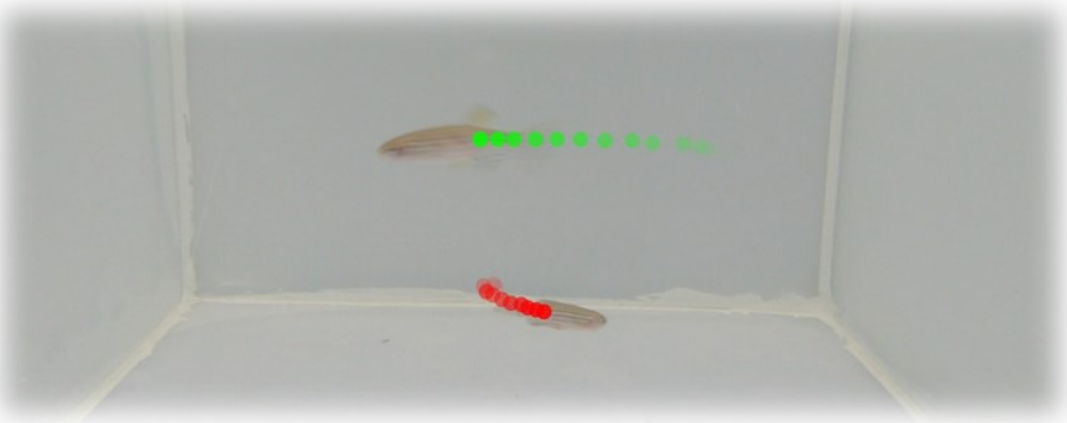
Regression Based Multi-View Zebrafish Tracking



MASTER THESIS

Mathias Gudiksen
Vision, Graphics and Interactive Systems
Aalborg University
The 3rd of June 2021

AALBORG UNIVERSITY
STUDENT REPORT

Aalborg University/Technical
Faculty for IT and Design
Vision, Graphics and Interactive Systems
Frederik Bajers Vej 7
9220 Aalborg Øst
http://www.es.aau.dk

Title:

Regression Based Multi-View Zebrafish Tracking

Project:

Master Thesis

Project Period:

1. February 2021 - 03. June 2021

Project Group:

Group 1046

Members:

Mathias Gudiksen

Supervisors:

Thomas B. Moeslund
Malte Pedersen

Number of Pages: 62 pages.
Appendix: 6 pages.
Ended 03-06-2021

Abstract:

Zebrafish (Danio rerio) has become increasingly important in medical trials due to the neurobiological proximity between humans and zebrafish. Studies, where zebrafish are exposed to various medications and drugs and their behavior, are analyzed. In such trials, a precise and accurate description of the fish movement in the tank is needed. Tracking of zebrafish has become an active topic of research and is supported by computer vision to accommodate the need for movement trajectories.

In this project, a system for automatic tracking for tracking multiple zebrafish is proposed, based on multi-object tracking state-of-the-art methods within pedestrian tracking. A framework exploiting the regression head of the object detector is used for generating 2D trajectories in two views, top and front, which are used to reconstruct and estimate a 3D position. A large amount of identity swaps is found to be harmful for the 3D triangulation module, return poor performance on the benchmark dataset. A method for detecting and minimizing identity swaps in a given sequence, based on sharing information between views in the sequence.
Results show great potential for a method to detect identity swaps, where the current method for correcting the detected swaps is not supported by the obtained results. More work should be invested in finding an optimal solution for modifying the tracklets around the ID swap detections.

Preface

This report is written during the final semester of the master programme in Vision, Graphics and Interactive Systems at Aalborg University. The report is written by Mathias Gudiksen. The topic for this project is mainly Computer Vision, where the focus is to use computer vision and image processing to detect and track shoals of zebrafish in 2D for later assemble 3D tracks based on the tracklets from two distinct views.

Citations throughout this report is referenced using the Vancouver reference style, which is a numerical citation method. Any material in this report, which is not cited, is composed by the author. Illustrations and simulations are carried out in MATLAB and results and real data is extracted and plotted using Python.

Mathias Gudiksen, VGIS10
Mgudik15@student.aau.dk
Aalborg University
3rd June, 2021

Acronyms

AP Average Precision

CDF Cumulative Distribution Function

DBT Detection Based Tracking
DFT Detection Free Tracking

FCN Fully Convolutional Network
FPN Feature Pyramid Network

IoU Intersection over Union

mAP mean Average Precision
MOT multi-object tracking

RPN Region Proposal Network

SGD Stochastic Gradient Descent
SOTA State-of-the-art

Contents

Chapter 1

Introduction

Nowadays, animals play an important role in biological and medical research, the most well known example of this is the usage of mice for testing. Animals are used for testing drugs and treatments before any trials on humans are started. Lately, scientists have diverged from mammals and started using aquatic animals [40]. In over 200 years, researchers and scientist have been using fish as biomedical models, starting with goldfish (Carassius auratus) for the study of aquatic toxicology, growth, stress, immunology, and reproduction [40]. In the 1970s, zebrafish (Danio rerio) were used for the first time as a biological model by George Streisinger, as zebrafish was simpler than a mouse and genetics were easier to manipulate [20]. Working with zebrafish has several advantages which make the research easier, and among these advantages are: Easy manipulation of its genome, high fecundity, short generation time, external fertilization [40].

The zebrafish embryo is transparent, which allows scientists to study the organ systems stages of development in the fish and after 48 hours, the embryo has formed a complete organ system. Zebrafish are excellent for scientific experiments, as zebrafish are similar to humans in several ways [40]. Zebrafish develop similar organs to what is found in human bodies (see Fig. 1.1), which makes it possible to evoke various diseases from the early stage of the fish's life. This makes it possible to test out medicine and observe the organs due to the translucent bodies. Besides the organ system, Santoriello and Zon claims that zebrafishes genetics are 70% similar to humans [35], which makes it possible to manipulate the genetics and create so-called *mutants*, by adding or removing specific genes [40]. Many researchers exploit the neurobiological proximity between humans and zebrafish, to measure responses to stimuli by analyzing the zebrafish behavior [26]. This approach is widely used for new drug and gene discovery in the biological and medical field, and these studies are often conducted by video recording a zebrafish in a laboratory. These recordings are analyzed and the fish's movement and behavior is used as a measure of the response of a given stimulus. Traditionally, these recordings are analyzed manually by the researcher, which is highly subjective and financially costly. In the next section, the importance of behavior analysis will be elaborated.

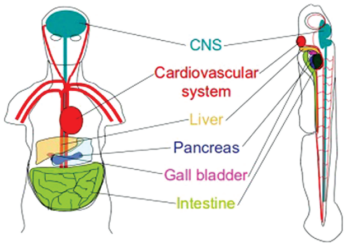
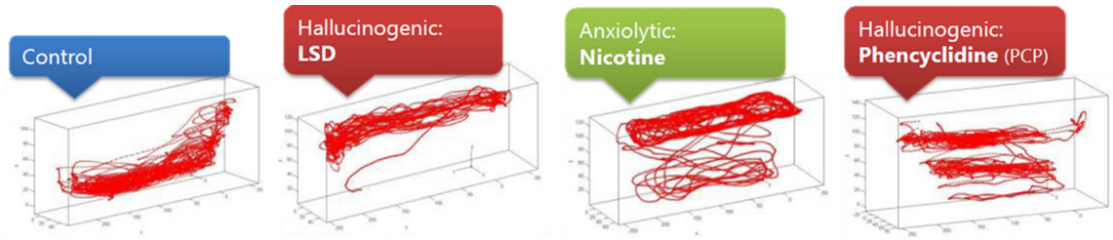


Figure 1.1: Similarities between organ system of humans and zebrafishes. Image from [40].

1.1 Behavior Analysis of Zebrafish

The demand for behavior analysis of zebrafishes is increasing, as stated by Teame et al. [40]. Precise and accurate descriptions of the fish's movement are needed to fully describe the fish's behavior. Stewart et al. [39] showed that the behavior of zebrafish change when exposed to a given stimuli. Doing so, allows the researcher to measure the impact of the stimulus. In Fig. 1.2 the movement of zebrafish is observed, such that responses can be analyzed.



(a) (b) (c) (d)

Figure 1.2: Tracked movement of a single zebrafish for behavioral analysis of various drugs. The red line is the trajectory of the fishs movement in the water tank and the axes are coordinates used as position. Different locomotor patterns are observed. Images adapted from [39].

From Fig. 1.2, it can be seen that different drugs result in special locomotor patterns. For instance, infecting the zebrafish with Lysergic Acid Diethylamide (LSD), the zebrafish tends to swim in the top of the tank (see Fig. 1.2b), while nicotine seems to provoke circular motion in the fish (see Fig. 1.2c) [39]. Zebrafish swim in the entire water tank, and the behavior is influenced by the applied drug in the tank. In these types of setups, it is crucial to allow the fish to swim in all possible directions and not limit the fish to a narrow space. Furthermore, it is hard for the observer to follow the fish's movement with more than a single fish present, which limits the efficiency of the experiment. If more fish could be added to the tank, the expense of running such trials would be reduced. Automating the trajectory labelling would minimize the subjective influence by the observer, reduce the cost of having a human following and noting every single movement of the fish. One possible solution to this problem would be to use computer vision to automatically track each fish in the aquarium, as computer vision does not physically interact with the fish and affect the zebrafish' behavior. Currently, the only commercial zebrafish tracking software on the market is Noldus EthoVision XT[1] that can track in 3D. This system is limited to a single zebrafish. EthoVision is used in numerous zebrafish behavior research projects and the starting price for EthoVision XT is currently 5 850 USD.

When aiming for a automatic tracking system for multiple fish, multi-object tracking might be the obvious choice, as tracking using cameras are considered passive compared to active tracking, for instance using a GPS device inserted in the fish. In this project, automatic tracking of multiple zebrafish in a controlled environment using computer vision will be examined.

In the next section, multi-object tracking will be described and common metrics to measure the performance of tracking algorithms will be presented.

[1] www.noldus.com/ethovision-xt

1.2 Multi-Object Tracking

Object tracking has become a very popular topic of research, especially tracking of multiple objects [4]. The most common topics for multi-object tracking (MOT) is person/pedestrian tracking, as these techniques can be used as important building blocks for plenty of situations, like gesture recognition, face or speaker identification and pose estimation [4]. Therefore, most recent papers within MOT, reports their findings and results on existing pedestrian MOT benchmarks[2]. In the case of comparison, a unified set of metrics is needed to directly compare the scores on the benchmarks. In this work, we will use the most common metrics used for benchmarks similar to MOT15, 16, 17, 20 [27]. When computing performance metrics, various situations are counted to evaluate the tracker, most of these situations are illustrated in Fig. 1.3.

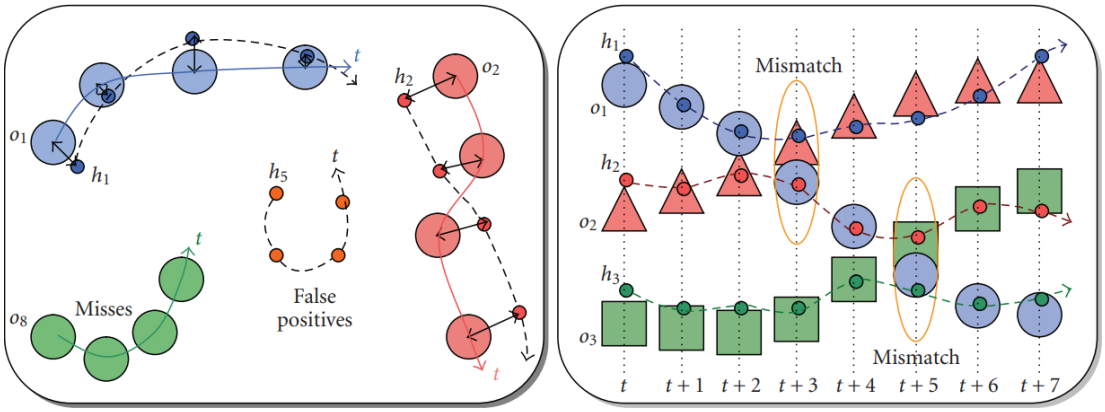


Figure 1.3: From left, big colored dots illustrate ground truth objects o, where small dots indicate object hypothesis h given by the object detector. Distance between h and o is inaccuracy of the object detector, which is used to calculate MOT Precision. Hypothesis without any associated ground truth object location is considered as false positives. In contrary ground truth objects without associated hypothesis is called a miss. To the right, we see three tracks intersecting, where the identity of the tracks (the color of the small dot) changes, this is counted as a mismatch. Image from [4].

Tracks are connected if the distance between the object hypotheses in the previous frame and current frame is below a certain threshold δ, this is often specified by the Intersection over Union (IoU) of the bounding boxes, however, other metrics could be used to represent the object, for instance, appearance information. This distance is often abbreviated as the cost, and a cost matrix is composed of size $|D| \times |T|$, where $|D|$ is the number of object hypotheses and $|T|$ is the number of active trackers. Most often, tracking is handled by solving the minimum cost assignment, where the total distance between object hypotheses between frames is minimized [4].

When an active tracker is unable to find and track an object for λ_{kill} frames, the tracker is deactivated. When a new object is detected outside any of the previous trackers' area, a new tracker is initialized. In the following section common performance metrics will be presented.

[2]https://motchallenge.net/

1.2.1 Performance Metrics

In this section metrics related to MOT will be presented and described, these metrics are used to rate and score tracking algorithms. Every metric is associated with an arrow, this arrow indicates if a higher or lower number is preferred for the metric.

Multiple Object Tracking Accuracy (MOTA) ↑ :
This metric measures the accuracy of the tracker, which accounts for the errors made during tracking. This accounts for misses, false positives and mismatches in all frames. This is an intuitive measure of the performance of the tracker to keep consistent trajectories and IDs, regardless of the precision of the detections. MOTA can be calculated as follows, [4].

$$\text{MOTA} = 1 - \frac{\sum_t (\text{m}_t + \text{fp}_t + \text{mme}_t)}{\sum_t \text{g}_t} \tag{1.1}$$

where, m_t is the number of misses, fp_t is the false positive detections, mme_t is the number of mismatches and g_t is the number of object present in frame t.

Multiple Object Tracking Precision (MOTP) ↓ :
This metric is the total error in the estimated position for matches between object hypothesis and the ground truth. This metric shows the trackers' ability to precisely locate the position of the object, regardless of its ability to maintain consistent ID as well as consistent trajectories. MOTP can be calculated as following, [4]:

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{1.2}$$

where, we sum the distance between all object-hypotheses and their ground truth d, and divide by the total number of matches c [4].guarantees an increase to the MTBF or leaves it unchanged A match is considered when a pair of object hypothesis h and ground truth object o are matched, as seen in Fig. 1.3.

Precision & Recall ↑ :
Precision and recall are two terms relating to the associated object detector. These metrics describes the object detector's ability to correctly find all ground truth objects (Recall) and how exact the detections are (Precision). These terms are illustrated in Fig. 1.4 and can be calculated by Eq. (1.3).

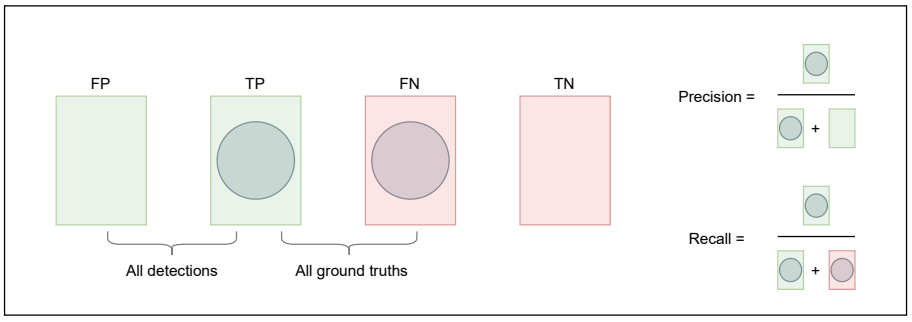


Figure 1.4: Illustration of how precision and recall are defined. Where a green box indicate a positive detection, red box indicate no detection and gray circle is the object of interest.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \qquad \text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{1.3}$$

Recall is considered as the ratio of correctly detected objects over the total number of ground truth objects. Whereas precision is related to the percentage of how many of the detected objects are correct. These metrics can be used to describe how well the object detector is performing.

Identification Precision & Recall ↑ :
Identification Recall is a measure, that computes the fraction of the ground truth objects that are correctly identified. Hence it uses True Positives to indicate a detected object with a correct ID and FN to indicate an object which is not detected.

Identification Precision is used to measure the fraction of the correctly identified detections among all the detections. Hence it uses FP to measure detected objects without the correct ID.

Identification F1-score (IDF1) ↑ :
This F1 metric is used to score the identification part of the tracking procedure. F1-score is a combination of the Precision and Recall for the detected IDs. F1-score can be calculated as following:

$$\text{IDF1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{1.4}$$

A high F1-score means that the tracker is good at maintaining IDs for the trajectories.

