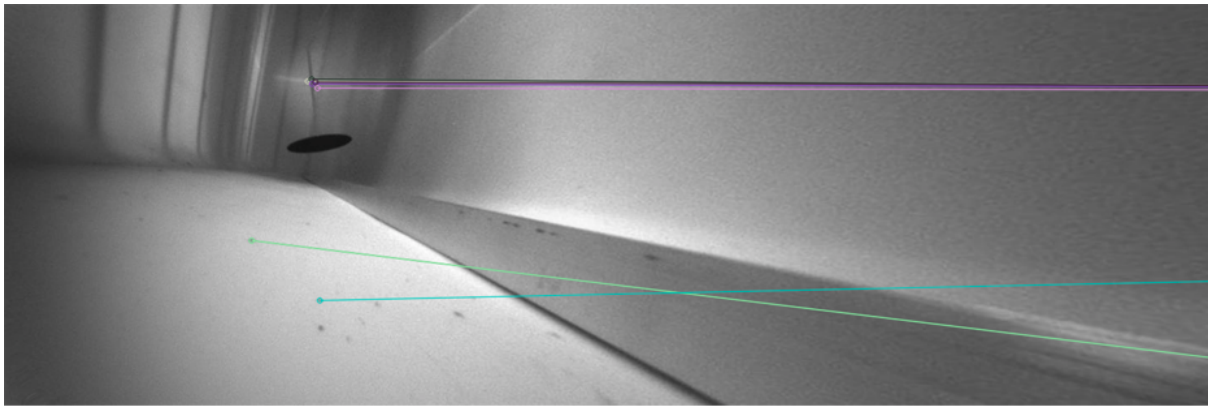
Figure 6.8.2: Starting from the top, the figure is showing results from SIFT and Superglue matching at 100cm distance.



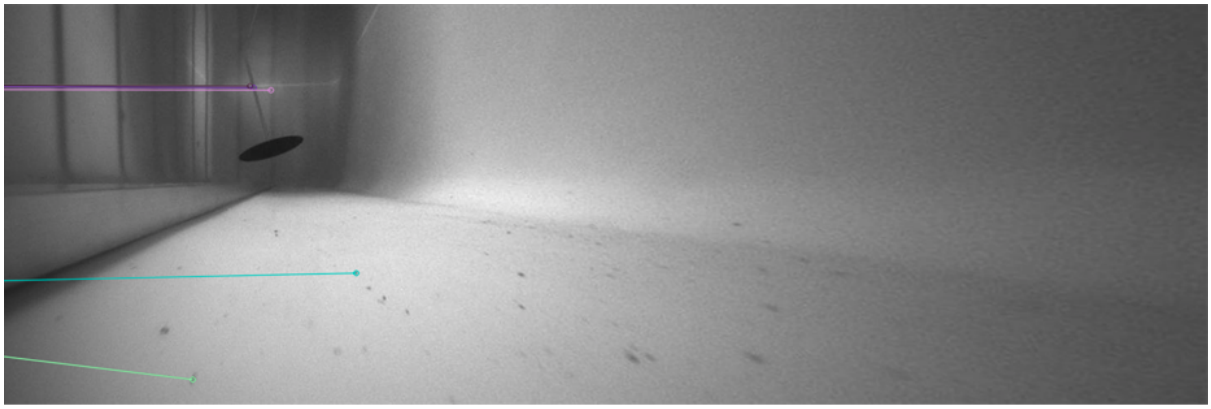
(a) ORB left



(b) ORB right

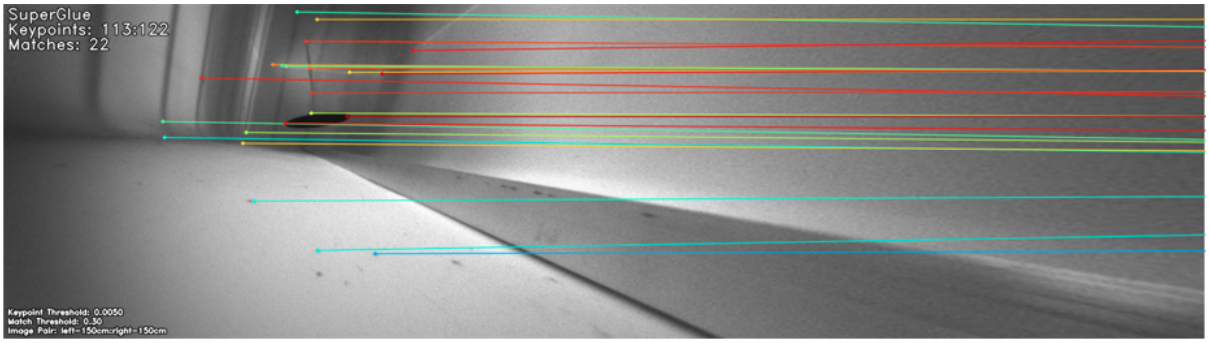


(c) SIFT left



(d) SIFT right

Figure 6.8.3: Starting from the top, the images are illustrating matching results from ORB and SIFT on a 150 cm distance.

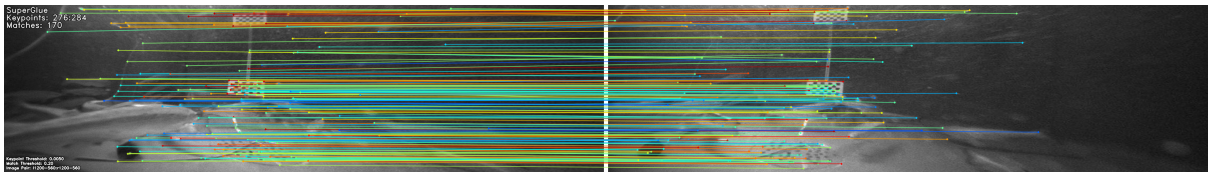


(e) Superglue left

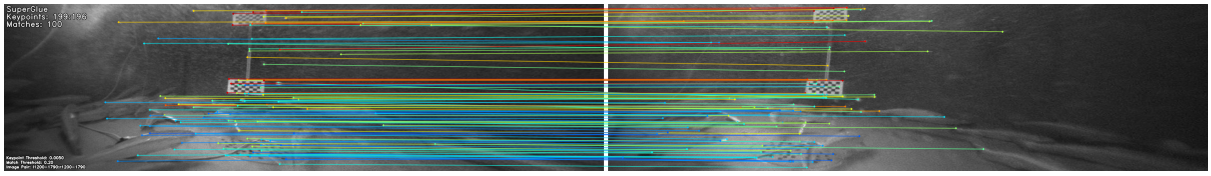
(f) Superglue right

Figure 6.8.3: Results from Superglue matching on 150 cm distance.

Two examples of feature matching with Superglue are shown in figure 6.8.4. It extracts features from all over the scene, making it difficult to evaluate the feature matching performance. Performing feature matching on the dataset from SINTEF makes it challenging to view whether the points are correctly matched or not. This is because the point correspondences are drawn with a line between the image pair. Hence, the performance of Superglue is purely based on the matching results from the NTNU ITK lab dataset. It provides a clear view of the matched points because of the reduced amount of objects in the scene.



(a)



(b)

Figure 6.8.4: Superglue feature matching applied on two image pairs from the SINTEF dataset.

CHAPTER 7

METHOD AND EXPERIMENTAL RESULTS

This chapter starts to give an overview of the suggested method. After providing the overview, detection results from the training of YOLOv5 are presented. This is followed by a validation of the distance measurement, validated on the NTNU ITK Lab dataset. Lastly, experimental tests on live fish footage from the SINTEF show the distance and velocity measurement results.

7.1 Overview of procedure

The flowchart of the suggested method is illustrated in figure 7.1.1.

The stereo camera starts by capturing an image sequence from the scene. Assuming that the cameras are calibrated, these images are rectified using MATLAB. After rectifying the images, YOLOv5 is applied on both left and right image sequences to detect the fish. When YOLOv5 has finished the detection, it draws bounding boxes around each detected fish. Superglue performs feature matching and draws a line between the matching points in the image pairs. The bounding boxes are used as a criterion to avoid including matches from other objects in the scene. If the point correspondences lie within the left and right bounding box, these matching points are used in the triangulation process to calculate the 3D coordinates. After calculating all the 3D points belonging to the detected fish, the average of the 3D points is computed, and then the Euclidean distance is calculated by using equation (3.8.4). After the distance is computed, the velocity can be calculated by taking the distance of the current image minus the previous image over a sequence of images.