Mostly Tracked (MT) ↑ :
This metric is measuring the number of GT tracks, that are tracked more than 80% correct.

Mostly Lost (ML) ↓ :
This metric is measuring the number of GT tracks, that are tracked less than 20% correctly.

ID Swap ↓ :
ID Swap is when two objects swap their identities, and the metric measures how many times IDs are switched during the sequence. An illustration of the concept is seen in Fig. 1.5.

Fragmentation ↓ :
Fragmentation is how many times a ground truth trajectory is interrupted or untracked [27], this is illustrated in Fig. 1.5. Fragmentations are often caused by missing detections and a low number of fragmentations is desired.

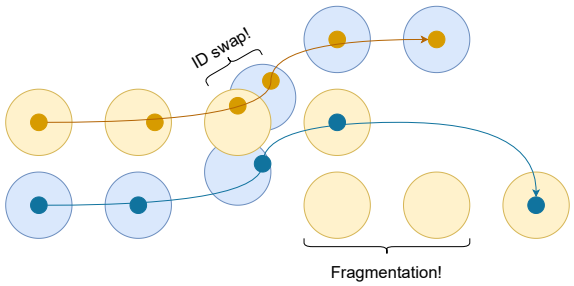


Figure 1.5: Illustration of the concept ID swap and fragmentation, each light color represent ground truth objects, and small dark color represent detection with ID.

Mean Time Between Failure (MTBF) ↑ :
Mean Time Between Failure is a metric which is dealing with the bias in previous metrics, where the length of the sequence is not considered. Sequences with two frames can score high on MOTA and IDF1, without considering the time aspect of the sequence. Carr and Collins [8], proposed a novel metric, which measures the average time a tracker can follow an object before making any mistakes. The great thing about MTBF is that the metric is monotonic, which means that reducing any potential tracking errors guarantees an increase to the MTBF or leaves it unchanged [8]. This property makes it incredibly easy to compare performance between trackers. MTBF is given by the standard formulation $MTBF_s$ and the monotonic variant $MTBF_m$. The standard formulation provides a quality ranking much similar to MOTA, where the monotonic version guarantees an improved score or no change when reducing the number of mistakes. However, the monotonic variant tends to penalize false positives too much [8].

Higher Order Tracking Accuracy (HOTA) ↑ :
HOTA is a novel MOT metric, which is developed to accommodate previous metrics' disadvantages [23]. Previous metrics tend to overemphasize either detection or association performance. Where a tracker can be great at object detection and localization but fail to identify the detected object and still score high on MOTA. HOTA evaluate the tracker's performance threefold, dividing the metric up into three Intersection over Union scores, evaluating localization, detection and association and rating them equally. Hence to score high on HOTA, the tracker needs to be good at all three steps. In this project, the HOTA metric is calculated using the TrackEval framework [18][3].

In summary, the three most relevant metrics are HOTA, MOTA and IDF1 score, which tell something about the total tracking, where MOTA is a measure of how frequent errors occur and IDF1 controls the ability to maintain the correct ID of the trackings. HOTA is a measure of the overall tracking, where localization, detection and association are measured and rated equally. Generally, HOTA is the easiest metric to make comparison between algorithms, where IDF1 is better at explaining the identification performance.

[3] `www.github.com/JonathonLuiten/TrackEval`

1.3 Related Work

In this section, previous work within the field of MOT, making the foundation for tracking multiple zebrafish will be examined. We will go through the currently used methods and techniques and present the corresponding issues relating to the methods according to the authors of the work.

1.3.1 Multi Object Tracking (MOT)

Tracking of multiple objects is different from tracking a single object [24]. MOT aims at analyzing video sequences, detecting, tracking and identifying a subset of objects without prior knowledge of the appearance of the objects. The output is typically rectangular bounding boxes enclosing the object, with an associated target ID to distinguish between intra-class objects [10]. In contrast to single object tracking, where appearance information is known a priori, the aim is to initialize the object of interest and attempt to follow the object throughout the video sequence for as long as possible. Object detection is a crucial step in MOT to provide information for the tracking algorithm to associate bounding boxes over time.

Detection Based Tracking (DBT) is one of the most common methods for initializing trackers [3, 6, 9, 24] where the tracking relies on an object detector, generating object hypotheses which are connected to form trajectories for the objects. On the contrary, Detection Free Tracking (DFT) is initialized without an object detector and requires manual initialization of the tracker [24, 41], which limits the model to tracking a fixed number of objects, often used in single object tracking. Using DFT often involves methods like Normalized Cross-Correlation Template Matching or Mean Shift to track objects.

Association problem is one of the biggest issues when employing DBT, connecting detections between frames into trajectories. A cost matrix is constructed by measuring the similarity between already tracked detections and new detections. Minimum cost optimization algorithms are used to find the association of detections that has the minimum cost such as Hungarian Algorithm [1, 3, 6, 9, 24, 30].

Spatio-temporal overlap between detected bounding boxes, also known as IoU, are often used as a measure of distance between bounding boxes [5, 6, 24], where the overlap of bounding boxes from the previous frame and current frame is calculated. IoU is a simple and effective method for computing a cost matrix when the object detector is almost perfect. However, in cases of missing detections from either False Negative detections or occlusions, IoU falls short.

Motion models are often combined with regular trackers, predicting the motion of the objects. Pedestrians often move linearly and predictably, hence simple linear motion models can be used [3, 7], or more complex algorithms such as Kalman Filter [1, 6, 32, 28]. The motion model can be used to predict future bounding boxes or limit the search space. In cases of unpredictable motion or long occlusions, motion models tend to fail.

Visual appearance modelling can be employed to describe the appearance of the tracked object, allowing the algorithm to re-identify lost tracks [3, 6, 21, 28, 32]. Visual appearance can be supplementary to IoU in the cost matrix. In cases where the object has unique features, i.e. pedestrians [3, 7], appearance modelling shows its potential, successfully linking broken tracklets into full trajectories.

1.3.2 Zebrafish Detection and Tracking

Using MOT to aid the behavioral analysis of zebrafish is an active topic of research. Recent studies have used laboratory setups where the aquarium was mounted with a top view camera [1, 28, 31, 32, 33]. Head tracking is a common method for robust detection of zebrafish, as the head is the only rigid part of the fish [1, 30, 32, 33]. The use of object detection for detecting zebrafish is a great choice, however extensive data is needed to perfectly model a zebrafish to ensure sufficient detections to keep consistent ID through partial occlusions. And object detectors are sensitive to total occlusions and changes in pose and appearance of the object.

Unsupervised tracking has emerged in the field of zebrafish tracking, using regular image processing techniques for detecting fish, such as background subtracting [28, 30, 31] and tracking the zebrafish using IoU. Unsupervised tracking seemed to compete well against object detection approaches, as no specific training is needed. However, during partial occlusions, unsupervised detection is prone to fail, as the extracted BLOBs are merged and cannot easily be split [28, 32]. Occlusion handling is a complex problem that often requires sophisticated and domain-specific solutions.

Stereo vision is a technique used to accommodate the occlusion problem, by expanding the dimension from 2D to 3D by introducing multiple views [9, 19, 25, 30]. Either by introducing a mirror [25], or adding an extra camera [9, 19, 30]. To provide even more information [19] introduced 2 side cameras, providing depth information to the tracking algorithm. The advantage and intuition of working in 3D is that most fish species do move in 3D space, such that 3D trajectories are necessary to fully describe their behavior. Furthermore, introducing multiple cameras can provide additional information during occlusion from another view, as two objects cannot be at the same location.

Motion models can be beneficial to approximate the location of an object during occlusions if the motion of the object is regular and predictable. Normally simple linear motion models are sufficient, but zebrafish's motions are not regular and are considered erratic. Kalman filters have been used with success by [1, 28, 32]. Kalman filters are complex filters used to estimate the state of an object. In most cases, the Kalman filter can approximate the fish's location, but when occlusions are longer than about 15 frames, the tracked object is often lost.

Re-identification can be used if a tracker has lost the tracked object. Often, re-identification happens by modelling the visual appearance of the tracked objects and compares new detections' similarity with the lost objects. The appearance of zebrafish is very similar, which makes re-identification a difficult task. Metric learning is often employed to define feature vectors used to distinguish between objects. Using Local Maximal Occurrence descriptor in conjunction with Hue, Saturation and Value color space [15] achieved impressive results of mean Average Precision (mAP) of 99% showing that metric learning can be used to discriminate between zebrafishes. However, this experiment was only tested in 2D with the assumption of a perfect object detector.

An extensive amount of research has been put into zebrafish tracking in various settings, but the methods used within the field have not seen a rapid change, the approach is more or less the same between papers. Pedestrian tracking has seen a rapid change and the focus in this project is to translate State-of-the-art (SOTA) within MOT to the field of zebrafish tracking. Current SOTA builds upon different neural network architectures,

for instance, Bergmann, Meinhardt, and Leal-Taixe, exploited the regression head of the Faster RCNN network [3], by assuming connected detections are not moving significantly between frames. By doing so, spawning the previous detection in the next frame and using the regression head to fit the bounding box to the object in the next frame. Doing so, allows the author to easily connect detections with short spatial distance, while maintaining the exact ID, by transferring the ID from the previous box.

This method is further extended by Brasó and Leal-Taixé, by using the same approach for tracking, while this paper introduces a method for learning an appropriate neural solver for the object tracking task, adding up MOTA with 2% on the benchmarks MOT16 and MOT17 [7]. In the following section, the benchmark dataset used in this project will be described.

1.4 3D-ZeF20 Dataset

Working in the field of MOT, it is important to have a consistent dataset. In this project it has been chosen to work with the 3D-ZeF20 dataset provided by Pedersen et al. [30][4]. This dataset is considered suitable for the application since it contains several occlusions which often occur in zebrafish tracking. Unlike several other datasets, this dataset is not recorded in shallow water, thereby not limiting the fish to swim only in a single plane, but allowing the fish to swim in 3D space. This increases the risk of occlusions of the fish, but more accurately describe their real behavior, allowing a more detailed analysis of the fish's movement and behavior.

1.4.1 Physical Setup

The great advantage of this dataset is that it is recorded from a top view, like most of the existing datasets around, but also recorded from the side view. This allows the tracking algorithm to gain more information about the individual fish and its trajectory. The physical setup used for video acquisition when recording data for the dataset can be seen in Fig. 1.6.

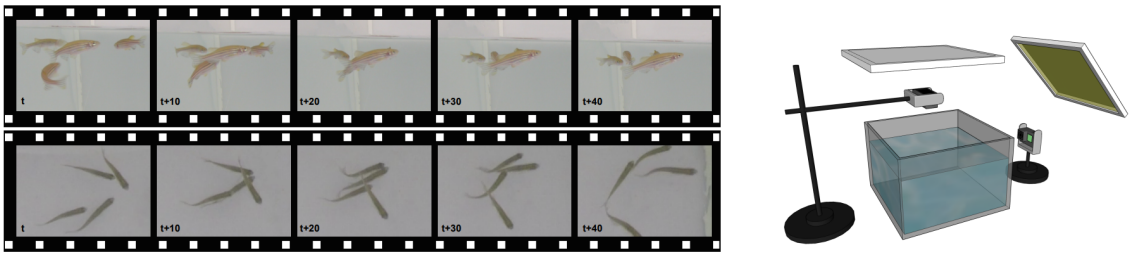


Figure 1.6: Illustration of the physical setup of the dataset, together with five images illustrating the potential of multiple views, by showing an occlusion from both views. Image is taken from the dataset [30].

Another great advantage of using this dataset is that it was recorded using off-the-shelf hardware (Two GoPro Cameras), wherein existing datasets, for instance, the data used in idTracker, is made up of highly specialized equipment. IdTracker uses a carefully designed

[4]https://motchallenge.net/data/3D-ZeF20

setup to improve the contrast between the background and the fish, to enhance the contour of the fish by back-lightning the fish with infrared light to improve the silhouette of the fish, scarifying texture of the fish and level of detail [31, 1].

The water tank in 3D-ZeF20 is of dimension $30 \times 30 \times 30$ cm, with a water depth of 15 cm, where idTracker uses an container of size $62 \times 45 \times 18$ cm, with a water level of 3 cm [31], moreover this dataset is not publicly available. In [15] a container of size $32 \times 32 \times 32$ cm, and water depth of 10.5 cm is used. In [32, 1, 33] following are used $30 \times 30 \times 3$ cm. As it can be seen, the most common method is filling the water tank with 3 cm of water, which constrains the movement to be approximated 2D, minimizing the risk of occlusion from the top view significantly. This can be a huge advantage in regards to the algorithm. However, constraining the space of the fish prevents natural behavior in the fish, thus the tracked behavior might not be representative of the actual behavior. As seen in Section 1.1, special behavior is often recognized as the fish stays either entirely in the top or bottom of the tank, these types of behavior might be prevented in low water depth tanks.

In total eight video sequences was recorded, with different number of zebrafish, an overview of the recordings can be seen in Table 1.1.

	ZeF-01	ZeF-02	ZeF-03	ZeF-04	ZeF-05	ZeF-06	ZeF-07	ZeF-08
Set	Train	Train	Valid	Valid	Test	Test	Test	Test
Fish	2	5	2	5	1	2	5	10
Fps	60	60	60	60	60	60	60	60
Length	120 s	15 s	30 s	15 s	15 s	15 s	15 s	15 s
Frames	14 376	1 800	3 600	1 820	1 800	1 800	1 800	1 800
Boxes	28 754	9 000	7 200	9 100	1 800	3 600	9 000	18 000
Complexity (Ψ)	0.26	0.5	0.03	0.63	0.00	0.01	0.16	0.28

Table 1.1: Overview of the dataset and the splits used in this project [30].

All the videos are recorded with a resolution of 2704×1520 pixels, 60 FPS, 1/60 s shutter speed, 400 ISO, and a linear field of view. The videos are recorded in RGB, as stated by the author of the dataset because the zebrafish can change their body pigmentation based on several factors, thus the color information might be valuable for tracking algorithms as well as behavior analyzing biologists [30]. The same training and validation splits as the benchmark are used to ensure comparison [30].

1.4.2 Complexity

Intuitively, the number of fish in the tank makes tracking more difficult. However, as stated by Pedersen et al., while this is true to some extent, what contributes the most to the complexity is the amount of occlusion in the data. Complexity for each sequence is calculated using several factors all related to occlusions. It should be noted that in [30], an occlusion is defined as follows: *"if two fish are part of an occlusion it counts as two events"* The following metrics are used to determine the complexity of the respective dataset:

Occlusion Count (OC): The average number of occlusion events pr second.
Occlusion Length (OL): The average time in seconds of all occlusion events.
Time Between Occlusions (TBO): The average time in seconds between occlusions.
Intersection Between Occlusions (IBO): A measure of how large a part of the fish is part of an occlusion event.

Above descriptions are adopted from Pedersen et al. [30]. IBO is used to compensate for the greedy occlusion count when the fish just exactly touch each other. This metric measures the overlap and indicates the extent of the occlusion. IBO is measured as the intersection between bounding boxes in the occlusion divided by the area of the bounding box, a more in-depth explanation of the concept can be found in the paper [30]. A single complexity score (Ψ) is calculated for every sequence in the dataset, and the equation used can be seen in Eq. (1.5), [30].

$$\Psi = \frac{1}{n} \sum_{v}^{\{T,F\}} \frac{\mathrm{OC}_v \mathrm{OL}_v \mathrm{IBO}_v}{\mathrm{TBO}_v} \tag{1.5}$$

Where, n is the number of camera views, where $v = T, F$ refers to Top and Front view respectively. A low complexity score indicates that the sequence is less complex and vice versa. Detailed information about the complexity measures of each sequence can be found in the original work, and in Table 1.1 the total complexity score for each sequence is listed. To provide visual understanding of the dataset, a few frames is shown with annotations in Fig. 1.7.

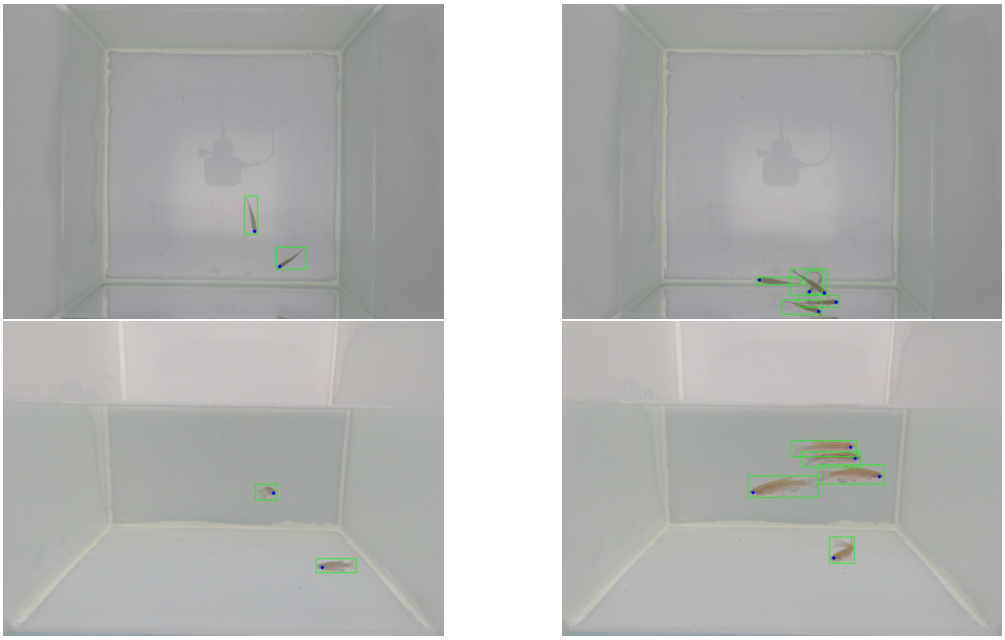


Figure 1.7: Examples from the dataset, with annotations. The blue dot represent the annotated fish head and the green box represent the annotated bounding box. The two images in the top are from top view and the two images in the bottom, is the same time step, from the front view.

In Fig. 1.7, the benefit of having multiple viewpoints can be seen, as in the case with the five fish, it clutters in top view but in front view, it is possible to see the fish more clearly. Furthermore, we have much more textural information about the individual fish in the front view. The front view is not always the best, as occlusions more often occur in that viewpoint, in these cases the top view can often be used to keep track of the occluded fish. Working with two cameras opens up the opportunity to work in 3D space. It has been chosen to track in 2D and reconstruct the 2D tracks into 3D, instead of reconstructing the detections and do tracking in 3D.

1.5 Problem Statement

In this section, the main issues when tracking zebrafish will be highlighted, and several questions are formed to set the focus for the later work. This focus is based on considerations made in the previous analysis. The following questions will be answered:

- How can multiple zebrafish be tracked in two distinct camera views?
- How can SOTA in MOT be adapted for tracking of zebrafish?
- How can information between views be shared to improve tracking performance during occlusions?
- How can 3D trajectories be constructed from two views?
- Is it possible to beat the current best performance on the 3D-ZeF20 benchmark?

Chapter 2

Tracktor

As mentioned in Section 1.3, Bergmann, Meinhardt, and Leal-Taixe, has developed a *detection based tracking* framework, which exploits the associated object detector to keep track of detections and corresponding identities [3]. They show that it is possible to obtain state-of-the-art tracking performance, by only training a neural network for object detection. This is a major advantage since object detection datasets are far more common than datasets containing annotated trajectories. The work present two major contributions, as they introduce Tracktor, which is a framework for tackling MOT by exploiting an object detectors regression head for temporal realignment of bounding boxes [3]. Furthermore, they propose two simple extensions to improve the aforementioned framework, by introducing re-identification Siamese Network and a simple linear motion model [3]. As stated by the authors, the framework does not need any tracking specific training and no complex optimization is needed at test time, hence the tracker is *online*. In this project, the framework developed by Bergmann, Meinhardt, and Leal-Taixe [3] will be employed for automatic tracking multiple zebrafish. The core element in Tracktor, is a regression-based detector, in this work they employed Faster R-CNN [34], with ResNet-101 [16] with a Feature Pyramid Network (FPN) [22] for feature extraction. The next sections will go more into detail with these algorithms. In Fig. 2.1, a detailed overview of the Tracktor framework can be seen and in Section 2.2.1 the framework and its implementation will be elaborated in more detail.

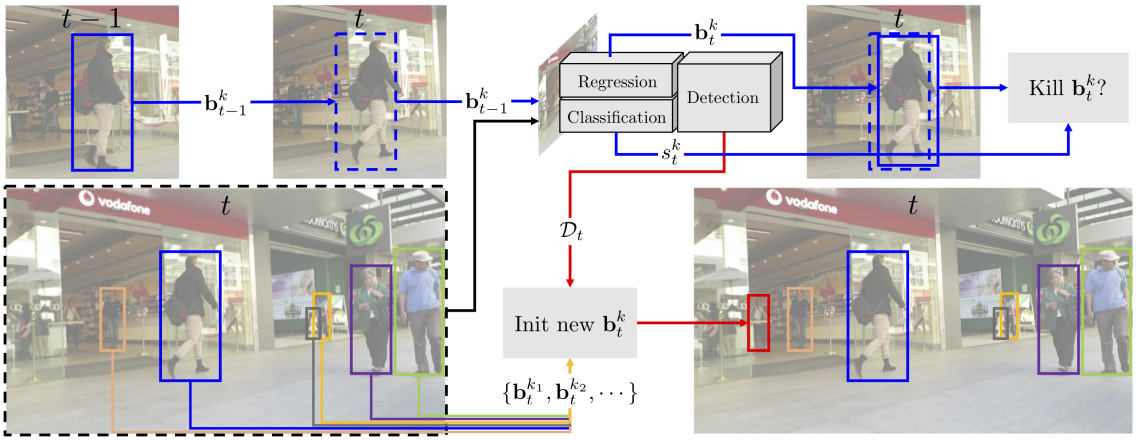


Figure 2.1: Overview of the Tracktor Framework for MOT [3]. Where b^k_t is the bounding box for object k at time t.

2.1 Object Detection with Faster R-CNN

Ren et al. proposed an effective solution to perform region proposals, where proposal computation is almost cost-free [34]. While developing Faster R-CNN, the authors have adopted its predecessor, Fast R-CNN for detection [13]. Therefore, Faster R-CNN has improved the speed of Fast R-CNN by handling the region proposal differently. Fast R-CNN was using an algorithm for finding potential object locations named Selective Search [13, 34]. This algorithm is an order of magnitude slower than the detection network, hence the bottleneck for achieving near real-time speed is found at the region proposal network. Ren et al., proposed a Fully Convolutional Network (FCN) for generating region proposals called Region Proposal Network (RPN). This network shares weight with the detector network, such that convolutions are shared at test-time, reduces the cost of generation region proposals [34]. Doing so allows the unified network to obtain a frame rate of 5 fps, including all the steps in the pipeline. In Fig. 2.2 a brief overview of Faster R-CNN can be seen.

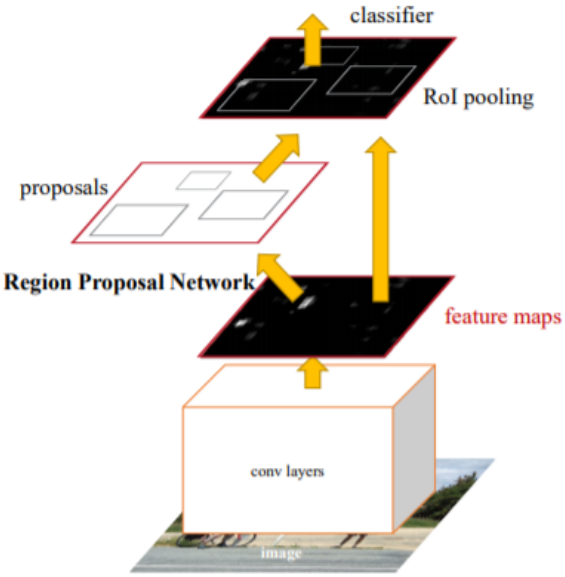


Figure 2.2: Overview of the end-to-end unified network Faster R-CNN, which outputs boxes and class scores for every object in the image. Image adopted from [34].

In Fig. 2.2, it can be seen that Faster R-CNN consists of convolutional layers, a RPN, which takes the feature map from the convolutional layers as input and outputs region proposals, which are fed to the classifier to determine the class label. The major advance in Faster R-CNN is that convolutions are shared between the classifier and the proposal network. In the next section, the RPN will be elaborated in detail, followed by a brief explanation of how the network is trained.

2.1.1 Region Proposal Network

As seen from Fig. 2.2, the RPN is added on top of a deep convolutional network, which extracts features from an image using convolutional kernels. The RPN takes the feature map from the backbone network and outputs a set of rectangle object proposals together with an objectness score for each proposal. These proposals are generated by sliding a

small network over the convolutional feature map from the last convolutional layer, often using a 3×3 respective field. Every sliding window is mapped to a lower-dimensional feature and fed into two fully connected layers. The first network is responsible for fitting a bounding box to the object, called *box-regression layer* and the other layer is responsible for predicting a class label for the box, called *box-classification layer* [34]. This process is illustrated in Fig. 2.3.

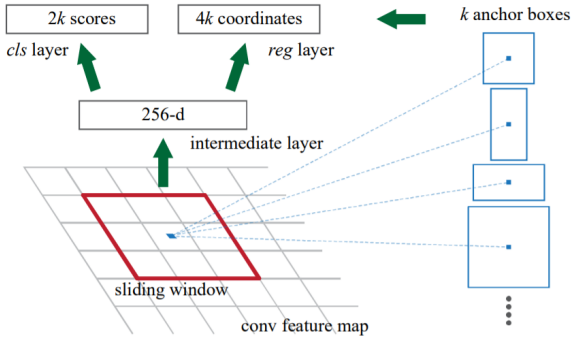


Figure 2.3: Process of feeding sliding window to box-regression and box-classification layer. Image adopted from [34].

Where in each sliding window location, a set of anchors are defined. In Fig. 2.3, k anchors are illustrated. Anchors can be viewed as predefined object proposals, such that the regression box layer does not need to learn to predict the entire bounding box, but instead regress the object's location relative to the k reference anchor boxes. The box-regression layer outputs $4k$ coordinates to determine the dimension of the bounding box (x, y, w, h) and the box-classification layer outputs $2k$ scores, which estimates the probability of an object is present or not [34]. Anchors are centred in the window and often different aspect ratios and scales are used for each anchor. In the work by Ren et al. [34], they use 3 scales and 3 aspect ratios, giving $k = 9$ anchors at each window position in the feature map. For a feature map of size $W = H = 50$ we generate in total $(W \times H \times k) = 22\,500$ anchors.

Training an RPN is carried out by assigning a binary class label (object or not) to every anchor. A positive label is given for anchors with highest IoU overlap with the annotated bounding box and anchors with an IoU overlap above 0.7 with any ground truth bounding box. Following the scheme for labelling below:

$$\text{label} = \begin{cases} \text{Positive}, & \text{IoU} > 0.7 \\ \text{Don't Care}, & 0.3 \leq \text{IoU} \geq 0.7 \\ \text{Negative}, & \text{IoU} < 0.3 \end{cases}$$

In cases where IoU is between 0.3 and 0.7, we exclude the anchors from training, since the anchor is in-between positive and negative and not strictly negative. With the assigned labels, the objective is to minimize the multi-task loss function as used in Fast R-CNN, see Eq. (2.1) [34].

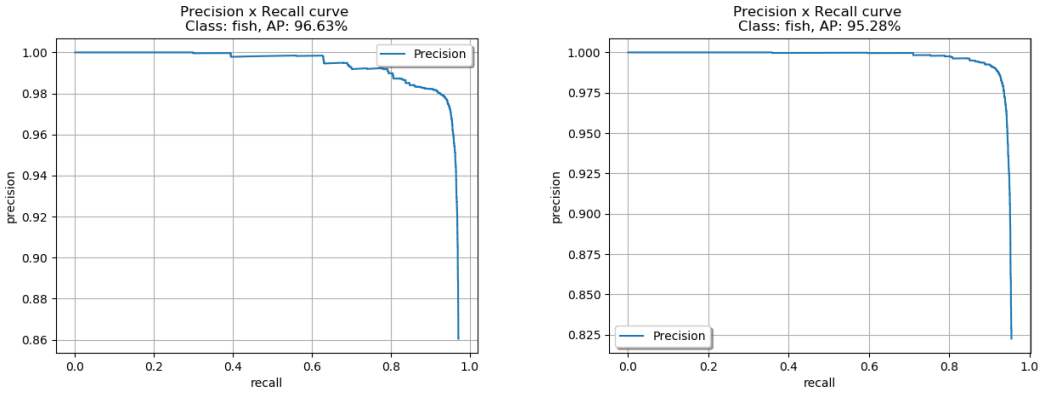
$$L\left(\{p_i\}, \{t_i\}\right) = \frac{1}{N_{cls}} \sum_i L_{cls}\left(p_i, p_i^*\right) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}\left(t_i, t_i^*\right) \tag{2.1}$$

Where, i is the index of the anchor, L is the total loss, L_{cls} is the class loss (entropy loss), L_{reg} is the regression loss (smooth L1 loss). p_i^* is 1 when anchor i is positive and

0 when negative, t_i is the predicted bounding box coordinates (x, y, w, h) and t_i^* is the ground truth bounding box. N_{cls} and N_{reg} are normalization factors, where cls is the size of the mini-batch and reg are the number of anchor locations. λ is used to account for the lower influence of the regression loss on the total loss. Scaling the regression loss to the same magnitude of N_{cls}, usually $\lambda = 10$. p_i^* is the ground truth label and is equal to 1 when a positive anchor is evaluated, meaning that regression loss will only contribute to the total loss for when the anchor is indeed positive. L_{reg} is only activated when $p_i^* = 1$ and disabled otherwise ($p_i^* = 0$), by the term $p_i^* L_{reg}$, to only include positive anchors misalignment into the loss function.

As stated by the authors, the RPN can easily benefit from deeper and more expensive features [34]. In many cases, Faster R-CNN is used with Deep Residual Networks (ResNet) for feature extracting and classification.

Now that the theory behind Faster R-CNN employed in Tracktor is explained in detail, the network can be trained to fit the model to the application. Faster R-CNN is trained on the 3D-ZeF20 dataset, using ZeF-01 and ZeF-02 as training sequences and ZeF-03 and ZeF-04 for validation and performance scoring. The model is trained using Stochastic Gradient Descent (SGD) with a learning rate of $1e^{-5}$, momentum of 0.9 and weight decay of $5e^{-4}$. During training horizontal flipping are used as data augmentation and the network is trained for 20 epochs when the validation loss settled. A model has been trained for each view and the precision/recall curves can be seen in Fig. 2.4.



(a) Top View (b) Front View

Figure 2.4: Precision / Recall curves for the Faster RCNN Object detector used, plots generated using the method provided by [29].

As it can be seen from Fig. 2.4, the Average Precision (AP) for top and front view are 96.63% and 95.28% respectively, which indicate that the model is well trained and able to correctly detect and localize zebrafish in both views.

Tracktor comes with a re-identification step, which is used to rematch lost tracks, this method will be examined in the following section.

2.2 Re-identification using TriNet

In the work by Bergmann, Meinhardt, and Leal-Taixe, they have utilized a short term re-identification model, for matching deactivated tracklets with newly detected objects

and attempt to re-identify the lost tracklets. For this purpose, a Siamese CNN called TriNet is employed. TriNet is based on ResNet50 and trained using Triplet loss [17]. In this section TriNet will be explained and how it is trained using Triplet Loss.

TriNet

TriNet is based on ResNet50, where the last layer is discarded. Two dense layers are added, where the first layer contains 1024 neurons and the last layer contains 128 neurons. The last layer outputs the embedded version of the input, hence the size of the embedding is 128. For re-identifying tracks, the tracks are compared using the euclidean distance in the embedded space. The lower the distance, the more similar the inputs should be. To train the network to output embeddings that are close for semantic similar images, and far apart for semantic different images, a special type of loss function is employed.

Triplet Loss

In general, Triplet loss is about sampling three images, one anchor image a, one image of the same object p, abbreviated *positive sample* and an image of a different object n *negative sample*, such that we can minimize the following equation Eq. (2.2) [17].

$$\mathcal{L}_{\text{tri}}(\theta) = \sum_{\substack{a,p,n \\ y_a = y_p \neq y_n}} [m + D_{a,p} - D_{a,n}]_+ \tag{2.2}$$

where $\mathcal{L}_{\text{tri}}$ is the triplet loss, y_a and y_p is the class label / ID of the given object, and y_n is the class label for the negative object. $D_{a,p}$ is the distance between anchor the positive sample, $D_{a,n}$ is the distance between anchor the negative sample. m is a minimum margin used for ensuring that positives are closer to the anchor than negatives samples by a margin and to avoid the neural network to output zero for all embeddings.

The expression in Eq. (2.2) can be reformulated as following, which makes the minimization process more clear.

$$\mathcal{L}_{\text{tri}}(\theta) = \sum_{i}^{N} \left[\|f_\theta(x_i^a) - f_\theta(x_i^p)\|_2 - \|f_\theta(x_i^a) - f_\theta(x_i^n)\|_2 \right] \tag{2.3}$$

where, f_θ is the transformation function for generating embeddings, which in this case is non-linear because of the neural network employed, $f_\theta(x_i^a)$ is the embedded version of the anchor in sample i and $\|\|_2$ is the euclidean distance between two vectors. From above, it can be seen to minimize the expression, we simply have to reduce the distance between sample a and p, while we keep the distance between a and n large. Hence the distance between images of the same track would be tightly clustered in the embedded space while maintaining a relatively large distance ($\geq m$) to the remaining classes [17]. Optimizing the loss over the entire dataset would force the model to produce similar embeddings and the network can be trained end-to-end. In Fig. 2.5, an overview of the triplet loss can be seen.