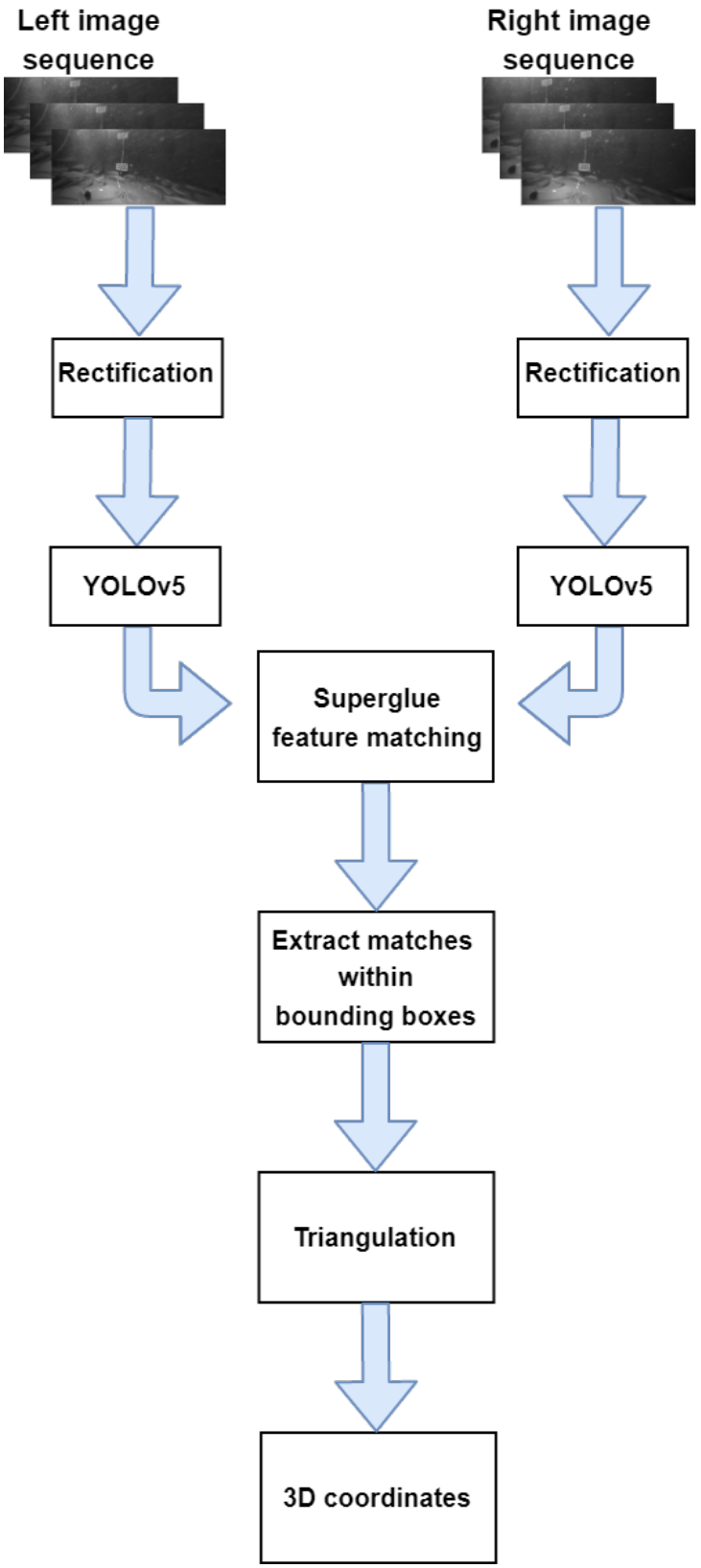
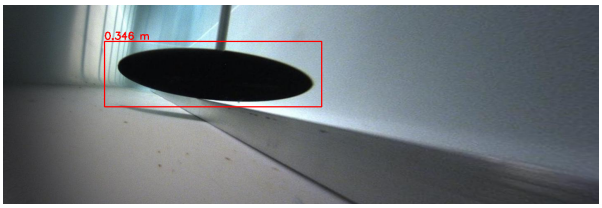


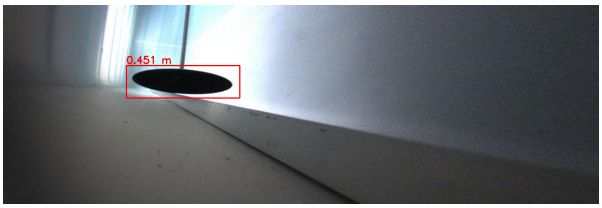
Figure 7.1.1: The flowchart of the suggested method

7.2 Validation with distance measurement

A testing setup was created in the NTNU ITK lab to validate the distance measurement accuracy. It was not possible to have ground truth available in the dataset from SINTEF. Hence, the NTNU ITK lab was used as a performance test. Ground truth was created using a long aluminium plate, measured and marked with different distances from 25 cm to 175 cm. Figure 7.2.1 shows the detection of the object, attached with the distance measurement, and table 7.2.1 shows an overview of the results with the error percentage. The object might have some offset from the ground truth since the object was manually held from above the water. Another factor that can cause an error in the measurement is the number of point correspondences detected. This is because the distance is calculated based on the average of the 3D points. The distances 50 cm to 175 cm has an error of sub 10%, which is sufficient. However, 25 cm has a significantly higher error percentage than the other distances. A figure of the feature matching result is attached in figure 7.2.2 to evaluate the possible error in this scenario. In the top image, Superglue finds two correct matches located on the top and right sides of the object. Since the object is outside the area of the right camera, it fails to find features on the left side of the object. Below, the left and the right image shows the bounding box predictions. The matches are both located within the bounding boxes, which means they are included in the distance computation. Based on this, it should give a lower error percentage. However, it can be challenging to see whether the object is placed correctly on the measured ground truth when submerging objects. The possible explanation for the significant error may be that the object was located further away from the ground truth, leading to an incorrect measurement due to human error.



(a) A measurement of 34.6 cm at 25 cm



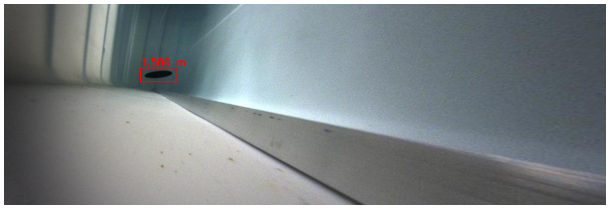
(b) A measurement of 45.1 cm at 50 cm

(c) A measurement of 78.8 cm at 75 cm

(d) A measurement of 96.9 cm at 100 cm

(e) A measurement of 133.7 cm at 125 cm

(f) A measurement of 147.5 cm at 150 cm



(g) A measurement of 158.6 cm at 175 cm

Figure 7.2.1: Results of distance measurement seen from left camera. The bounding boxes have been customized such that they display the distance instead of the confidence generated by YOLOv5.

Table 7.2.1: Table showing the results between ground truth and measured distance.

Ground truth	Measured distance	Error
$0.250m$	$0.346m$	$\pm 38.4\%$
$0.500m$	$0.451m$	$\pm 9.80\%$
$0.750m$	$0.788m$	$\pm 5.06\%$
$1.000m$	$0.969m$	$\pm 3.10\%$
$1.250m$	$1.337m$	$\pm 6.96\%$
$1.500m$	$1.475m$	$\pm 1.67\%$
$1.750m$	$1.586m$	$\pm 9.37\%$

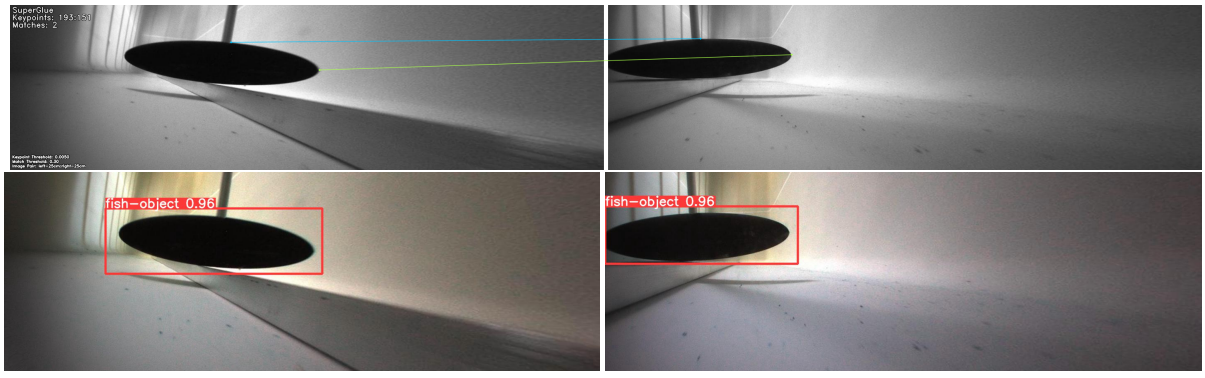
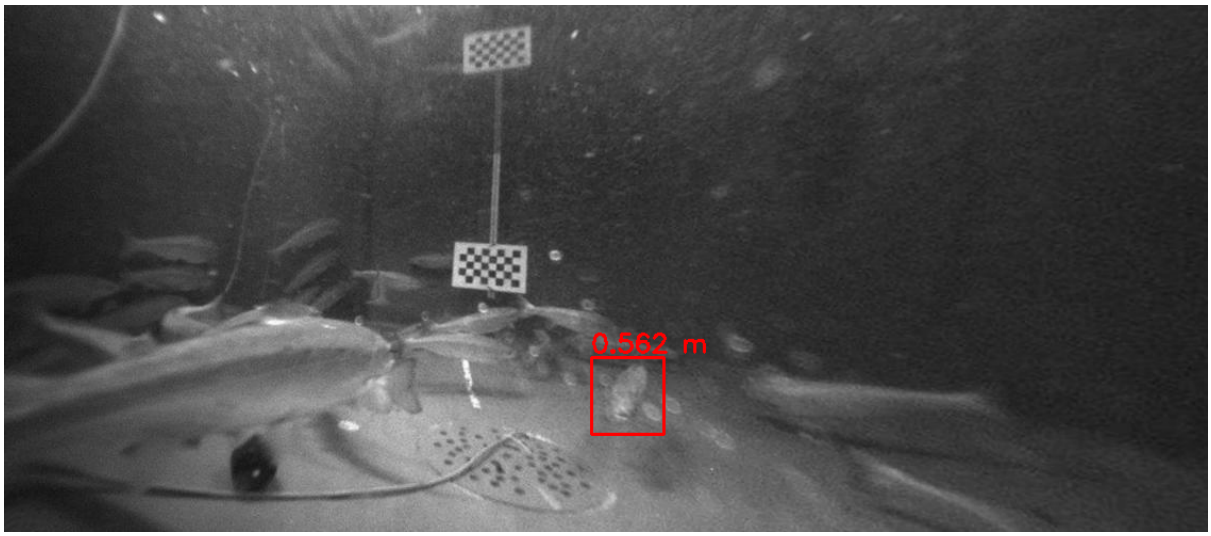


Figure 7.2.2: Result of detection and feature extraction on 25 cm distance.