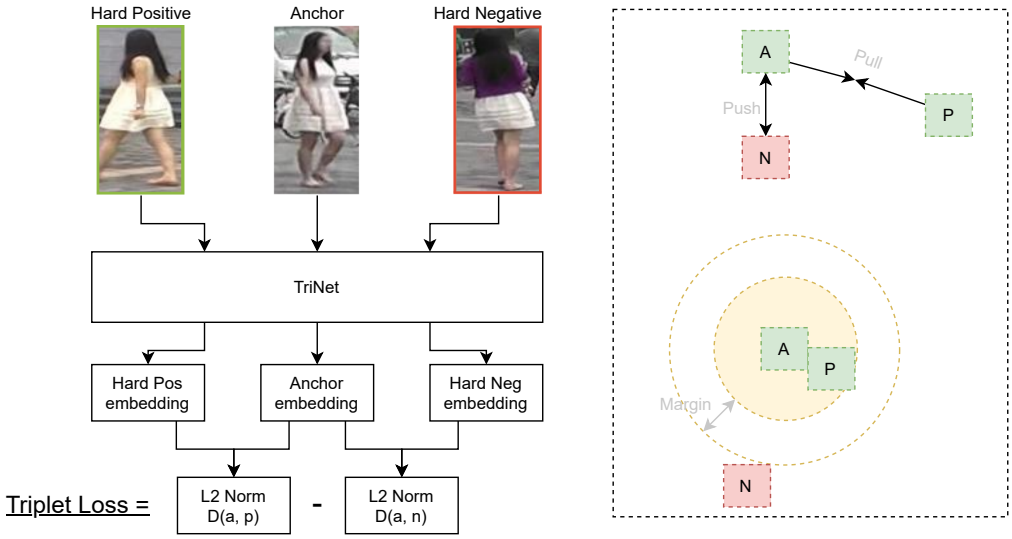


Figure 2.5: Illustration of how triplet loss are defined and the overall goal accomplished by training a network using triplet loss. Images from the Market-1501 dataset[1].

From left in Fig. 2.5, it can be seen that triplet loss consists of two distance metrics $D(a, p)$ and $D(a, n)$ which is the distance from the anchor to the positive and negative sample respectively. On the right side, the samples in the embedded space can be seen, where positive samples are pulled towards the anchor and negative samples are pushed away, such that the positive and negative sample are separated by a margin m. The major problem in triplet loss is how to choose the triplets. As stated by Hermans, Beyer, and Leibe [17], the mapping function f_θ can quickly learn to map trivial triplets, hence a large fraction of the dataset becomes uninformative. Therefore, they propose a method for mining samples for Triplet loss, which ensures that hard samples are used in the training. However, only training on too hard samples, would force the network to learn from outliers, and prevent the network from learning normal embeddings. They proposed a modification to the classical approach when using triplet loss, which they call Batch Hard sample mining.

Batch Hard Triplet Mining

The core idea behind Batch Hard sample mining is to form batches by randomly selecting P identities, and from these identities, we randomly sample K images from each. This gives a mini-batch of $P \times K$ images. From these images, we choose the *hardest positive* and the *hardest negative* samples from the mini-batch. Hence the batch hard loss function becomes the expression in Eq. (2.4) [17].

$$\mathcal{L}_{\mathrm{BH}}(\theta; X) = \overbrace{\sum_{i=1}^{P} \sum_{a=1}^{K}}^{\text{all anchors}} [m + \overbrace{\max_{p=1\ldots K} D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right)}^{\text{hardest positive}} - \overbrace{\min_{\substack{j=1\ldots P \\ n=1\ldots K \\ j \neq i}} D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^j\right)\right)}^{\text{hardest negative}}] \tag{2.4}$$

Where D is the distance metric for calculating the distance between two embeddings, where euclidean is often used, X is the mini batching currently sampled. Most of the

[1] www.kaggle.com/pengcw1/market-1501

time, the selected triplets can be considered as *moderate triplets*, since it is not the absolute hardest sample, but only the hardest sample among the images in the batch. According to the authors, moderate triplets are the best for learning with triplet loss. In the work by Hermans, Beyer, and Leibe [17] and Bergmann, Meinhardt, and Leal-Taixe [3], they have been using $P = 18$ persons and $K = 4$ images when training. Otherwise, the TriNet is optimized using Adam with $\beta = (0.9, 0.999)$ with a decaying learning rate. During training data augmentation techniques, such as horizontal flipping and randomly cropping are employed. The TriNet architecture is designed for person re-identification, hence the input size is $H \times W = 256 \times 128$, therefore the ground truth bounding box annotations are cropped and resized to fit the input dimension of the network [3].

2.2.1 Implementation

Now that both the object detection network Faster R-CNN and the re-identification network TriNet has been explained and trained. We can proceed by explaining the implementation, which assembles the framework Tracktor. In Fig. 2.6 a simplified illustration is conducted to provide an overview of how the individual modules are interacting.

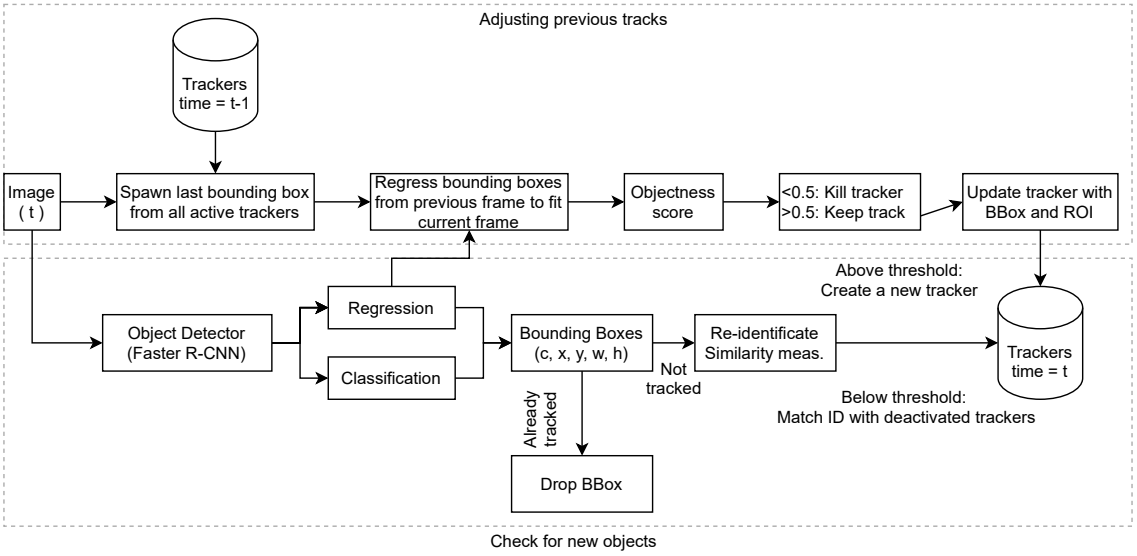


Figure 2.6: Diagram explaining the implementation of Tracktor, divided into two steps. The first step is to adjust the previous track, by exploiting the regression head of Faster R-CNN to refit the bounding boxes from image (t-1) and updating the tracker with new boxes. The second step is to initialize new tracks, by checking the Faster R-CNN for object detections not already tracked, and if detected check if the newly detected object is similar to an already deactivated track.

2.3 Results

After training the object detector and re-identification network, the validation set can be evaluated using the trained models. The tracktor framework comes with a simple motion model under the assumption of constant velocity, which as the name imply, assume a constant motion of the tracked object. From Section 1.3 we stated that motion models in zebrafish tracking weren't very successfully, due to the erratic motion of zebrafish. The

motion model shifts the bounding box by the average vectorial difference, based on the five previous frames. To investigate whether this motion model has a positive effect on the tracking performance, three distinct experiments have been conducted. A baseline model, which uses the regression head only, without re-identification and motion. A model using regression head and re-identification step and lastly all three methods. The results can be found in Table 2.1 for top view, and in Table 2.2 for front view.

	HOTA	MOTA	MOTP	IDF1	IDR	IDP	Prc	Rcll	FP	FN	MT	ML	ID_{swap}	Frag.	MTBFs	MTBFm
ZeF-03	52.7%	95.0%	0.133	57.3%	57.2%	57.4%	0.98	0.976	72	86	2	0	22	14	135.269	70.34
ZeF-03+	**65.9%**	**95.4%**	**0.133**	**75.5%**	**75.4%**	**75.7%**	**0.98**	**0.976**	**72**	**86**	2	0	**8**	**14**	**195.389**	**103.441**
ZeF-03++	42.7%	95.0%	0.137	34.4%	34.4%	34.5%	0.978	0.975	78	90	2	0	11	19	159.7	83.6
ZeF-04	54.3%	87.6%	0.179	58.5%	58.1%	59.0%	0.948	0.933	235	303	5	0	27	41	82.2	43.2
ZeF-04+	**64.2%**	**88.0%**	**0.179**	**74.0%**	**73.4%**	**74.5%**	**0.948**	**0.933**	**235**	**303**	5	0	**7**	**41**	**91.0**	**48.0**
ZeF-04++	59.1%	86.3%	0.180	66.3%	65.8%	66.7%	0.939	0.926	276	335	5	0	13	48	76.9	40.3

Table 2.1: Tracking results on validation data from top view.
+ = with re-identification step and ++ = with re-ID and Motion modeling.

	HOTA	MOTA	MOTP	IDF1	IDR	IDP	Prc	Rcll	FP	FN	MT	ML	ID_{swap}	Frag.	MTBFs	MTBFm
ZeF-03	40.5%	95.6%	0.116	33.9%	34.0%	33.9%	0.979	0.98	77	72	2	0	11	9	**252.0**	**135.7**
ZeF-03+	**48.1%**	95.6%	**0.116**	43.8%	43.9%	43.8%	0.979	0.98	77	72	2	0	**8**	**9**	**252.0**	**135.7**
ZeF-03++	**48.1%**	**95.9%**	0.117	**44.0%**	**44.0%**	**43.9%**	0.979	**0.982**	**75**	**65**	2	0	9	12	196.4	104.0
ZeF-04	39.7%	85.8%	0.150	37.9%	37.4%	38.5%	0.946	0.919	238	369	5	0	41	50	69.8	36.7
ZeF-04+	40.5%	**85.9%**	0.150	38.4%	37.8%	39.0%	**0.946**	**0.919**	**238**	**369**	5	0	35	**50**	**71.0**	**37.4**
ZeF-04++	**41.2%**	85.5%	**0.149**	**41.7%**	**41.0%**	**42.3%**	0.945	0.916	242	383	5	0	**34**	54	69.6	36.6

Table 2.2: Tracking results on validation data from front view.
+ = with re-identification step and ++ = with re-ID and Motion modeling.

From Tables 2.1 and 2.2 it can be seen that the Siamese re-identification model based on TriNet greatly improves the IDF1 score in all cases, as well as HOTA. The greatest improvement is in the top view, which might be reasonable. In the front view, the shape and look of the fish can change drastically when the fish change direction, i.e. swimming towards the camera versus when swimming perpendicular to the camera, which makes it hard to re-identify. However, in the top view, the size and shape does not change significantly, which makes the top view more suitable for re-identification.

Another interesting finding is that in the top view the motion model does not improve the performance at all, in fact, it does decrease the performance. It might be due to the constant velocity assumption since the motion is far from linear and constant in the top view. Unlike top view, the front view is affected just a small portion, increasing the identification metric (IDF1, IDR and IDP), this might be due to the motion can more or less be approximated linear, when swimming perpendicular to the camera, and slowly changing direction. However, motion does not contribute significantly and in many cases, it decreases the detection performance. In ZeF-03 the motion model improved MOTA by 0.3% wherein ZeF-04, MOTA was decreased by 0.3% below the baseline. From this perspective, it cannot be justified to keep the motion model, but it raises the potential to investigate the motion further in terms of a different approach. The re-identification model works impressively and should be a must-have in the framework, the re-identification model adds more computational load to the Tracktor framework, but in this application real-time online tracking is not the scope.

2.3.1 Discussion

From these results it could be concluded that the re-identification network successfully increasing the performance, wherein, the motion model did not work as expected. However, there is room for adjustments since these models come with a wide range of hyperparameters that could be tuned for the specific application. Tracktor is officially build for pedestrian tracking and therefore the default parameters are thereof. For instance, to even get the framework to run we had to change different parameters, such as the B and P values in Triplet Loss for the re-identification model, due to the sparse number of different IDs in the dataset versus MOTchallenge. The motion model is currently implemented by averaging the difference between the last five bounding box position and shape and hence forecasting where the new bounding box might be. The amount of frames used is a hyperparameter that could be tuned. Increasing the number of frames used in the average makes the motion more smooth as more information is used for generating the direction, but in contrast, the motion model becomes less adaptive to sudden changes. As the movement of zebrafish is considered erratic, a lower number of frames might improve the motion model impact as the model becomes more adaptive.
Furthermore, the re-identification model contains lots of parameters and are entirely build for pedestrian identification. For instance, TriNet is designed to take images of size 128×256 as input, which is a typical size and aspect ratio used for pedestrians, and ground truth, as well as detected object, are cropped and resized to fit this dimension. However, this aspect ratio might not be the most appropriate ratio for ROIs with zebrafish in. In Chapter 4, the re-identification model will be further examined, to specialize the model to the field of zebrafish instead of pedestrians.

To understand where and why errors are made, the tracklets are plotted with the assigned ID, such that a relation can be found, in Fig. 2.7 the evaluation of the validation set is seen.

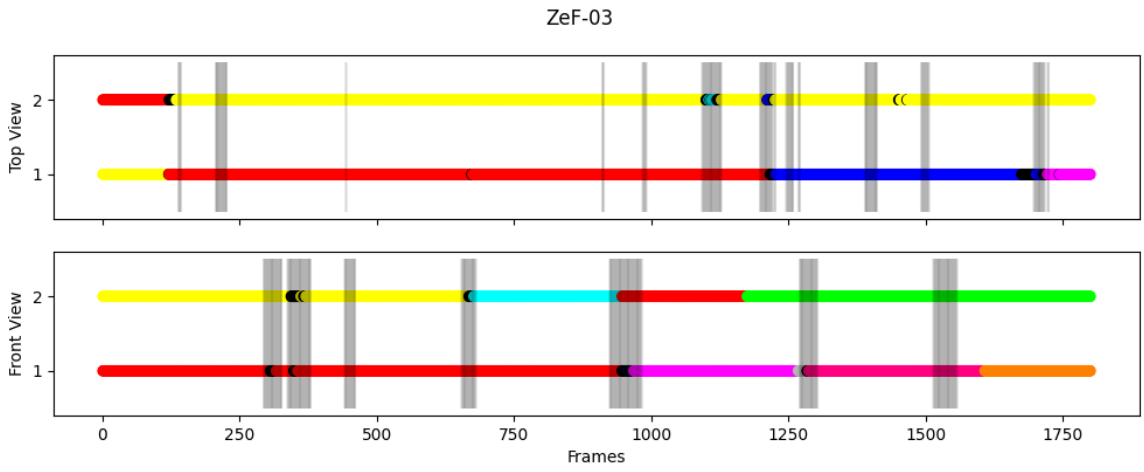


Figure 2.7: Evaluation of ZeF-03 using Tracktor with reID. Vertical gray lines indicate an occlusion had happened in that particular frame. Each color represents a unique ID, and a change from one color to another indicates an ID swap. Black means missing detection. Integers on the vertical axis represent ground truth annotated ID.