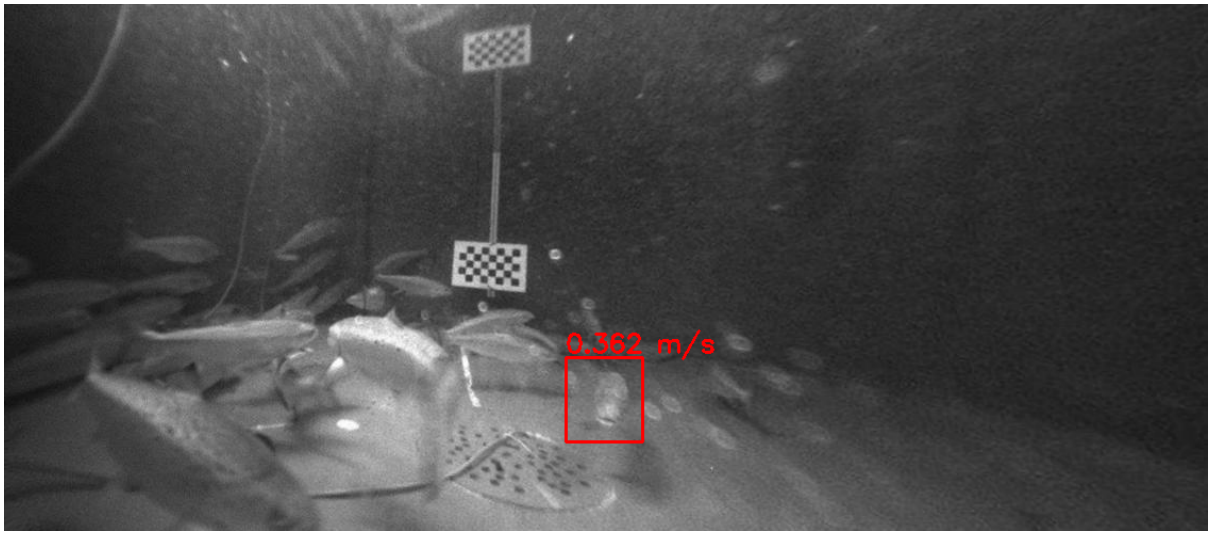
7.3 Experiments on dataset from SINTEF

The suggested method was applied to the dataset from SINTEF. It was noticed during the experimental tests that YOLO failed to detect the fish when the image sequence was too long. The failed detection could be a result of the complex movement of the fish and a small training dataset. Hence, the image sequence consisted of ten images, limited to one fish. Each image sequence is reduced to two images in the visualization since they occupy a lot of paper space. The large image size is necessary in order for the distance and speed measurements in the images to be legible for the reader. The first image of the sequence is presented along with the final image in the sequence. The first image initializes the fish with the distance of the fish, and the second image contains information on the velocity from the previous to the current image number.

In figure 7.3.1, the fish was faced toward the camera and moved in the same direction during the whole sequence. The detection network drew a good bounding box around the detected fish. The bounding box did not overlap with other objects with features, which means that it included the correct matches. Based on the direction it was swimming, the distance of the fish should decrease over time. The measurements are shown in figure 7.3.2. It shows the distance and the speed of the fish over the sequence. Considering that the validation from table 7.2.1 shows that there was some error in the measurements, the distance has some small spikes, similar to the other experiments. An explanation could be that the feature matching mainly found matches from the tail and the head. The fish have a certain length, and since the distance was calculated by the average of all 3D points detected, there can be errors in the measurements. The head and tail were often matched during the experiments, while in some samples, only the head or tail was matched. Despite this, the graph shows a decreasing trend at the end of the sequence. The method provided good results in this sequence even though there was no ground truth available to verify this. Assuming that the distance was correct, the fish showed a varying speed throughout the sequence.

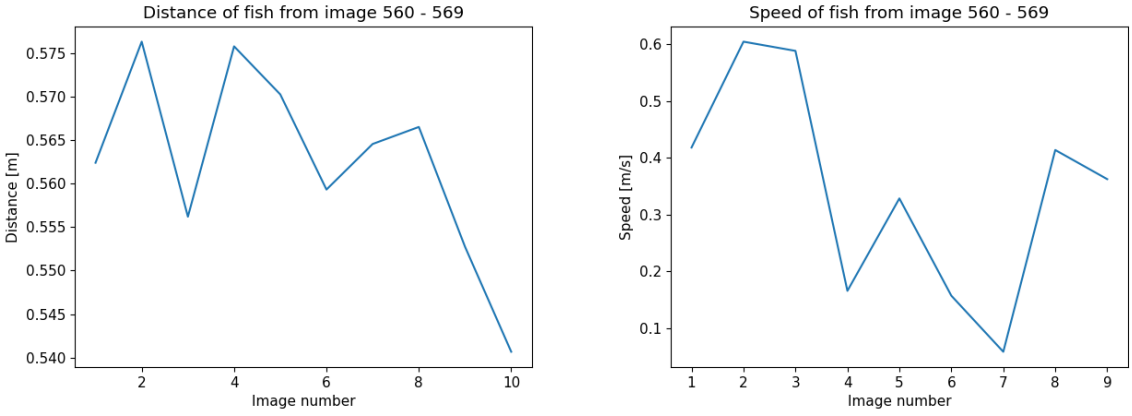


(a) Image 560



(b) Image 569

Figure 7.3.1: Images 560 - 569 showing the detection from the left camera. The figure shows the distance to the fish in the first image of the sequence, then shows the speed from image 568 to 569.

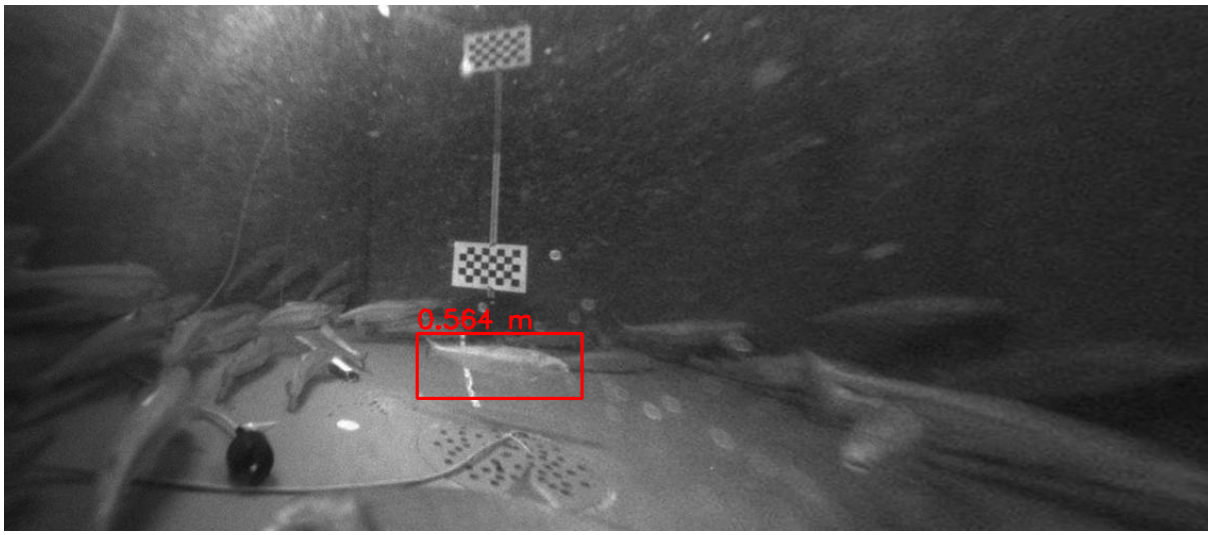


(a) Distance estimated from image 560 - 569

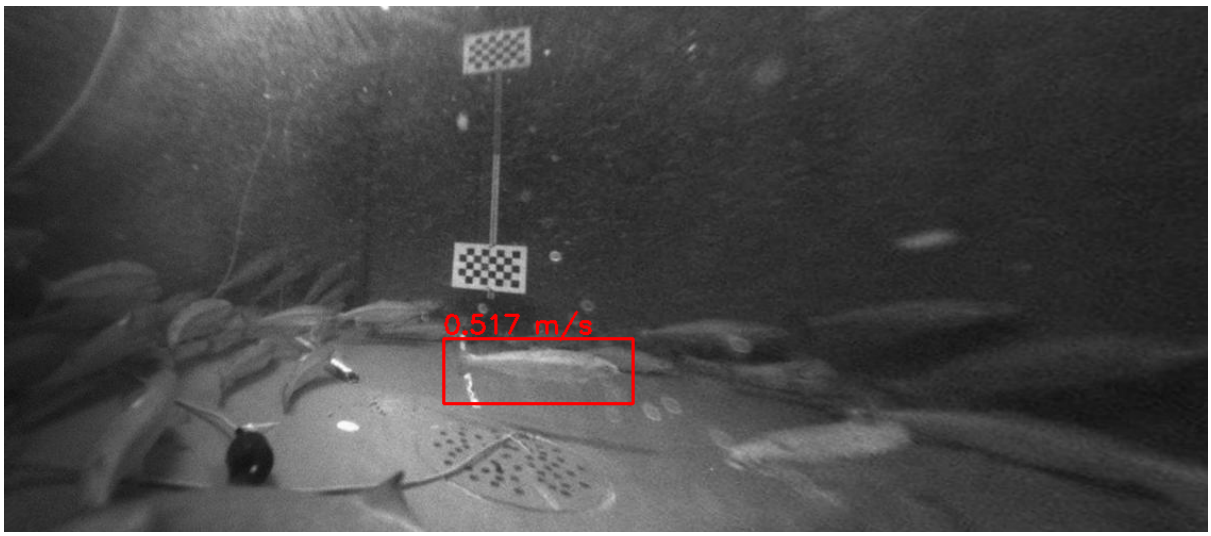
(b) Speed estimated from image 560 - 569

Figure 7.3.2: Measurements from image 560 - 569

The image sequence in figure 7.3.3 shows a fish that swims sideways. The detection was successful as it did not contain other objects of interest, such that outlier matches were not included in the calculations. Since the fish was swimming sideways, the distance to the fish should stay the same. Figure 7.3.4 shows the distance and speed of the fish. The distance measurement varies slightly. It seems correct as the distance in the first and last images was approximately the same. An idea would be to increase the number of images to get a clear view of whether the distance changes or not.

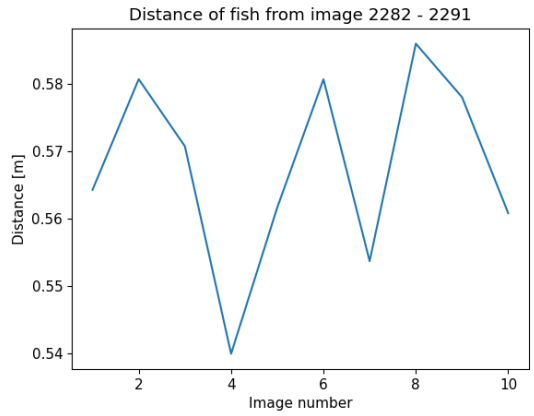


(a) Image 2282

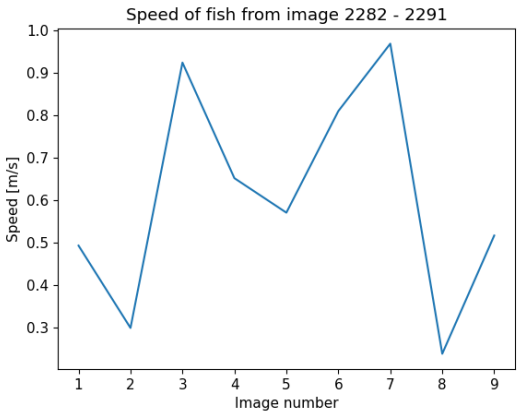


(b) Image 2291

Figure 7.3.3: Images 2282 - 2291 showing the detection from the left camera. The figure shows the distance to the fish in the first image of the sequence, then shows the speed from image 2290 to 2291



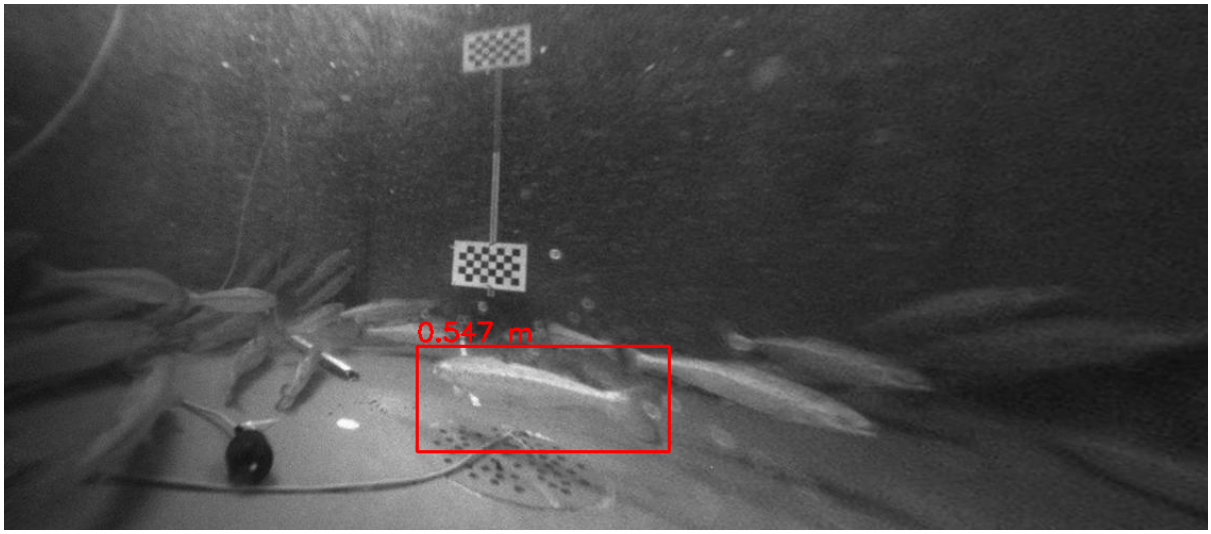
(a) Distance estimated from image 2282 - 2291



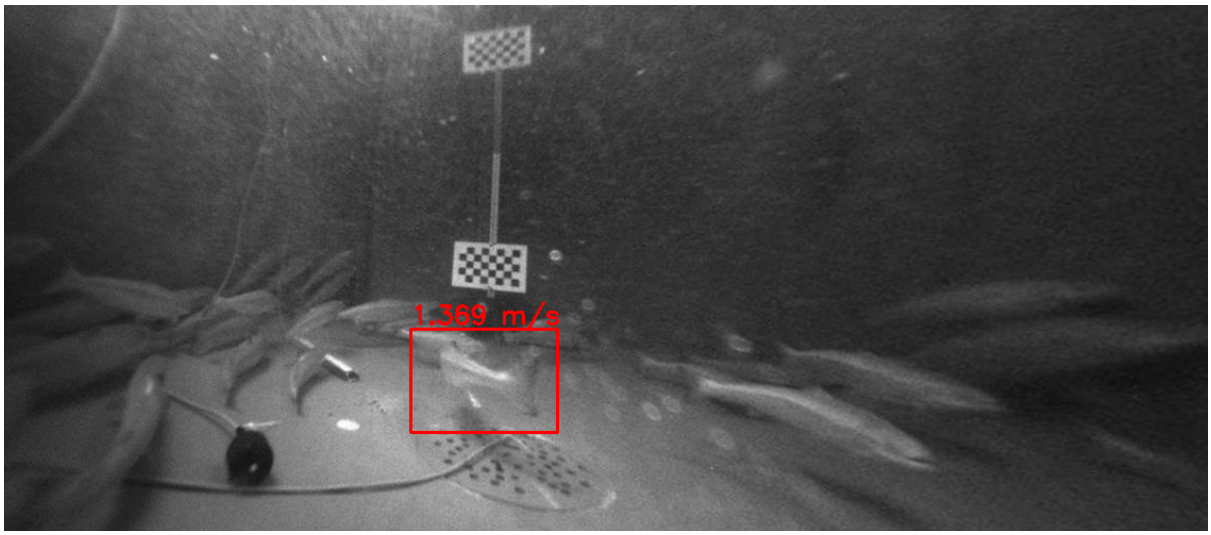
(b) Speed estimated from image 2282 - 2291

Figure 7.3.4: Measurements from image 2282 - 2291

Figure 7.3.5 shows results from image 2322 to 2330, which contains nine images instead of ten images. Superglue failed to extract matches from the detected fish in the last image pair during this test. To extract the values outputted by the method on a sequence of images was time consuming. Because of this, the sequence was included in the experiment results. It still shows valuable information about the method. The explanation might be that Superglue was pretrained in a different environment. Considering this, it still provided good results. The detected fish started in one direction but turned and swam away from the camera during the sequence. Based on this, the distance of the fish increased over time. In figure 7.3.5b, the drawn bounding box slightly overlapped with another fish in the background. It might be possible that some outlier matches are included in the measurement. There are no larger changes in the distance measurements compared to the previous sequences. Figure 7.3.6a shows the distance measurement over time, which shows a positive trend with the increased distance in the last image. The speed from frame to frame is shown in figure 7.3.6b. Similar to previous sequences, extra images should have been included to view the distance changes better.