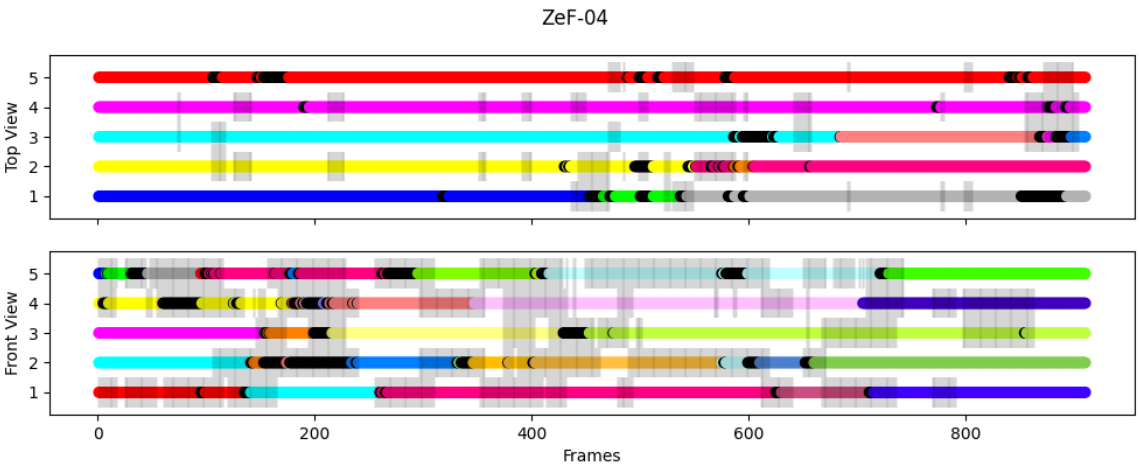


Figure 2.8: Evaluation of ZeF-03 using Tracktor with reID. Vertical gray lines indicate an occlusion had happened in that particular frame. Each color represent a unique ID, and a change from one color to another indicates ID switch. Black means missing detection. Integers on the vertical axis represent ground truth annotated ID.

First of all, it can clearly be seen from Figs. 2.7 and 2.8 that front view is the most problematic view. It contains more and longer occlusions, in fact nearly all frames in Fig. 2.7(d) contain occlusions of two or more fish, which makes it hard to keep track of the individual fish. It could be seen from the vertical gray lines, ID switches and missing detection most often occur when occlusions occur. From the above analysis, it is assessed that the object detector should be adjusted to be more precise and accurate to overcome most of the occlusion. Furthermore, the re-identification network should be tuned such that the number of ID switches is lowered. An assessment of the above reveals that minor performance improvements are achievable through further optimization of the object detector, while more substantial improvements are available for the re-identification model, as ID consistency is currently the major issue. In the following chapters, each component of Tracktor will be examined and adjusted in favor of zebrafish.

Chapter 3

Improving Faster R-CNN

From the analysis at the end of Section 2.3.1, it was assessed that the object detector should be optimized in favor of zebrafish instead of pedestrians. In this chapter, various approaches will be explained and tested to verify the impact of the customization. Different methods will be implemented to improve the tracking score. In the case of Faster R-CNN is was seen that better detections during occlusions are desired to keep track of the identity of the individual fish. In the first section, several data augmentation method will be employed to improve the robustness of the detection model followed by an analysis of the object detections on the validation set.

3.1 Data Augmentation

Tracktor is developed with simple data augmentation methods during training of the Faster R-CNN model. As mentioned earlier, Tracktor is built for pedestrian tracking, hence the only data augmentation used is horizontal flipping [3]. This makes great sense, when talking about humans, as these behave regularly and are close to symmetric. For instance, vertical flipping makes no sense, since pedestrians rarely walk with their feet upwards. It is another case with zebrafish, as these are allowed to swim in a three-dimensional space, the fish can be positioned and oriented in almost infinite possible ways. In the former case, vertical flipping might be an optimal solution for data augmentation when tracking the zebrafish from the top view. Five different data augmentation methods (see Fig. 3.1), including horizontal flip, are used during the training of Faster R-CNN.

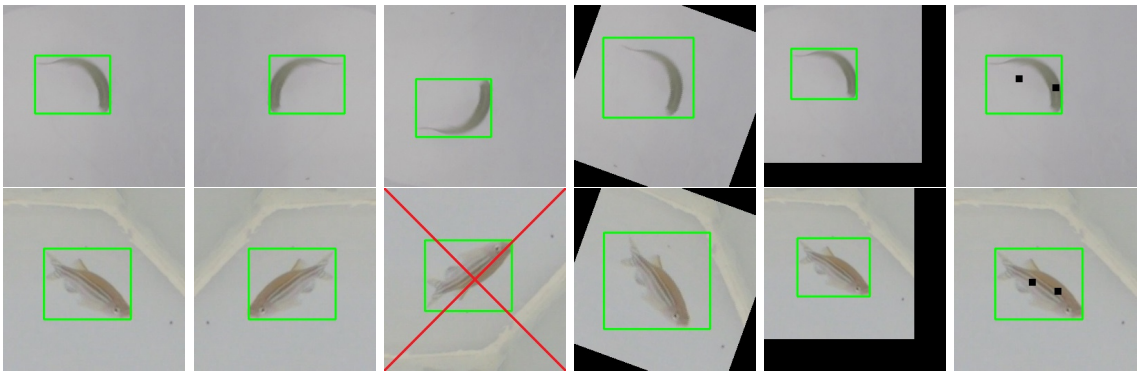


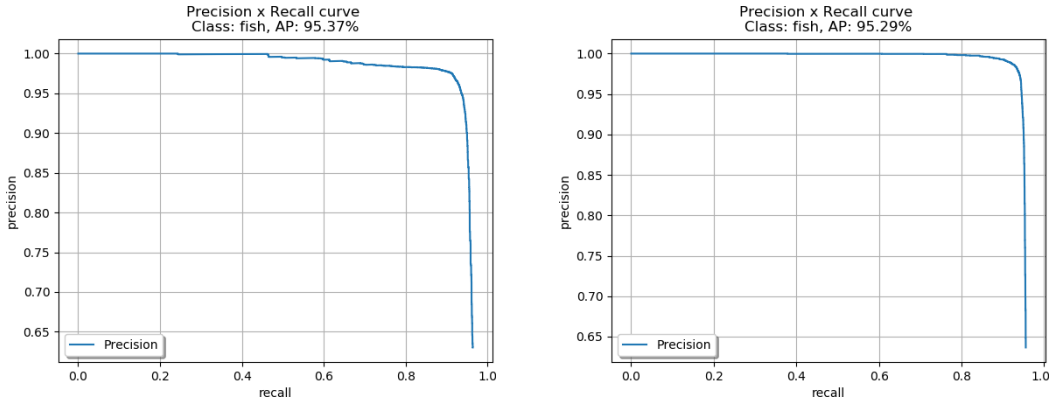
Figure 3.1: Illustration of the augmentations applied during training.

Vertical Flip is implemented in top view only since it is not normal behavior for a fish to swim upside down and such cases will not appear in front view. Random rotation is implemented to add additional instances where the fish is swimming upwards or downwards by rotating the image and adjusting the bounding box, as the fish can swim in all directions. Random changing the scale of the image is implemented to account for when a fish swim away or towards the camera, hence the size of the fish changes due to perspective in the field of view. And lastly randomly cutting out a small piece of the object is implemented to improve the understanding of the fish and as an attempt to make the object detector more robust to partial occlusions [37].

Retraining the object detector using the same hyperparameters as described earlier, together with the newly implemented data augmentations methods, provides two new detection models. Faster R-CNN is trained with:

- Random Horizontal Flip, with probability $p = 0.5$
- Random Vertical Flip, with probability $p = 0.5$
- Random Cutout, with probability $p = 0.2$
- Random Rotation, with probability $p = 0.2$ and $\theta = \{-10°, 10°\}$
- Random Scale, with probability $p = 0.2$ and $\sigma = \{0.8, 1.2\}$

First, we evaluate the object detector, for its precision/recall curve, which can be seen in Fig. 3.2. And the most important MOT metrics are evaluated and compared to the baseline in Table 3.1. In both cases the validation set is evaluated with the re-identification model attached.



(a) Top View (b) Front View

Figure 3.2: Precision / Recall curves for the Faster RCNN Object detector with new augmentation methods, plots generated using the method provided by [29].

	Top view					Front view				
	HOTA	MOTA	IDF1	ID_{swap}	Frag.	HOTA	MOTA	IDF1	ID_{swap}	Frag.
ZeF-03	**65.9%**	**95.4%**	**75.5%**	**8**	**14**	**48.1%**	**95.6%**	**43.8%**	8	**9**
ZeF-03 (Aug)	58.6%	94.0%	68.0%	9	27	48.0%	95.1%	43.7%	**7**	10
ZeF-04	**64.2%**	**88.0%**	**74.0%**	**7**	**41**	**40.5%**	**85.9%**	**38.4%**	35	50
ZeF-04 (Aug)	61.3%	85.4%	70.9%	10	56	39.8%	85.1%	35.8%	**33**	**38**

Table 3.1: Comparison of the most important key MOT metrics, both with re-ID step.

From Fig. 3.2, it can be seen that the object detector is well trained as previous, with

minor difference in regards to Fig. 2.4. This could indicate that the model cannot be trained to perform any better, as there is a certain limit to the performance of an object detector. An average precision of about 95% is rather high and it might be difficult to boost this score even further. Looking at Table 3.1, we can see that the object detector trained with more data augmentation did not perform any better on the validation set. It did perform slightly worse than the model without data augmentation. Therefore, the newly trained models are discarded and the baseline models will be used further on.

3.1.1 Controlling Precision/Recall Trade-Off

Another aspect worth investigating is to find the optimal confidence threshold for when the object detector should classify a region as positive. This threshold can usually be chosen by looking at the precision/recall curve as seen in Fig. 2.4. Here it is optimal to get as close as possible to the upper right corner of the graph, meaning that we have precision and recall both equal 1. However, this is rarely the case and we have to tune the threshold to fit the best trade-off between precision and recall. In most cases, when the confidence threshold is increased, the number of false positives is decreased, hence precision is increased. But increasing the threshold forces the detector to be more certain about a detection, which increases the number of false negatives, thus recall decreases.

One method for finding a potential confidence threshold is to plot the Cumulative Distribution Function (CDF) of the confidence scores for the True Positive detected objects on the validation set.

$$\text{CDF}(c) = \frac{1}{\|TP\|} \sum_{i=1}^{\|TP\|} \mathbb{1}(P_i < c) \tag{3.1}$$

where, CDF(c) is the cummulative distribution function for confidence threshold c, $\|TP\|$ is the number of true positive detections and P_i is the probability or confidence score for detection i.
This way we can see where the confidence for most of the detection are distributed, and we can see the fraction of True Positives for different thresholds. Increasing the threshold will lead to some of the true positives are being classified as false negatives but the number of false positives will also decrease, it has been chosen to use $P(X \leq c) = 0.05$ meaning that we save 95% of the true positive detections. In Fig. 3.3 we can see the distribution of the confidence scores.

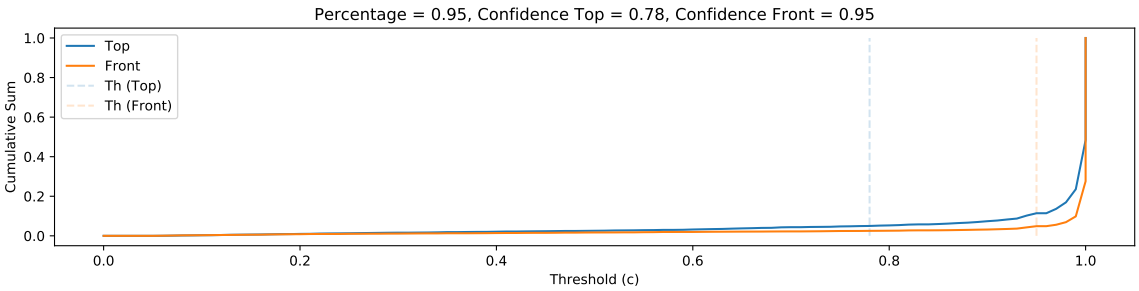
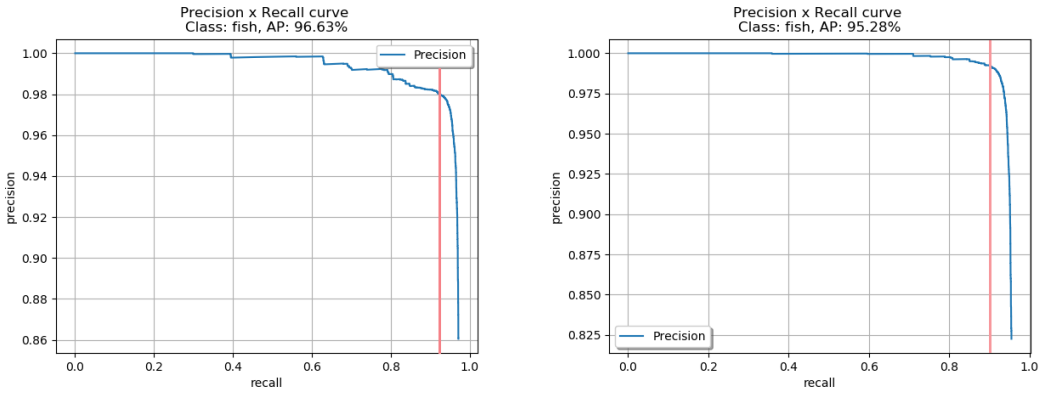


Figure 3.3: Cumulative Distribution Function for True Positive Detections in the validation set. The calculated thresholds are marked with vertical dashed lines, representing $P(X \leq c) = 0.05$ for both top and front view.

From the plot in Fig. 3.3, it can be seen that most of the true positives lie close towards 1, meaning that we can easily increase the confidence threshold for the object detector in both cases. From calculation above it could be found that $P(X \leq 0.78) = 0.05$ for top view and $P(X \leq 0.95) = 0.05$ for front view. Plotting the calculated thresholds on the precision/recall curve from Fig. 2.4, we can see that we obtain the point on the curve closest to the upper right corner. The plot with threshold drawn can be seen in Fig. 3.4.



(a) Top View (b) Front View

Figure 3.4: Precision / Recall curves for the Faster RCNN Object detector used with the calculated threshold, plots generated using the method provided by [29].

Originally the confidence threshold for Tracktor was set to 0.3, such that a confidence score above 0.3 is needed to establish a new track resulting in several false positive detections. Increasing the threshold might get rid of the false positives while introducing a few false negative detections. In an ideal setting, the tracker should be able to make up for the missing detections by keeping track of the object and eventually re-identify the track if detections are missing. In Table 3.2, the MOT scores can be found for the object detector with increased confidence threshold.

	ZeF-03					ZeF-04				
Detector	HOTA	MOTA	IDF1	ID_sw	Frag.	HOTA	MOTA	IDF1	ID_sw	Frag.
(T) Original	**65.9%**	95.4%	**75.5%**	8	**14**	**64.2%**	**88.0%**	**74.0%**	7	41
(T) (c = 0.78)	54.0%	**96.4%**	60.9%	**5**	17	61.4%	87.9%	71.3%	7	**33**
(F) Original	48.1%	95.6%	43.8%	8	9	40.5%	85.9%	38.4%	35	50
(F) (c = 0.95)	**49.7%**	**96.5%**	**43.9%**	**6**	**8**	**47.8%**	**88.2%**	**47.8%**	**22**	**38**

Table 3.2: MOT evaluation after specifying appropriate confidence threshold.

From Table 3.2, it can be seen that increasing the threshold in front view greatly increasing HOTA, MOTA and IDF1 for the validation set, while reducing the ID swap and fragmentations for about 25%. This is considered a great improvement to the front view system. On the other side, the top view didn't seem to benefit from increasing the threshold, in fact, most of the time the original threshold seemed to work best. Especially the IDF1 score did decrease the most, which could be an indication that top view is highly dependent on every detection to maintain the correct id, which is partly an issue relating to the re-identification network. If the top view is sensitive to missing detections, it means that the system has trouble re-identify detections. Either due to the fish is

changing appearance quickly or because no sufficient unique features are present to keep track of.

3.1.2 Conclusion

From the above analysis and test, it can be concluded that increasing the confidence threshold for when the object detector should consider a proposed region as a positive detection is beneficial for the front view system. However, in the top view, no significant improvement was found, hence the confidence levels should be set to 0.3 and 0.95 for the top and front view respectively. These values will be used in the following tests and evaluations.

Chapter 4

Improving Re-Identification

In general, re-identification models are often based on appearance modelling. An appearance model consists of two main components, a visual representation, describing the visual characteristics of the detected object and a statistical measure, which is a measure of the similarity between observations [24]. Deep metric learning is used as an appearance model in Tracktor. The visual representation is generated by exposing the detected object to a CNN which outputs an embedding. This embedding can be seen as a compressed version of the image, which summarizes the most important features of the object. In metric learning, we are learning an representation of the image, which are used in a distance metric to compute similarities. For computing the statistical measure, Tracktor utilizes the Euclidean distance for computing the distance between embeddings [3, 17]. This is a very complex task, as highly specialized models and loss functions are needed, this is described in detail in Section 2.2. However, these complex models come with a wide range of hyperparameters, which should be tuned for the application, in this case, zebrafish tracking. In this chapter, the different parts of the re-identification training and evaluation will be examined, starting with the Triplet Loss, followed by the backbone network ResNet50 adapted into TriNet, lastly the framework will be examined concerning thresholding values and patience of the re-identification process.