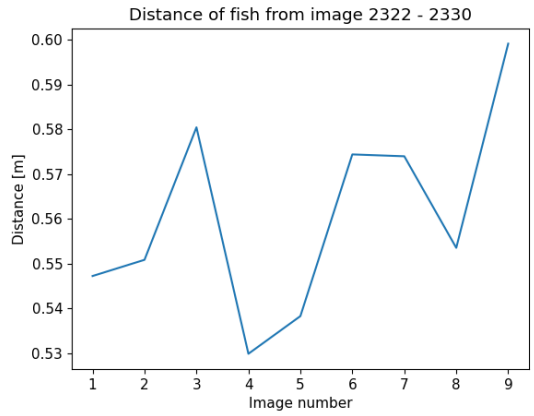


(a) Image 2322

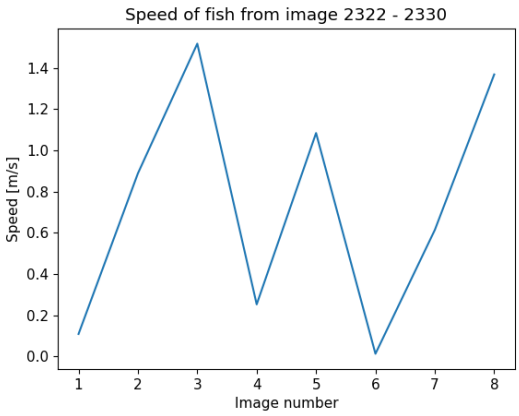


(b) Image 2330

Figure 7.3.5: Images 2322 - 2230 showing the detection from the left camera. The figure shows the distance to the fish in the first image of the sequence, then shows the speed from image 2329 to 2330



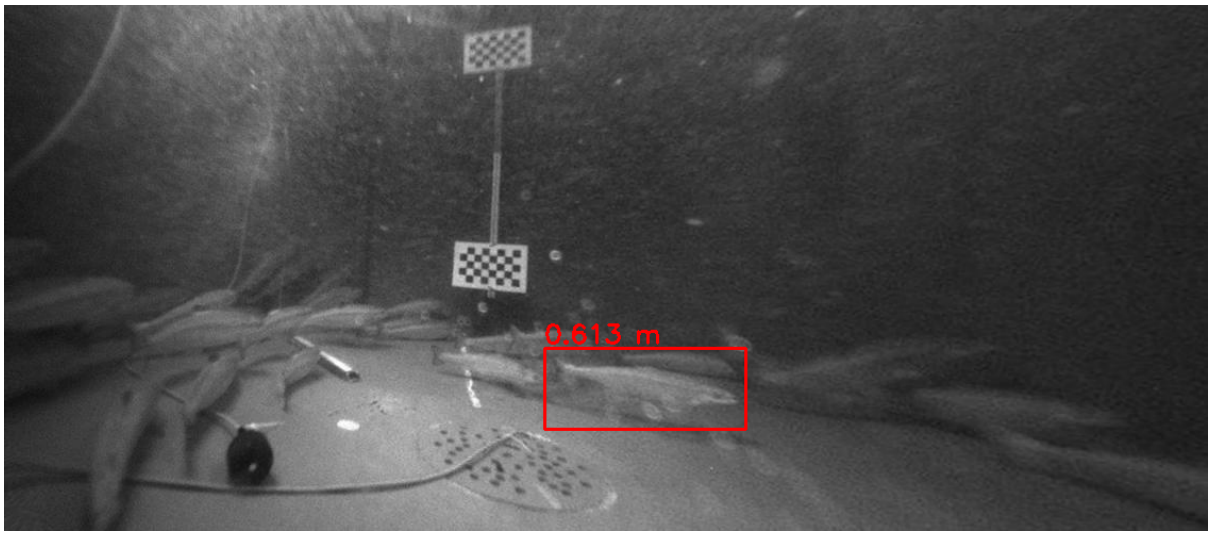
(a) Distance estimated from image 2322 - 2330



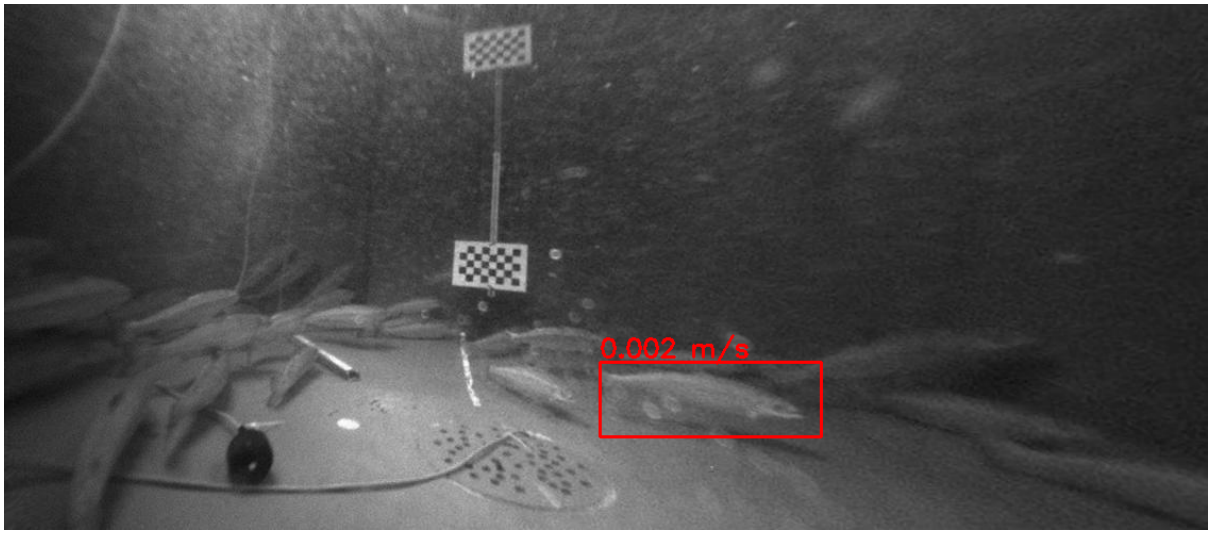
(b) Speed estimated from image 2322 - 2330

Figure 7.3.6: Measurements from image 2322 - 2330

The last image sequence shown in figure 7.3.7 detected a fish that swam sideways and slightly towards the camera. The network drew a bounding box overlapping with a fish in the background. However, the distance measurements found in figure 7.3.8a show that the overlap did not interfere with the measurements. This means that the correct matches were included in the computations. Since the fish is swimming slightly closer to the camera, the distance decreases slightly over time, which is shown in figure 7.3.8a. Figure 7.3.8b shows the speed of the fish over the sequence of images. According to the graph, the fish stays in the same place at the end of the sequence.

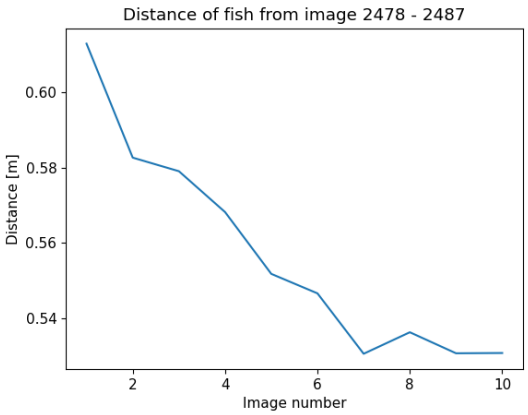


(a) Image 2478

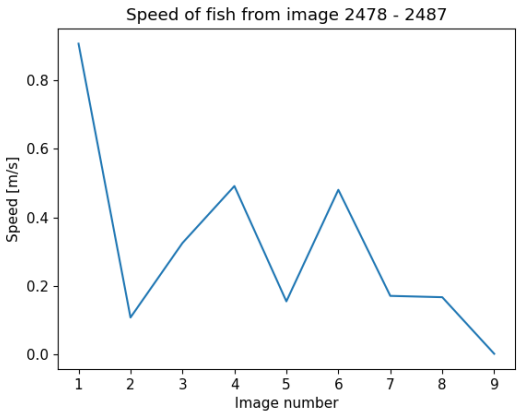


(b) Image 2487

Figure 7.3.7: Images 2478 - 2487 showing the detection from the left camera. The figure shows the distance to the fish in the first image of the sequence, then the speed from image 2486 to 2487



(a) Distance estimated from image 2478 - 2487



(b) Speed estimated from image 2478 - 2487

Figure 7.3.8: Measurements from image 2478 - 2487

CHAPTER 8

CONCLUSION AND FUTURE WORK

This thesis aimed to explore the usage of camera sensors, computer vision techniques and deep learning algorithms to measure fish-population parameters, such as distance and swim speed. The main contribution of this thesis is a proposed method for approximating the distance and speed of fish by applying YOLOv5 to detect the fish and Superglue to establish point correspondences on video captured by a stereo camera. These point correspondences are used for triangulation to reconstruct the 3D points used to calculate the Euclidean coordinates. The average of these coordinates was used to calculate the distance and speed of the object. YOLOv5 was trained on 1778 images, which was enough to establish good bounding boxes around the fish but with varying confidence. A pretrained Superglue produced significantly better results than the traditional methods: ORB and SIFT. The left and right bounding boxes were used as a criterion to remove outlier point correspondences, which contributes to robustly measuring distance and speed with minimal noise from other objects of interest. The distance measurement was validated with ground truth available in a controlled environment. It gave promising results on a single object with sub 10% error with the exception of one measurement, which most likely was due to a human error. After the validation, the method was tested on image sequences in a more realistic environment containing multiple fish. These image sequences contained ten images because YOLO lost detection when the image sequence was too long. The experimental test gave spiky numbers on the distance and speed measurements, which is to be expected when the validation showed errors in the measurements. Another reason that could contribute to the spiky results was the feature matching. The feature matching sometimes matched the head and the tail, while in some cases, it matched only the head or the tail of the fish, which might have caused the distance and speed measurements to vary. Considering that the image sequences contained ten images, the proposed approach gave promising results as the measurements followed the movement of the fish to some extent. The length of the image sequences should be increased in future work to get a better overview of how the algorithm follows the fish for a more extended period. The lost detections can be solved by increasing the training dataset, which trains the detection network to learn from many representations of the fish.

Furthermore, the detection network was trained to detect the entire fish. This was because the fish could be distorted or blurred, which made it difficult to see the texture of the fish. Another idea for future work would be to train the detector to detect either the head or the tail of the fish. This may contribute to more robust distance, and speed measurements since the matches from the tail or head might be lost in some scenarios.

Finally, a tracking method should be implemented in future work. It is essential if this method were to be incorporated into a ROV. A tracking algorithm will contribute to automatically detecting the same object over a sequence of images. In future work, a tracking method should be handpicked based on the literature review that was conducted in this thesis. Deep learning approaches for tracking should be considered as they are state-of-the-art. A downside of deep learning methods is that they require a lot of data to give the best performance. If deep learning methods are problematic to incorporate into the proposed method, traditional algorithms, such as optical flow or indirect methods, can also be good alternatives. However, traditional

approaches for tracking might struggle in underwater environments because of the noisy images.