4.1 Adjusting Triplet Loss

Triplet Loss are a well-known loss function when doing deep metric learning. The loss itself is not superior alone, but with a proper sampling mining method, it can be extremely effective in learning a proper embedding space. Originally TriNet is trained using Triplet Loss with sampling hard batches, by choosing the hardest positive and negative instance within a mini-batch, given an anchor. In the work by Hermans, Beyer, and Leibe [17], they state that in most cases sampling hard batches is the optimal choice, but that the entire batch could be used too. This method is named *batch all* [17]. The equation is given below [17]:

$$\mathcal{L}_{\mathrm{BA}}(\theta; X) = \overbrace{\sum_{i=1}^{P}\sum_{a=1}^{K}}^{\text{all anchors}} \overbrace{\sum_{\substack{p=1 \\ p\neq a}}^{K}}^{\text{all pos.}} \overbrace{\sum_{\substack{j=1 \\ j\neq i}}^{P}\sum_{n=1}^{K}}^{\text{all negatives}} \left[m + d_{j,a,n}^{i,a,p}\right]_+, \tag{4.1}$$

$$d_{j,a,n}^{i,a,p} = D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_p^i\right)\right) - D\left(f_\theta\left(x_a^i\right), f_\theta\left(x_n^j\right)\right).$$

where, $\mathcal{L}_{\text{BA}}$ is the loss function with batch all mining, P is the number of unique identities, K is the number of unique images pr identity and $d_{j,a,n}^{i,a,p}$ is the distance metric similar to batch hard, accounting for all instances in the batch.
It should be noted that both the batch hard and batch all sampling methods correspond to the standard triplet loss, as this is not changing the loss function and both methods will turn out the same for infinite training. But for a finite set of epochs, choosing hard batches might be more effective. In Fig. 4.1, a small experiment has been conducted to verify this statement.

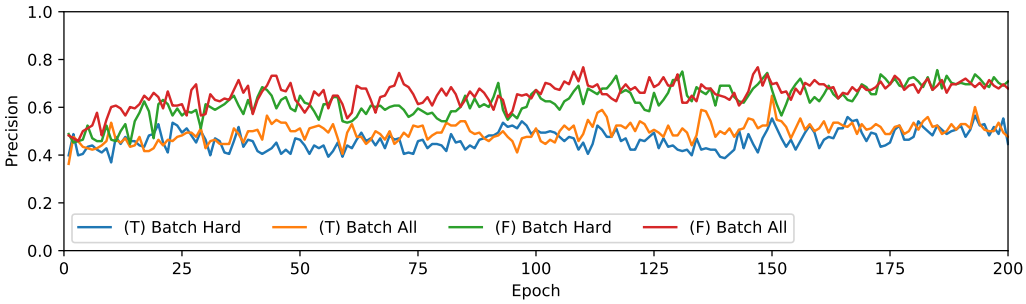
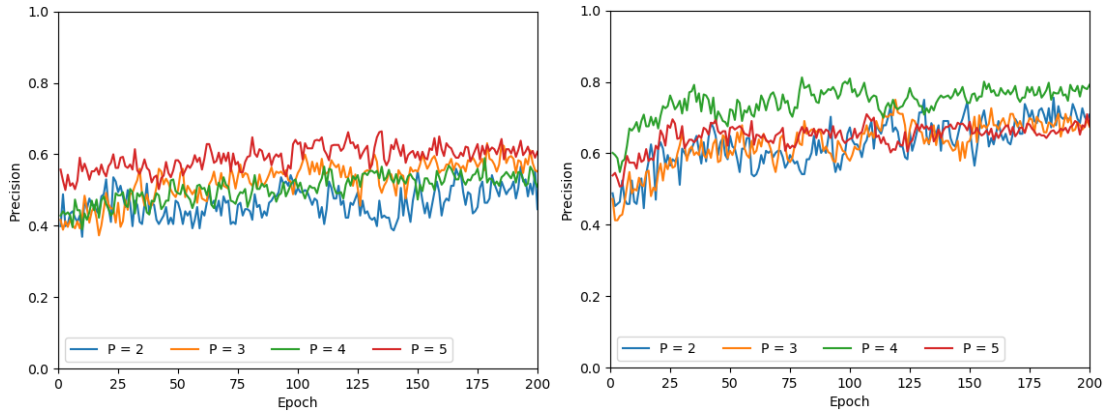


Figure 4.1: Experiment from investigating the impact of sample mining method.

From Fig. 4.1, it can be seen that, for both views, the curves for two sampling methods follow each other, ending up equally. Batch all seems to learn slightly faster than batch hard, but after 150 epochs they are approximately equal. This verifies the above statement that for long enough training these sampling methods turns out to be equal, but batch all sampling method introduces redundant information by training on easy positives and easy negatives. The above experiment indicates that batch hard sampling is sufficient for the application and batch all is omitted from further study.

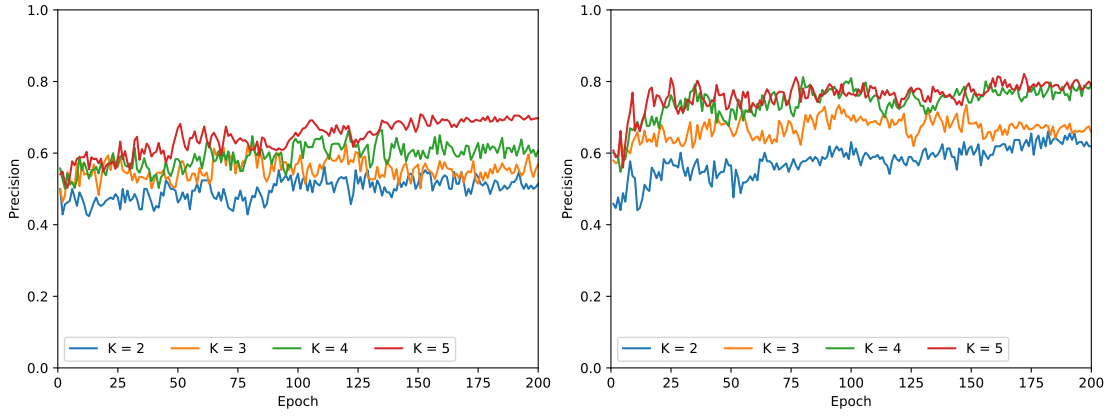
Training solely on easy examples results in zero loss, which does not contribute to any weight update, hence only hard samples are relevant for the training procedure. Training only on hard samples makes the network learning from outliers, and prevents the network from learning "normal" samples, therefore training on a mini-batch of the total dataset is desired. The size of this mini-batch is determined by the number of unique identities P and the number of unique images K for each identity. Using batch hard strategy generates PK triplets, meaning that increasing P and K increases the number of training examples, as well as increasing the diversity of the batch. In the work by Hermans, Beyer, and Leibe, they had to limit their batch size to 72, so they have chosen $P = 18$ and $K = 4$, a large number of persons is not a big issue when talking about pedestrian tracking, as these datasets contain plenty of persons. In 3D-ZeF20, the training set only contains in total 7 identities, which might overlap between sequences, due to the same fish might be on both ZeF-01 and ZeF-02. However, for simplicity, these seven identities are treated as seven unique fish for now. Running an experiment on the most optimal set of identities P and samples pr identity K, the results can be seen in Fig. 4.2.



(a) Top View (b) Front View

Figure 4.2: Experiment verifying impact of increasing the number of identities. A maximum of $P = 5$ is set due to memory constrains during training.

From Fig. 4.2, it can be seen that increasing the value of P increases the precision of the model. This means that introducing more identities in the training procedure increases the model's ability to generalize. Intuitively, this makes great sense as the model does not overfit that easy to the training data when including more zebrafish in the training. For top view $P = 5$ is performing best on the validation set and front view $P = 4$ seems to be perform slightly better than $P = 5$ for some unknown reason. In Fig. 4.3, the number of instances pr identity in the batch are examined.



(a) Top View (b) Front View

Figure 4.3: Experiment verifying impact of increasing the number of samples pr identity in the batch. A maximum of $K = 5$ is set due to memory constrains during training.

From Fig. 4.3, it can be seen that in both cases K = 5 seems to perform the best, which intuitively makes sense since the sample set within the loss function searches for hard samples is larger.

Training with Triplet Loss, a margin is specified to ensure a minimum distance between positive and negative samples, in previous work this margin is set to 0.2, and no effect about changing this margin is reported.

Investigating if increasing the minimum distance between positive and negative embeddings influences the validation precision. In Fig. 4.4 the validation precision during training is reported.

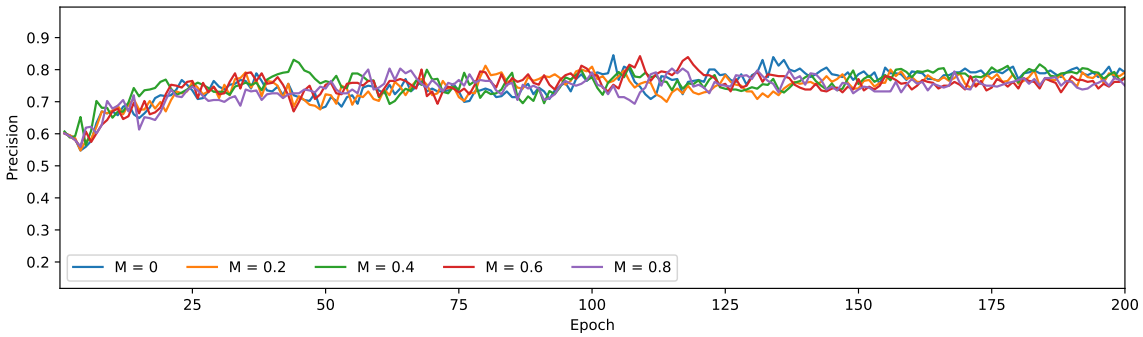
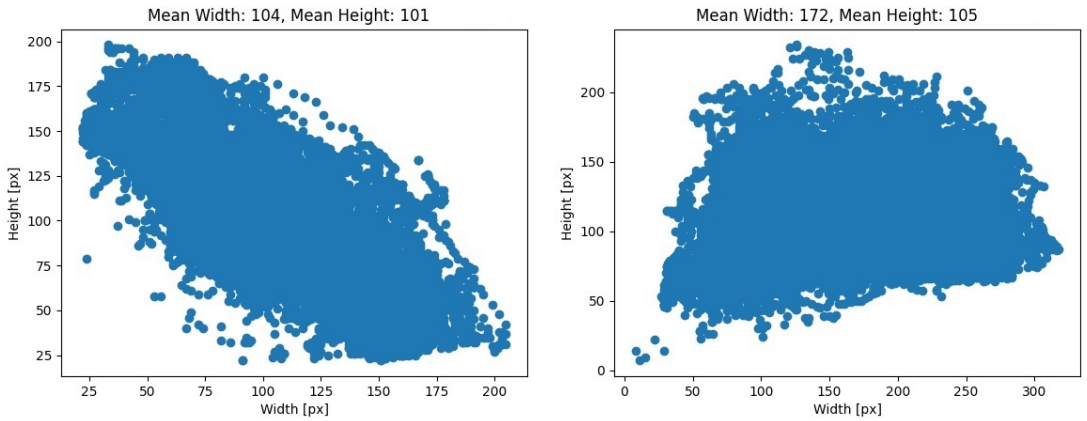


Figure 4.4: Experiment investigating the magnitude of the margin's influence on the validation loss.

From Fig. 4.4, it can be seen that no real change happens to the validation precision during training when adjusting the margin in Triplet loss. Quite interesting, is that when a margin set at $M = 0$ the training does not collapse, and outputs all zeros as mentioned in [17]. This could be due to the pretrained weights used for initialization. Originally, TriNet is derived by using a ResNet50 with pretrained weights from ImageNet. This could eventually prevent the network from collapsing. However, for simplicity and safety, a margin of 0.2 as suggested by Hermans, Beyer, and Leibe [17] is maintained. In the following section, parameters related to the backbone network will be evaluated.

4.2 Tuning TriNet

TriNet, the backbone network used for extracting features used for generating embeddings is built upon ResNet50 [16]. This network takes an input shape of 128×256 pixels. This shape is highly suitable for humans, as the aspect ratio is used in many human detection applications, such as the HOG descriptor [11]. However, zebrafish do rarely fit the shape of a human, hence this input shape might not be the optimal choice. In Fig. 4.5, the distribution of the width and height of the ground truth annotations in the training set can be seen.



(a) Top View (b) Front View

Figure 4.5: Scatter plot of the annotated bounding boxes in ZeF-01 and ZeF-02.

From the analysis in Figs. 4.5a and 4.5b it is seen that top view is very close to square with a aspect ratio of $W/H \approx 1$. On the contrary, front view is more rectangular $W/H \approx 1.63$. This is evident looking at the fish from the side it is longer than it is tall. Changing the input shape of TriNet to fit the zebrafish natural shape, changes have to be made. First of all, the network architecture is designed for the shape of 128×256 pixel. This means changes to layers are needed to fit a new input shape. In the upper layers, right before embeddings are outputted, an average pooling layer is found, this should be changed to an adaptive average pooling, such that it can adapt to the input size. Furthermore, to contain an equal amount of pixels in the region of interest (ROI), we fix the number of pixels and calculate a new shape to fit the desired aspect ratio. The total number of pixels in the original ROI is $128 \times 256 = 32\,768$ pixels, therefore $W_{new} \cdot H_{new} = 32\,768$, under the constrain that $W_{new} = aH_{new}$, where a is the aspect ratio. The new shapes are found by solving the following equation:

$$W_{new}^2 = a \cdot \text{pixels} \rightarrow W_{new} = \sqrt{a \cdot \text{pixels}} \tag{4.2}$$
$$S_{\text{top}} = (181, 181), \qquad S_{\text{front}} = (231, 141)$$

where, W_{new} and H_{new} is the size of the new input shape S and a is the ratio between width and height. Implementing this into the pipeline, give the performance seen in Fig. 4.6.

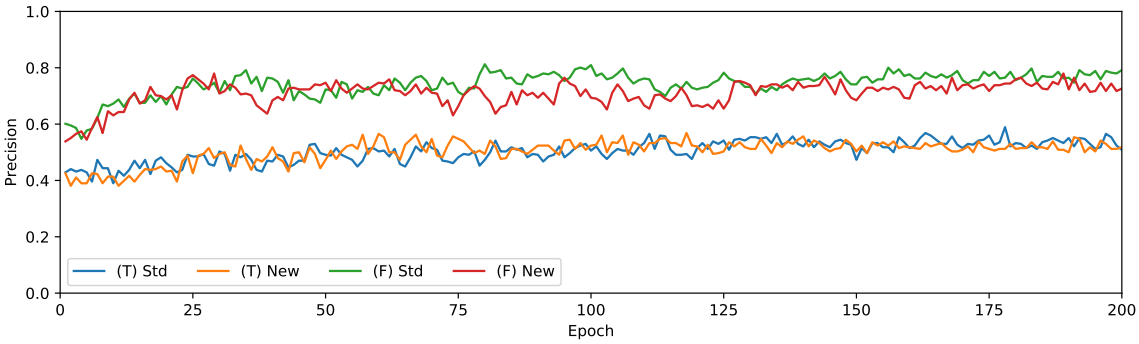
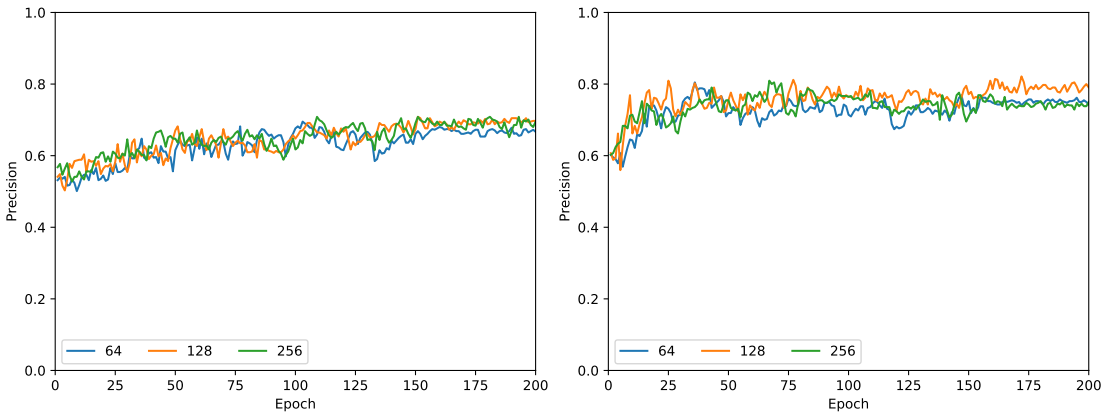


Figure 4.6: Experiment examine the impact of setting correct input shape, to fit zebrafish instead of pedestrians. (T) = top view and (F) = front view and "Std" is the standard input shape (128×256) and "New" is the calculated shape, for the respective view.

From the experiment it can be concluded that changing the input shape, does not affect the performance positively, in front view it perform a bit worse than the original shape. No sufficient evidence is found to keep the new input shape, and for simplicity and consistency we have chosen to keep the original shape. It seems like the extracted features are not dependent on the aspect ratio but on the content of the image.