BIBLIOGRAPHY

[1] K. C. R. Jin, "Specialization project: Deep-learning algorithms for estimation of fish-population parameters from sonar- and video data," 2021.

[2] E. Kelasidi, "Race fish-machine interaction," 05 2020. [Online]. Available: https://www.sintef.no/prosjekter/2020/race-fish-machine-interaction/

[3] X. Yang, S. Zhang, J. Liu, Q. Gao, S. Dong, and C. Zhou, "Deep learning for smart fish farming: applications, opportunities and challenges," *Reviews in Aquaculture*, vol. 13, no. 1, pp. 66–90, 2021.

[4] H. V. Bjelland, M. Føre, P. Lader, D. Kristiansen, I. M. Holmen, A. Fredheim, E. I. Grøtli, D. E. Fathi, F. Oppedal, I. B. Utne *et al.*, "Exposed aquaculture in norway," in *OCEANS 2015-MTS/IEEE Washington*. IEEE, 2015, pp. 1–10.

[5] M. Føre, K. Frank, T. Norton, E. Svendsen, J. A. Alfredsen, T. Dempster, H. Eguiraun, W. Watson, A. Stahl, L. M. Sunde *et al.*, "Precision fish farming: A new framework to improve production in aquaculture," *biosystems engineering*, vol. 173, pp. 176–193, 2018.

[6] H. Aasjord and I. Geving, "Accidents in norwegian fisheries and some other comparable norwegian industries," in *Proc. IFISH4. Fourth International Fishing Industry Safety & Health Conference*, 2009.

[7] F. Oppedal, T. Dempster, and L. H. Stien, "Environmental drivers of atlantic salmon behaviour in sea-cages: a review," *Aquaculture*, vol. 311, no. 1-4, pp. 1–18, 2011.

[8] H. B. Amundsen, W. Caharija, and K. Y. Pettersen, "Autonomous rov inspections of aquaculture net pens using dvl," *IEEE Journal of Oceanic Engineering*, vol. 47, no. 1, pp. 1–19, 2021.

[9] M. Bjerkeng, T. Kirkhus, W. Caharija, J. T Thielemann, H. B Amundsen, S. Johan Ohrem, and E. Ingar Grøtli, "Rov navigation in a fish cage with laser-camera triangulation," *Journal of Marine Science and Engineering*, vol. 9, no. 1, p. 79, 2021.

[10] C. Noble, K. Gismervik, M. H. Iversen, J. Kolarevic, J. Nilsson, L. H. Stien, and J. F. Turnbull, "Welfare indicators for farmed rainbow trout: tools for assessing fish welfare," 2020. [Online]. Available: https://www.fhf.no/prosjekter/prosjektbasen/901157

[11] A. Dutta, A. Mondal, N. Dey, S. Sen, L. Moraru, and A. E. Hassanien, "Vision tracking: a survey of the state-of-the-art," *SN Computer Science*, vol. 1, no. 1, pp. 1–19, 2020.

[12] S. Hua, M. Kapoor, and D. C. Anastasiu, "Vehicle tracking and speed estimation from traffic videos," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2018, pp. 153–160.

[13] X. Qimin, L. Xu, W. Mingming, L. Bin, and S. Xianghui, "A methodology of vehicle speed estimation based on optical flow," in *Proceedings of 2014 IEEE International Conference on Service Operations and Logistics, and Informatics*. IEEE, 2014, pp. 33–37.

[14] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: A survey," *Acm computing surveys (CSUR)*, vol. 38, no. 4, pp. 13–es, 2006.

[15] R. Szeliski, *Computer vision: algorithms and applications 2nd edition*. Springer Science & Business Media, 2021. [Online]. Available: https://szeliski.org/Book/

[16] D. Fortun, P. Bouthemy, and C. Kervrann, "Optical flow modeling and computation: A survey," *Computer Vision and Image Understanding*, vol. 134, pp. 1–21, 2015.

[17] T. Senst, J. Geistert, I. Keller, and T. Sikora, "Robust local optical flow estimation using bilinear equations for sparse motion estimation," in *2013 IEEE International Conference on Image Processing*. IEEE, 2013, pp. 2499–2503.

[18] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[19] B. D. Lucas, T. Kanade *et al.*, "An iterative image registration technique with an application to stereo vision." Vancouver, British Columbia, 1981.

[20] M. J. Black and P. Anandan, "A framework for the robust estimation of optical flow," in *1993 (4th) International Conference on Computer Vision*. IEEE, 1993, pp. 231–236.

[21] ——, "The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields," *Computer vision and image understanding*, vol. 63, no. 1, pp. 75–104, 1996.

[22] T. Senst, V. Eiselein, and T. Sikora, "Robust local optical flow for feature tracking," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 22, no. 9, pp. 1377–1387, 2012.

[23] C. Tomasi and T. Kanade, "Detection and tracking of point," *Int J Comput Vis*, vol. 9, pp. 137–154, 1991.

[24] T. Senst, J. Geistert, and T. Sikora, "Robust local optical flow: Long-range motions and varying illuminations," in *2016 IEEE International Conference on Image Processing (ICIP)*. IEEE, 2016, pp. 4478–4482.

[25] J.-Y. Bouguet *et al.*, "Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm," *Intel corporation*, vol. 5, no. 1-10, p. 4, 2001.

[26] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 3354–3361.

[27] S. Baker, D. Scharstein, J. Lewis, S. Roth, M. J. Black, and R. Szeliski, "A database and evaluation methodology for optical flow," *International journal of computer vision*, vol. 92, no. 1, pp. 1–31, 2011.

[28] D. J. Butler, J. Wulff, G. B. Stanley, and M. J. Black, "A naturalistic open source movie for optical flow evaluation," in *European conference on computer vision*. Springer, 2012, pp. 611–625.

[29] J. Morat, F. Devernay, and S. Cornou, "Tracking with stereo-vision system for low speed following applications," in *2007 IEEE Intelligent Vehicles Symposium*. IEEE, 2007, pp. 955–961.

[30] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Scandinavian conference on Image analysis*. Springer, 2003, pp. 363–370.

[31] G. Farneback, "Very high accuracy velocity estimation using orientation tensors, parametric motion, and simultaneous segmentation of the motion field," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, vol. 1. IEEE, 2001, pp. 171–177.

[32] T. Kroeger, R. Timofte, D. Dai, and L. V. Gool, "Fast optical flow using dense inverse search," in *European Conference on Computer Vision*. Springer, 2016, pp. 471–488.

[33] S. Baker and I. Matthews, "Equivalence and efficiency of image alignment algorithms," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1. IEEE, 2001, pp. I–I.

[34] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. Van Der Smagt, D. Cremers, and T. Brox, "Flownet: Learning optical flow with convolutional networks," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2758–2766.

[35] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2462–2470.

[36] D. G. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the seventh IEEE international conference on computer vision*, vol. 2. Ieee, 1999, pp. 1150–1157.

[37] H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in *European conference on computer vision*. Springer, 2006, pp. 404–417.

[38] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in *2011 International conference on computer vision*. Ieee, 2011, pp. 2564–2571.

[39] T. Gao, G. Li, S. Lian, and J. Zhang, "Tracking video objects with feature points based particle filtering," *Multimedia Tools and Applications*, vol. 58, no. 1, pp. 1–21, 2012.

[40] X. Lu, T. Izumi, L. Teng, and L. Wang, "Particle filter vehicle tracking based on surf feature matching," *IEEJ Journal of Industry Applications*, vol. 3, no. 2, pp. 182–191, 2014.

[41] K. Mu, F. Hui, and X. Zhao, "Multiple vehicle detection and tracking in highway traffic surveillance video based on sift feature matching," *Journal of Information Processing Systems*, vol. 12, no. 2, pp. 183–195, 2016.

[42] J. Arróspide, L. Salgado, and M. Nieto, "Vehicle detection and tracking using homography-based plane rectification and particle filtering," in *2010 IEEE Intelligent Vehicles Symposium*. IEEE, 2010, pp. 150–155.

[43] H. Wang and C. Schmid, "Action recognition with improved trajectories," in *Proceedings of the IEEE international conference on computer vision*, 2013, pp. 3551–3558.

[44] H. Zhang, L. Xiao, and G. Xu, "A novel tracking method based on improved fast corner detection and pyramid lk optical flow," in *2020 Chinese Control and Decision Conference (CCDC)*. IEEE, 2020, pp. 1871–1876.

[45] K. Katija, P. L. Roberts, J. Daniels, A. Lapides, K. Barnard, M. Risi, B. Y. Ranaan, B. G. Woodward, and J. Takahashi, "Visual tracking of deepwater animals using machine learning-controlled robotic underwater vehicles," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 860–869.

[46] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[47] E. Bochinski, T. Senst, and T. Sikora, "Extending iou based multi-object tracking by visual information," in *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, 2018, pp. 1–6.

[48] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[49] Z. Cai, M. Saberian, and N. Vasconcelos, "Learning complexity-aware cascades for deep pedestrian detection," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 3361–3369.

[50] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*. IEEE, 2016, pp. 3464–3468.

[51] D. Bell, W. Xiao, and P. James, "Accurate vehicle speed estimation from monocular camera footage," in *XXIV ISPRS Congress*. Newcastle University, 2020.

[52] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in *2017 IEEE international conference on image processing (ICIP)*. IEEE, 2017, pp. 3645–3649.

[53] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," *arXiv preprint arXiv:1603.00831*, 2016.