Currently, the embedding is a 128-dimensional vector, extracted by a CNN, and used as an encoded version of objects. The size of this vector is a hyperparameter central to metric learning and should be tuned to fit the application. We have adopted the same procedure as in [2, 36, 38], by adjusting the embedding size between {64, 128, 256} while observing the performance. One would argue that the larger the embedding the more information is kept, hence it should be easier to re-identify. On the other hand, increasing the dimensionality introduces irrelevant information, which might make it harder for the model to compare the similarity and eventually overfit the training data. Lowering the dimension to a certain limit might force the network to learn robust features,

generating sparse but powerful embeddings. An experiment has been conducted to verify this statement, and the results can be seen in Fig. 4.7.



(a) Top View (b) Front View

Figure 4.7: Results from experiment analyzing performance for a given dimensionality for both views.

In Fig. 4.7a no big difference in performance is seen among the dimensions. A 128-dimensional vector is minimally better than the others. In Fig. 4.7b the front view is evaluated and a decent advance in performance, when using a dimensionality of 128, is seen. Keeping the standard output shape of the embedding vector. This conclusion is similar to what is observed in [36]. A 128-dimensional embedding seems to be a common solution in metric learning and will be used in this work as well.

The last options we are presented is to employ transfer learning. This means, when using existing architectures such as ResNet50, it is possible to use pretrained weights, often from the ImageNet dataset [12]. Doing so makes it possible to initialize the model with robust weights, used for extracting various features, which could be fine-tuned to fit the application. In nearly all cases pretraining improves the performance and reduces the potential for overfitting. In Fig. 4.8 the results of the experiment can be seen.

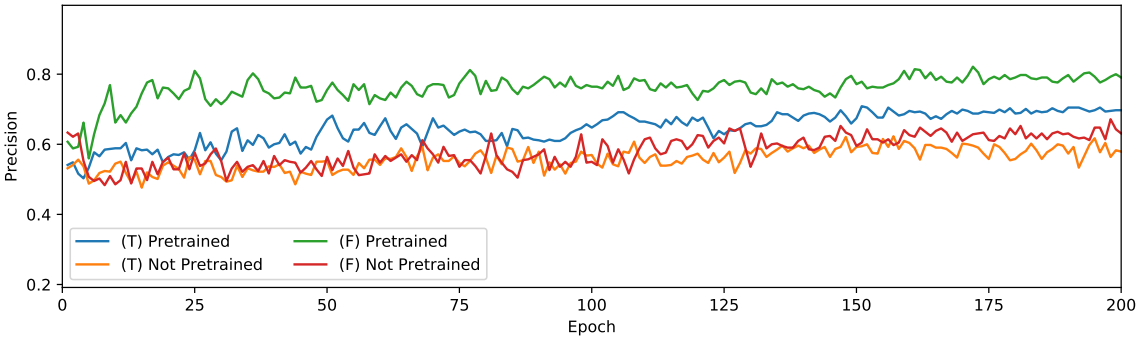
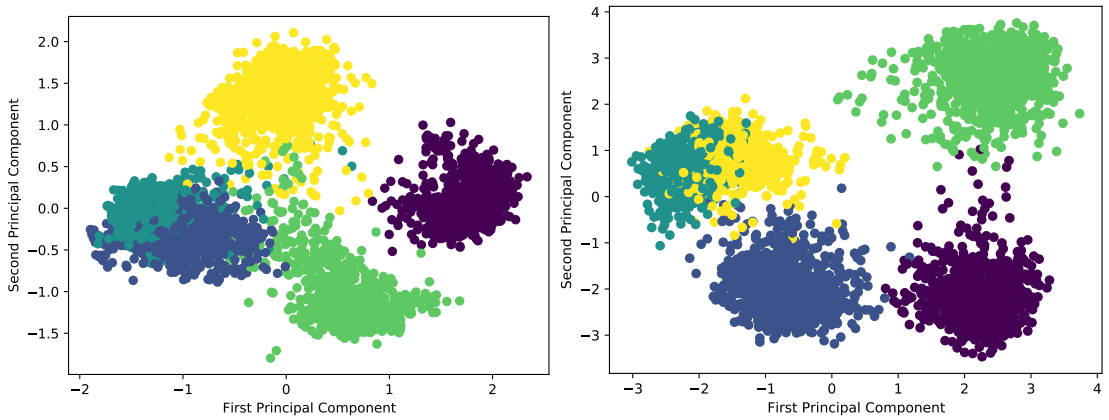


Figure 4.8: Experiment conducted by training two models with equal settings, one is pretrained and one is not. (T) = top view and (F) = front view.

From Fig. 4.8, the benefit of pretrained can easily be seen. In both views, the precision is significantly higher than the model without pretraining. The biggest improvement is seen in the front view (green line). Very interesting is that both models without pretraining seem to oscillate around a precision of 0.5, which is not better than random guessing. It

seems like the models are not learning, or at least very slowly. This is somehow expected, as training from random initialization, requires a longer training time to learn sufficient features. After all, pretraining is preferred and will be used further on.

Now that the final models are tuned and trained in a way that emphasizes zebrafish, embeddings can be generated and visualized. It is expected that embeddings of the same identity are clustered together, with a margin between other identities. The embedding is, as mentioned, a 128-dimensional vector, and is difficult to visualize, therefore Principal Component Analysis (PCA) has been used. PCA is used to reduce the dimensionality of the vector by projecting the vector from a high dimensional space $\mathbb{R}^{128}$ down to a lower-dimensional space $\mathbb{R}^2$ while preserving maximum variance. In Fig. 4.9, the reduced embeddings can be seen.



(a) Top View (b) Front View

Figure 4.9: Embeddings for the five zebrafish in ZeF-02 projected down to 2D plane using PCA, each color represent a distinct ID.

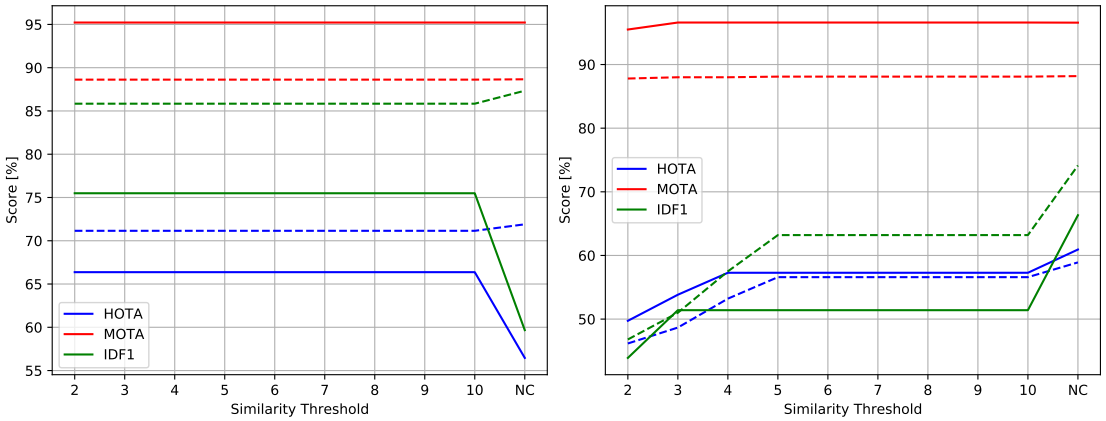
From Fig. 4.9 it is clear to see that embeddings from the same fish are clustered together, meaning that the model performs as expected on the training set. Before moving on to the evaluation of the re-id models during tracking, the framework will be optimized to fit the purpose of zebrafish tracking by applying domain knowledge. This will be examined in the following section.

4.3 Optimizing Re-Identification

Re-identification is constrained in two ways, the time we keep searching for a lost object. Patience is the time we will keep the tracker alive for, trying to re-identify new detected and untracked fish. Secondly, a threshold is defined to ensure a certain similarity is needed before re-identification can take place.

Currently, the patience is set to 50 frames which correspond to $\approx$ 833 ms with a framerate of 60 fps. According to the complexity analysis in [30], the longest average occlusion length is $\approx$ 660 ms for front view in ZeF-04, this indicate that patience is not needed to be tuned further and 50 frames will be used further on. A similarity threshold is implemented to ensure deactivated tracks are not connected to a new object entering the camera view. In this dataset, no new fish entering the camera view and hence the threshold could be loosened. If a fish has lost its track, and a new fish is detected, it is probably the

same fish or false-positive detections. To minimize the influence of false-positive detections and the risk of false reIDs a simple IoU metric is implemented to account for the spatial position. Hence only detections with a minor overlap are considered as potential reID candidates. In Fig. 4.10 an experiment has been conducted to find the optimal threshold setting for the framework.



(a) Top View (b) Front View

Figure 4.10: Experiment of similarity threshold, performance is measured in HOTA, MOTA and IDF1. NC in the horizontal axis indicate a non constrained experiment where the threshold is set to infinity. Solid lines are validation set ZeF-03 and dashed lines are validation set ZeF-04.

From the figures in Fig. 4.10, it can be concluded that raising the threshold improves the tracking performance, especially in front view a prominent performance boost is seen. In Fig. 4.10a we see a decrease in performance for ZeF-03 in the unconstrained setting. However, in the majority of cases, the performance is increased. In ZeF-03 two fish are present and five fish in ZeF-04. The sequence with the most fish does weigh more in the conclusion as ZeF-04 has a higher complexity than ZeF-03 and therefore a rise in performance for ZeF-04 is highly desired. It can be seen that MOTA is not affected in top view, hence it is a matter of identification solely. It is chosen to use the unconstrained re-identification model (threshold = ∞) for future work. In Table 4.1 the findings from above analysis are summarized. The parameters are more or less the equal in both views, except for the number of identities in the batch (P) when training with Triplet Loss.

	Mining	M	P	K	IS	Emb.	Pretrained	Patience	L2 Thresh
Top	Batch Hard	0.2	5	5	(128 x 256)	128	True	50	∞
Front	Batch Hard	0.2	4	5	(128 x 256)	128	True	50	∞

Table 4.1: Overview of the tuned parameters, for both views. Identical parameters is found except the value of P.

Evaluating the optimized framework on the validation set and comparing with the baseline results, gives the following results as seen in Tables 4.2 and 4.3.

	HOTA	MOTA	MOTP	IDF1	IDR	IDP	Prc	Rcll	FP	FN	MT	ML	ID_{swap}	Frag.	MTBFs	MTBFm
ZeF-03	**65.9%**	**95.4%**	0.133	**75.5%**	**75.4%**	**75.7%**	**0.98**	**0.976**	**72**	86	2	0	8	14	**195.4**	103.4
ZeF-03_{new}	56.46%	95.22%	**0.131**	59.7%	59.61%	59.73%	0.978	0.976	79	86	2	0	**7**	14	195.3	103.4
ZeF-04	64.2%	88.0%	0.179	74.0%	73.4%	74.5%	0.948	0.933	235	303	5	0	7	41	91.0	48.0
ZeF-04_{new}	**71.9%**	**88.66%**	**0.178**	**87.34%**	**86.66%**	**88.03%**	**0.95**	**0.936**	**222**	**293**	5	0	**1**	**40**	90.9	48.0

Table 4.2: Tracking results on validation data from top view.
Results presented in top are from the baseline network with reid model in Table 2.1

	HOTA	MOTA	MOTP	IDF1	IDR	IDP	Prc	Rcll	FP	FN	MT	ML	ID_{swap}	Frag.	MTBFs	MTBFm
ZeF-03	48.1%	95.6%	0.116	43.8%	43.9%	43.8%	0.979	**0.98**	77	**72**	2	0	8	9	252.0	135.7
ZeF-03_{new}	**60.93%**	**96.58%**	**0.114**	**66.32%**	**65.89%**	**66.76%**	**0.989**	0.977	**36**	83	2	0	**4**	**8**	**293.1**	**159.9**
ZeF-04	40.5%	85.9%	0.150	38.4%	37.8%	39.0%	0.946	**0.919**	238	**369**	5	0	35	50	71.0	37.4
ZeF-04_{new}	**58.92%**	**88.18%**	**0.146**	**74.14%**	**71.17%**	**77.37%**	**0.98**	0.901	**81**	446	5	0	**11**	**38**	**100.2**	**53.3**

Table 4.3: Tracking results on validation data from front view.
Results presented in top are from the baseline network with reid model in Table 2.2

It can be seen that especially the performance in front view has been increased the most. As touched upon earlier, it was assessed that re-identification was the largest obstacle and the area where most performance boost could be achieved. From the above results, it could be seen that an increment in IDF1 directly led to an increase in HOTA. Furthermore, the number of IDswaps has been decreased significantly as well the mean time between failure has increased. In Figs. 4.11 and 4.12 an updated version of Figs. 2.7 and 2.8 at Page 21. Here the ground truth objects are plotted with the associated ID represented by distinct colors.

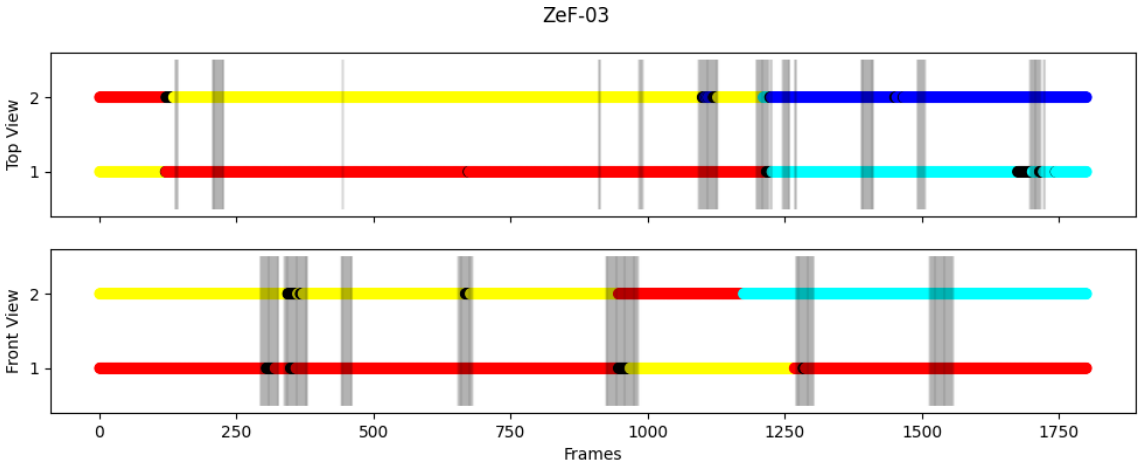


Figure 4.11: Evaluation of ZeF-03 using the improved version of Tracktor. Vertical gray lines indicate an occlusion had happened in that particular frame. Each color represent a unique ID, and a change from one color to another indicates ID switch. Black means missing detection. Integers on the vertical axis represent ground truth annotated ID.

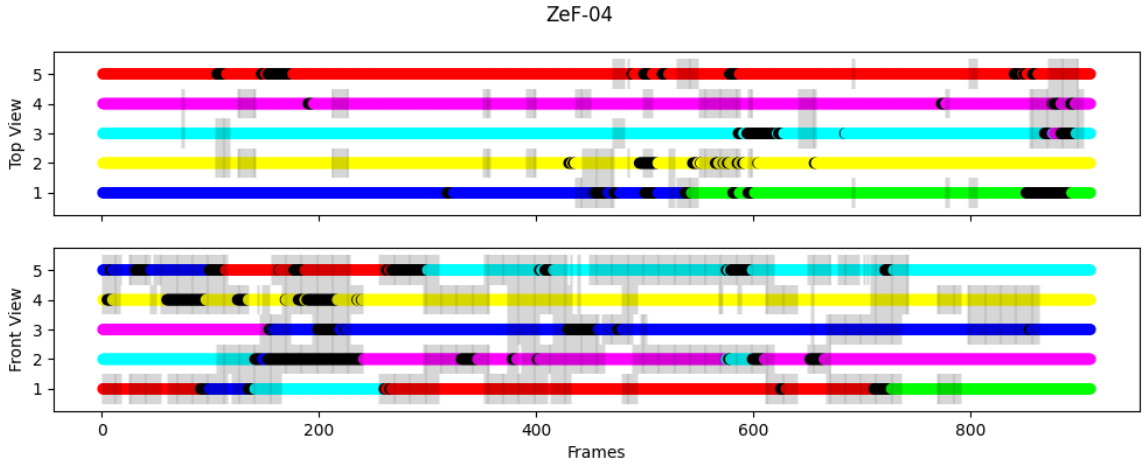


Figure 4.12: Evaluation of ZeF-04 using the improved version of Tracktor. Vertical gray lines indicate an occlusion had happened in that particular frame. Each color represent a unique ID, and a change from one color to another indicates ID switch. Black means missing detection. Integers on the vertical axis represent ground truth annotated ID.