[54] G. Ciaparrone, F. L. Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, "Deep learning in video multi-object tracking: A survey," *Neurocomputing*, vol. 381, pp. 61–88, 2020.

[55] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. Torr, "Fully-convolutional siamese networks for object tracking," in *European conference on computer vision*. Springer, 2016, pp. 850–865.

[56] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[57] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8971–8980.

[58] M. Kristan, J. Matas, A. Leonardis, M. Felsberg, L. Cehovin, G. Fernandez, T. Vojir, G. Hager, G. Nebehay, and R. Pflugfelder, "The visual object tracking vot2015 challenge results," in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 1–23.

[59] S. Cheng, B. Zhong, G. Li, X. Liu, Z. Tang, X. Li, and J. Wang, "Learning to filter: Siamese relation network for robust tracking," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 4421–4431.

[60] M. Kristan, A. Leonardis, J. Matas, M. Felsberg, R. Pflugfelder, L. ˇCehovin Zajc, T. Vojir, G. Hager, A. Lukezic, A. Eldesokey *et al.*, "The visual object tracking vot2017 challenge results," in *Proceedings of the IEEE international conference on computer vision workshops*, 2017, pp. 1949–1972.

[61] G. Bhat, M. Danelljan, L. V. Gool, and R. Timofte, "Learning discriminative model prediction for tracking," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6182–6191.

[62] M. A. Mahammed, A. I. Melhum, and F. A. Kochery, "Object distance measurement by stereo vision," *International Journal of Science and Applied Information Technology (IJSAIT)*, vol. 2, no. 2, pp. 05–08, 2013.

[63] P. Li, T. Qin *et al.*, "Stereo vision-based semantic 3d object and ego-motion tracking for autonomous driving," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 646–661.

[64] X. Chen, K. Kundu, Y. Zhu, A. G. Berneshawi, H. Ma, S. Fidler, and R. Urtasun, "3d object proposals for accurate object class detection," in *Advances in Neural Information Processing Systems*. Citeseer, 2015, pp. 424–432.

[65] A. Osep, W. Mehner, M. Mathias, and B. Leibe, "Combined image-and world-space tracking in traffic scenes," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1988–1995.

[66] J. Luiten, T. Fischer, and B. Leibe, "Track to reconstruct and reconstruct to track," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1803–1810, 2020.

[67] J. S. Aber, I. Marzolff, and J. Ries, *Small-format aerial photography: Principles, techniques and geoscience applications*. Elsevier, 2010.

[68] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[69] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 11, pp. 1330–1334, 2000.

[70] J. J. Høklie, "Master thesis: Passive depth estimation using stereo vision, an experimental study," 2017.

[71] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.

[72] H. Zhan, C. S. Weerasekera, J.-W. Bian, and I. Reid, "Visual odometry revisited: What should be learnt?" in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 4203–4210.

[73] M. Engebretsen, K. Gjerden, Ø. Utbjoe, and Våge, "Master thesis: Autonomous navigation, mapping, and exploration for underwater robots," 2020.

[74] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. A. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, 2018.

[75] S.-J. Lee, T. Chen, L. Yu, and C.-H. Lai, "Image classification based on the boost convolutional neural network," *IEEE Access*, vol. 6, pp. 12 755–12 768, 2018.

[76] M. W. Gardner and S. Dorling, "Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences," *Atmospheric environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[77] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[78] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Computers and electronics in agriculture*, vol. 147, pp. 70–90, 2018.

[79] "Tikz - neural networks," https://tikz.net/neural_networks/, accessed: 05.05.2022.

[80] "Deep learning vs. machine learning – the essential differences you need to know!" https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/, accessed: 05.05.2022.

[81] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[82] S. Kiranyaz, O. Avci, O. Abdeljaber, T. Ince, M. Gabbouj, and D. J. Inman, "1d convolutional neural networks and applications: A survey," *Mechanical systems and signal processing*, vol. 151, p. 107398, 2021.

[83] Y. Xiao, Z. Tian, J. Yu, Y. Zhang, S. Liu, S. Du, and X. Lan, "A review of object detection based on deep learning," *Multimedia Tools and Applications*, vol. 79, no. 33, pp. 23 729–23 791, 2020.

[84] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[85] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," in *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, 2021, pp. 13 029–13 038.

[86] P. Soviany and R. T. Ionescu, "Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction," in *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*. IEEE, 2018, pp. 209–214.

[87] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *arXiv*, 2018.

[88] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[89] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems*, vol. 29, 2016.

[90] M. Everingham, S. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.

[91] G. Jocher, "Yolov5," https://github.com/ultralytics/yolov5, 2022.

[92] P.-E. Sarlin, D. DeTone, T. Malisiewicz, and A. Rabinovich, "Superglue: Learning feature matching with graph neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 4938–4947.

[93] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

APPENDIX A

PYTHON SCRIPT FOR DATA ACQUISITION

```
# =============================================================================
# Copyright (c) 2001-2021 FLIR Systems, Inc. All Rights Reserved.

# This software is the confidential and proprietary information of FLIR
# Integrated Imaging Solutions, Inc. ("Confidential Information"). You
# shall not disclose such Confidential Information and shall use it only in
# accordance with the terms of the license agreement you entered into
# with FLIR Integrated Imaging Solutions, Inc. (FLIR).
#
# FLIR MAKES NO REPRESENTATIONS OR WARRANTIES ABOUT THE SUITABILITY OF THE
# SOFTWARE, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE
# IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR
# PURPOSE, OR NON-INFRINGEMENT. FLIR SHALL NOT BE LIABLE FOR ANY DAMAGES
# SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING
# THIS SOFTWARE OR ITS DERIVATIVES.
# =============================================================================
#
# AcquisitionMultipleCamera.py shows how to capture images from
# multiple cameras simultaneously. It relies on information provided in the
# Enumeration, Acquisition, and NodeMapInfo examples.
#
# This example reads similarly to the Acquisition example,
# except that loops are used to allow for simultaneous acquisitions.

import os
import PySpin
import sys

NUM_IMAGES = 5000  # number of images to grab

def acquire_images(cam_list):
    """
    This function acquires and saves 10 images from each device.

    :param cam_list: List of cameras
    :type cam_list: CameraList
    :return: True if successful, False otherwise.
    :rtype: bool
    """

    print('*** IMAGE ACQUISITION ***\n')
    try:
        result = True

```

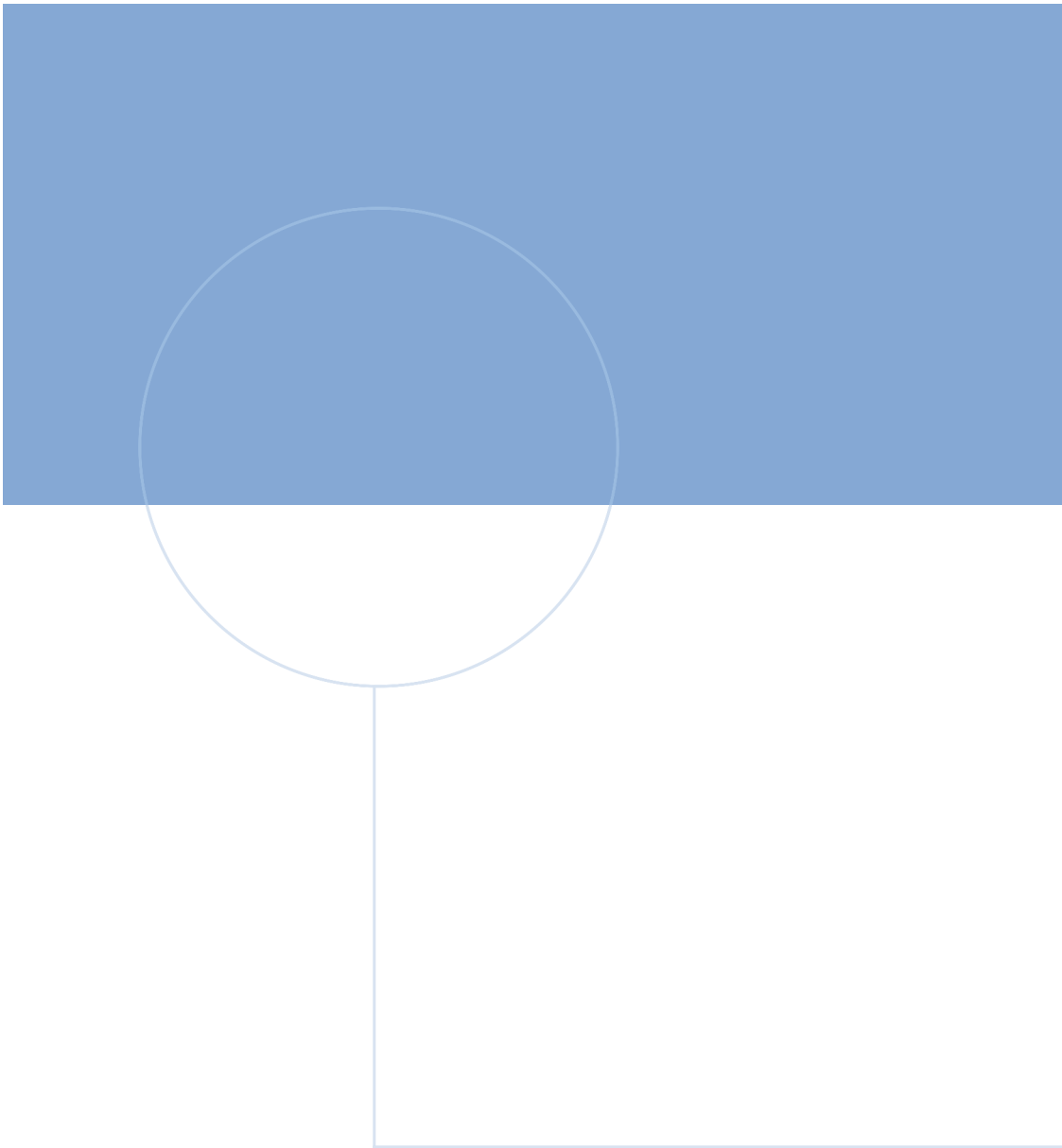
```
        # Prepare each camera to acquire images
        #
        # *** NOTES ***
        # For pseudo-simultaneous streaming, each camera is prepared as if it
        # were just one, but in a loop. Notice that cameras are selected with
        # an index. We demonstrate pseduo-simultaneous streaming because true
        # simultaneous streaming would require multiple process or threads,
        # which is too complex for an example.
        #

        for i, cam in enumerate(cam_list):

            # Set acquisition mode to continuous
            node_acquisition_mode = PySpin.CEnumerationPtr(cam.GetNodeMap().
    GetNode('AcquisitionMode'))
            if not PySpin.IsAvailable(node_acquisition_mode) or not PySpin.
    IsWritable(node_acquisition_mode):
                print('Unable to set acquisition mode to continuous (node
    retrieval; camera %d). Aborting... \n' % i)
                return False

            node_acquisition_mode_continuous = node_acquisition_mode.
    GetEntryByName('Continuous')
            if not PySpin.IsAvailable(node_acquisition_mode_continuous) or not
    PySpin.IsReadable(
                    node_acquisition_mode_continuous):
                print('Unable to set acquisition mode to continuous (entry \'
    continuous\' retrieval %d). \
                Aborting... \n' % i)
                return False

            acquisition_mode_continuous = node_acquisition_mode_continuous.
    GetValue()

            node_acquisition_mode.SetIntValue(acquisition_mode_continuous)

            print('Camera %d acquisition mode set to continuous...' % i)

            # Begin acquiring images
            cam.BeginAcquisition()

            print('Camera %d started acquiring images...' % i)

            print()

        # Retrieve, convert, and save images for each camera
        #
        # *** NOTES ***
        # In order to work with simultaneous camera streams, nested loops are
        # needed. It is important that the inner loop be the one iterating
        # through the cameras; otherwise, all images will be grabbed from a
        # single camera before grabbing any images from another.
        for n in range(NUM_IMAGES):
            for i, cam in enumerate(cam_list):
                try:
                    # Retrieve device serial number for filename
                    node_device_serial_number = PySpin.CStringPtr(cam.
    GetTLDeviceNodeMap().GetNode('DeviceSerialNumber'))

                    if PySpin.IsAvailable(node_device_serial_number) and PySpin
    .IsReadable(node_device_serial_number):
                        device_serial_number = node_device_serial_number.
    GetValue()
```

```
                        print('Camera %d serial number set to %s...' % (i,
    device_serial_number))

                    # Retrieve next received image and ensure image completion
                    image_result = cam.GetNextImage(1000)

                    if image_result.IsIncomplete():
                        print('Image incomplete with image status %d ... \n' %
    image_result.GetImageStatus())
                    else:
                        # Print image information
                        width = image_result.GetWidth()
                        height = image_result.GetHeight()
                        print('Camera %d grabbed image %d, width = %d, height =
     %d' % (i, n, width, height))

                        # Convert image to mono 8
                        image_converted = image_result.Convert(PySpin.
    PixelFormat_RGB8, PySpin.HQ_LINEAR) # Changed Mono8 -> RBG8

                        # Create a unique filename
                        if device_serial_number:
                            filename = 'AcquisitionMultipleCamera-%s-%d.png' %
    (device_serial_number, n)
                        else:
                            filename = 'AcquisitionMultipleCamera-%d-%d.png' %
    (i, n)

                        # Save image
                        image_converted.Save(filename)
                        print('Image saved at %s' % filename)

                    # Release image
                    image_result.Release()
                    print()

                except PySpin.SpinnakerException as ex:
                    print('Error: %s' % ex)
                    result = False

        # End acquisition for each camera
        #
        # *** NOTES ***
        # Notice that what is usually a one-step process is now two steps
        # because of the additional step of selecting the camera. It is worth
        # repeating that camera selection needs to be done once per loop.
        #
        # It is possible to interact with cameras through the camera list with
        # GetByIndex(); this is an alternative to retrieving cameras as
        # CameraPtr objects that can be quick and easy for small tasks.
        for cam in cam_list:

            # End acquisition
            cam.EndAcquisition()

    except PySpin.SpinnakerException as ex:
        print('Error: %s' % ex)
        result = False

    return result


def print_device_info(nodemap, cam_num):
```

```
    """
    This function prints the device information of the camera from the transport
    layer; please see NodeMapInfo example for more in-depth comments on printing
    device information from the nodemap.

    :param nodemap: Transport layer device nodemap.
    :param cam_num: Camera number.
    :type nodemap: INodeMap
    :type cam_num: int
    :returns: True if successful, False otherwise.
    :rtype: bool
    """

    print('Printing device information for camera %d... \n' % cam_num)

    try:
        result = True
        node_device_information = PySpin.CCategoryPtr(nodemap.GetNode('
    DeviceInformation'))

        if PySpin.IsAvailable(node_device_information) and PySpin.IsReadable(
    node_device_information):
            features = node_device_information.GetFeatures()
            for feature in features:
                node_feature = PySpin.CValuePtr(feature)
                print('%s: %s' % (node_feature.GetName(),
                                  node_feature.ToString() if PySpin.IsReadable(
    node_feature) else 'Node not readable'))

        else:
            print('Device control information not available.')
        print()

    except PySpin.SpinnakerException as ex:
        print('Error: %s' % ex)
        return False

    return result

def run_multiple_cameras(cam_list):
    """
    This function acts as the body of the example; please see NodeMapInfo
    example
    for more in-depth comments on setting up cameras.

    :param cam_list: List of cameras
    :type cam_list: CameraList
    :return: True if successful, False otherwise.
    :rtype: bool
    """
    try:
        result = True

        # Retrieve transport layer nodemaps and print device information for
        # each camera
        # *** NOTES ***
        # This example retrieves information from the transport layer nodemap
        # twice: once to print device information and once to grab the device
        # serial number. Rather than caching the nodem#ap, each nodemap is
        # retrieved both times as needed.
        print('*** DEVICE INFORMATION ***\n')
```

```
212
213         for i, cam in enumerate(cam_list):
214
215             # Retrieve TL device nodemap
216             nodemap_tldevice = cam.GetTLDeviceNodeMap()
217
218             # Print device information
219             result &= print_device_info(nodemap_tldevice, i)
220
221         # Initialize each camera
222         #
223         # *** NOTES ***
224         # You may notice that the steps in this function have more loops with
225         # less steps per loop; this contrasts the AcquireImages() function
226         # which has less loops but more steps per loop. This is done for
227         # demonstrative purposes as both work equally well.
228         #
229         # *** LATER ***
230         # Each camera needs to be deinitialized once all images have been
231         # acquired.
232         for i, cam in enumerate(cam_list):
233
234             # Initialize camera
235             cam.Init()
236
237         # Acquire images on all cameras
238         result &= acquire_images(cam_list)
239
240         # Deinitialize each camera
241         #
242         # *** NOTES ***
243         # Again, each camera must be deinitialized separately by first
244         # selecting the camera and then deinitializing it.
245         for cam in cam_list:
246
247             # Deinitialize camera
248             cam.DeInit()
249
250         # Release reference to camera
251         # NOTE: Unlike the C++ examples, we cannot rely on pointer objects
    being automatically
252         # cleaned up when going out of scope.
253         # The usage of del is preferred to assigning the variable to None.
254         del cam
255
256     except PySpin.SpinnakerException as ex:
257         print('Error: %s' % ex)
258         result = False
259
260     return result
261
262
263 def main():
264     """
265     Example entry point; please see Enumeration example for more in-depth
266     comments on preparing and cleaning up the system.
267
268     :return: True if successful, False otherwise.
269     :rtype: bool
270     """
271
272     # Since this application saves images in the current folder
273     # we must ensure that we have permission to write to this folder.
```

```
    # If we do not have permission, fail right away.
    try:
        test_file = open('test.txt', 'w+')
    except IOError:
        print('Unable to write to current directory. Please check permissions.')
        input('Press Enter to exit...')
        return False

    test_file.close()
    os.remove(test_file.name)

    result = True

    # Retrieve singleton reference to system object
    system = PySpin.System.GetInstance()

    # Get current library version
    version = system.GetLibraryVersion()
    print('Library version: %d.%d.%d.%d' % (version.major, version.minor, version.type, version.build))

    # Retrieve list of cameras from the system
    cam_list = system.GetCameras()

    num_cameras = cam_list.GetSize()

    print('Number of cameras detected: %d' % num_cameras)

    # Finish if there are no cameras
    if num_cameras == 0:

        # Clear camera list before releasing system
        cam_list.Clear()

        # Release system instance
        system.ReleaseInstance()

        print('Not enough cameras!')
        input('Done! Press Enter to exit...')
        return False

    # Run example on all cameras
    print('Running example for all cameras...')

    result = run_multiple_cameras(cam_list)

    print('Example complete... \n')

    # Clear camera list before releasing system
    cam_list.Clear()

    # Release system instance
    system.ReleaseInstance()

    input('Done! Press Enter to exit...')
    return result

if __name__ == '__main__':
    if main():
        sys.exit(0)
    else:
        sys.exit(1)
```



NTNU

Norwegian University of
Science and Technology