An improvement can be seen both visually in Figs. 4.11 and 4.12 and objective improvement using the MOT metrics in Tables 4.2 and 4.3. Unfortunately, we can observe minor ID swaps in the sequences compared to the ground truth annotated data. But in most cases, we have consistent long tracklets in either top or front view. In Chapter 6 we will investigate if this could be used to correct tracklets in the opposing view. In Tables 4.4 and 4.5 the performance on the validation set are compared to the performance on the validation set in the benchmark paper [30].

	HOTA	MOTA	MOTP	IDF1	IDR	IDP	Prc	Rcll	FP	FN	MT	ML	ID_{swap}	Frag.	MTBFs	MTBFm
Naive	-	59.8	0.969	48.9	57.1	42.8	0.726	0.967	1315	118	2	0	13	44	68.275	36.653
FRCNN	-	**96.8**	0.978	**76.3**	**75.9**	**76.8**	**0.992**	**0.979**	**30**	**76**	2	0	10	17	167.810	92.737
Our	**56.46%**	95.22	**0.131**	59.7%	59.61%	59.73%	0.978	0.976	79	86	2	0	**7**	**14**	**195.3**	**103.4**
Naive	-	89.4	0.964	50.6	51.7	49.6	0.931	0.970	330	135	5	0	19	36	92.083	52.619
FRCNN	-	**96.4**	0.972	48.8	48.3	49.4	**0.995**	**0.972**	**22**	**128**	5	0	15	**23**	**147.567**	**83.528**
Our	**71.9%**	88.66	**0.178**	**87.34%**	**86.66%**	**88.03%**	0.95	0.936	222	293	5	0	**1**	40	90.9	48.0

Table 4.4: Comparison with the 2D top results from the benchmark paper [30], it should be noted that these results cannot be directly compared due to minor changes in the sequence by the author of the dataset, the results are still representative for the sequence.

	HOTA	MOTA	MOTP	IDF1	IDR	IDP	Prc	Rcll	FP	FN	MT	ML	ID_{swap}	Frag.	MTBFs	MTBFm
Naive	-	-93.1	0.870	0.8	0.8	0.8	0.037	0.036	3444	3469	0	2	37	28	3.119	1.795
FRCNN	-	93.6	0.958	13.1	12.9	13.3	**1**	0.974	**0**	95	2	**0**	136	24	25.036	21.372
Our	**60.93%**	**96.58%**	**0.114**	**66.32%**	**65.89%**	**66.76%**	0.989	**0.977**	36	**83**	2	0	**4**	**8**	**293.1**	**159.9**
Naive	-	-68.8	0.875	3.1	2.9	3.4	0.1	0.085	3480	4162	0	5	37	44	6.690	3.464
FRCNN	-	79.0	0.957	17.8	16.2	19.7	0.996	0.818	**14**	826	3	0	117	72	26.791	17.402
Our	**58.92%**	**88.18%**	**0.146**	**74.14%**	**71.17%**	**77.37%**	**0.98**	**0.901**	81	**446**	**5**	0	**11**	**38**	**100.2**	**53.3**

Table 4.5: Comparison with the 2D front results from the benchmark paper [30], it should be noted that these results cannot be directly compared due to minor changes in the sequence by the author of the dataset, the results are still representative for the sequence.

From Tables 4.4 and 4.5 it is apparent that using tracktor for tracking zebrafish does impact the front view the most, improvements in almost every metric is found. The top

view is not affected that much and performance similar to the performance found in the benchmark paper is obtained, with a slight drawback.

To compare with the benchmark tests, we need to obtain 3D tracklets to test the method on the test set, as the annotations for the test set consists of 3D positions. In the following chapter estimation of 3D locations based on 2D information will be described and results on the benchmark test set will be presented.

Chapter 5

Tracking in 3D

Originally, the dataset 3D-ZeF20 [30], is built for tracking in 3D, and currently the two views have been handled separately by tracking the zebrafish in two dimensions for each view. To compare the results with the benchmark, it is necessary to project the tracklets to 3D space, by using the sets of 2D tracklets. To triangulate the detections and tracklets to match tracklets between views, the software developed for this benchmark will be used. To triangulate detections, the detected bounding boxes must be translated into a single point in 2D. Intuitively, the center of the bounding box would make an obvious choice, but problems arise. When the fish is turning around in the top view, the center of the bounding box is not located at the fish, which makes it difficult to find a corresponding point in the front view. As stated by the author of the dataset, the head of the fish is the only rigid part of the fish, hence this is considered the most robust and consistent triangulation point. See, Fig. 5.1.

(a) Center (b) Head

Figure 5.1: Potential points to used for 3D triangulation, where the green box is the detected bounding box and the blue point is the point used for triangulation.

Using the head as a tracking point in both views seems to be the optimal choice, as we are interested in the smallest projection error. To localize the head of the detected fish, we will reuse the module provided by [30][1]. Using the official implementation makes the tracking performance independent of the head localization and projection method. Hence we can ensure that an increase in performance in regards to the benchmark, is caused by the improved 2D tracks provided by Tracktor. In the following section, the method for head localization will be described briefly.

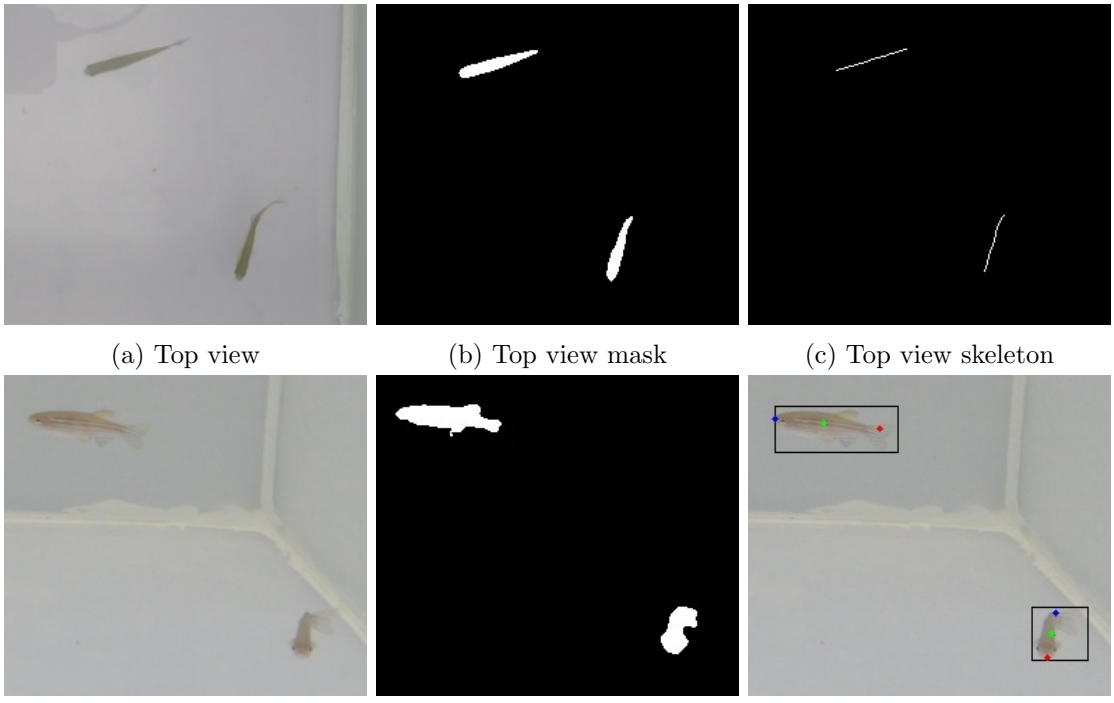
[1] `www.github.com/mapeAAU/3D-ZeF`

5.1 Head localization

Localizing the head is not a trivial task, but with appropriate assumptions, the position can be estimated very well. In the top view, the fish can be segmented using background subtraction together with bimodal thresholding, as the image histogram is assumed to be bimodal. The segmentation mask will be reduced using the skeletonization method by Zhang and Suen [42], by iteratively removing the contour of the mask, until a single-pixel line is left. Endpoints from the skeleton are extracted and the endpoint with the largest mass will be used as head position, under the assumption that the head is larger than the tail [30].

The front view is tackled a bit differently, and are a bit harder to determine the head position. Due to the translucent skin and texture of the zebrafish, the histogram cannot be considered bimodal. Histogram entropy thresholding is used to segment the fish in the front view, and three points are extracted as potential head positions. From the segmentation mask the leftmost and the rightmost part, along the major axis of the bounding box, is extracted. Together with the center of the mask, these three points are potential head positions, and all three points will be used in the 3D reconstruction module, where the two points with the largest reprojection error will be discarded [30].

This process is illustrated in Fig. 5.2, where the two top images show the process from top view, and the two images below illustrates the method for front view.



(a) Top view (b) Top view mask (c) Top view skeleton

(d) Front view (e) Front view mask (f) Front view keypoints

Figure 5.2: Extraction of potential points used for head localization, top view skeletonization is used and front view three points are extracted and used in further evaluation.

When head position estimates are obtained for each detected fish in both views. These points can be triangulated and matched, such that a 3D position for the head of the fish can be estimated.

5.2 Projection 2D into 3D

Now that all detections have been translated into 2D points, these points can be triangulated to match points between views and together form a 3D position of the fish's head.

Moving from separate 2D coordinate systems, and trying to align these in 3D space, often named world coordinates, we need to perform camera calibration of each camera. This is handled by a projection matrix, projecting 3D world coordinates to 2D image coordinates using the equation below [14]:

$$x = PX \tag{5.1}$$

where x is the position in the image, P is the projection matrix and X is the position in world space.
This projection matrix is made up of intrinsic and extrinsic parameters, which describes the internal and external properties of the camera respectively. Intrinsic parameters contain information about the focal length and account for fabrication inaccuracies between the camera center and the image plane. These parameters are associated with the respective camera and are fixed, therefore we could use the calibration used in the benchmark [30]. Extrinsic parameters describe the relationship between the camera and the world. The matrix describing the extrinsic parameters is made up of a rotation and translation [14].

The projection matrix can be constructed by the intrinsic parameters K and extrinsic rotation and translation R and t.

$$P = K[R \mid t] \tag{5.2}$$

The projection matrix P can be estimated by pair of fixed points in world and image coordinates, in this case, the corner of the aquarium intersecting with the water level are used as reference. All calibration information is public available with the benchmark dataset[2].

Now we have calibrated and aligned cameras, we can start projecting the detections into 3D world coordinates, this is carried out by transforming the detected points into rays. Rays are used to account for the unknown z coordinate describing the depth of the object. Every point on that line would be projected into the same image coordinate and therefore we need to do the same for the other view. Where these rays intersect, the position in world space is located.

In air, this ray is a straight line, but when changing medium (from air to water) refraction occur. This refraction bends the ray when hitting the aquarium and needs to be accounted for when triangulating points. This phenomena is illustrated in Fig. 5.3.

[2] `www.motchallenge.net`

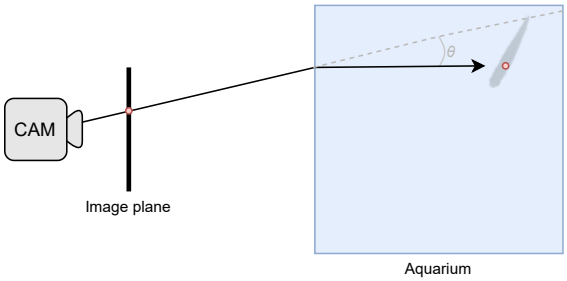


Figure 5.3: Illustration of the refraction problem, where the ray is bend when changing medium from air to water. The gray dashed line indicate the direction of the ray in air.

The angle at which the ray is bend, is determined by the refractive index, more precisely, the ratio between the two indexes. When an intersection is found, the world coordinates for the fish is estimated, and to measure the exactness of the projection, the world coordinates can be reprojected back to the respective views. This is illustrated in Fig. 5.4, where most of the terminology used further on are marked and illustrated.

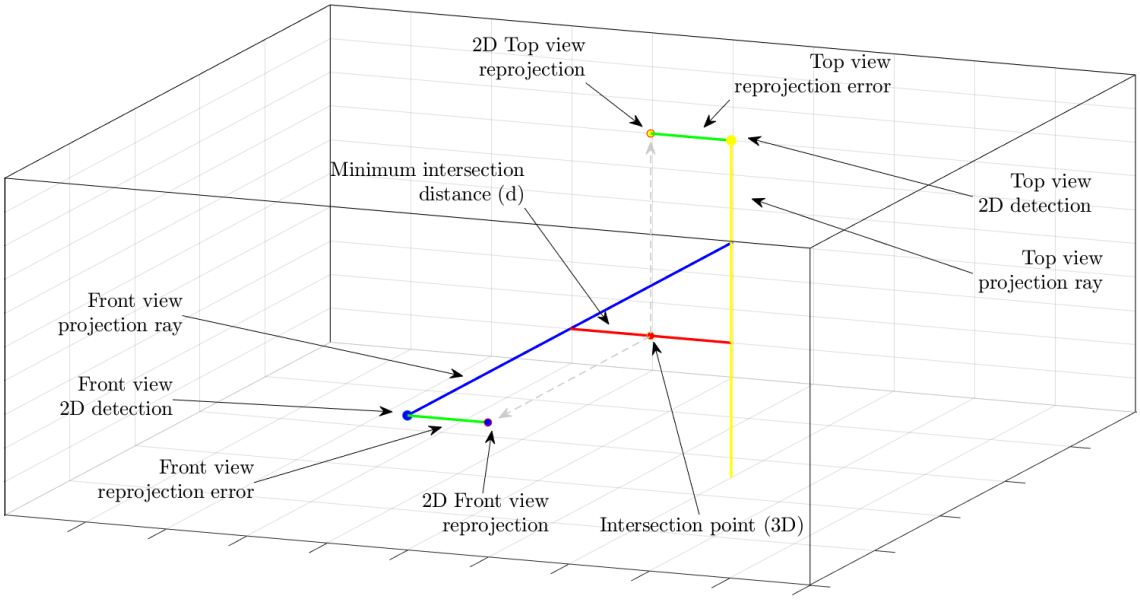


Figure 5.4: 3D triangulation using rays, for simplicity refraction between air and water is omitted. The big blue dot is the detected point in the front view and the small dot is the reprojected point, similar to the yellow in the top view. The red dot is the 3D estimated intersection point, where the distance between the blue and yellow ray is minimized. This distance is marked in red. The distance between the detected point and reprojected point is called the reprojected error and describes the error of projection between views.

When the 3D point is reprojected back to the 2D view in either front or top view, the reprojected point (small dot) and the detected point (big dot) can be compared and the distance between these points is called the reprojection error. This error is computed as follows:

$$d(p,r) = \sqrt{(r_1 - p_1)^2 + (r_2 - p_2)^2} \tag{5.3}$$

where p is the original point, r is the reprojected point and d is the euclidean distance between two points. The reprojection error is an important measure of the fit between a

pair of points from the top and front view. A large error might indicate that these points do not match well, and a small error indicates a perfect match. Projecting all the 2D tracklets into 3D tracklets and then combining all tracklets into 3D trajectories, makes it possible to submit and evaluate on the benchmark dataset. In the following section, the results will be presented.

5.3 Benchmark Test Results

Evaluating tracktor as tracking module for 2D tracklets, is carried out by testing the 3D trajectories estimated in previous section. In Table 5.1 the performance can be seen.

	MOTA	MOTP	GT	IDF1	IDR	IDP	Prc	Rcll	FP	FN	MT	PT	ML	ID_{swap}	Frag.	MTBFs	MTBFm
ZeF-05	67.67%	0.656	1	81.3%	70.4	96.2	96.21	70.44	25	266	0	1	0	0	25	24.38	12.19
ZeF-06	8.61%	0.633	2	42.2%	34.9	53.5	56.78	37.00	507	1134	0	2	0	4	49	13.06	6.59
ZeF-07	18.93%	0.659	5	44.7%	33.7	66.3	68.74	34.93	715	2928	0	4	1	5	102	14.69	7.31
ZeF-08	4.30%	0.665	10	37.7%	29.2	53.4	53.97	29.44	2260	6350	0	7	3	3	167	15.06	7.36

Table 5.1: Test results obtained from submitting the 3D trajectories to MOT Challenge.

From Table 5.1, we can see that ZeF-05 got a decent performance, where the rest of the sequences showed unsatisfactory performance. Investigating the results even further, we find that the Recall is relatively low together with the FN detections is quite high. This might indicate that we are missing plenty of detections in 3D. Interestingly enough, we have lots of false-positive detections too, which is a sign of a false triangulation match between the views. In the following section, these results will be discussed and strategies for improving the performance will be presented.

5.3.1 Discussion

At first, it seems like we are missing a lot of detections, but in this case, detections are points triangulated into 3D estimates. Missing such must be either because we do not overlap the ground truth position enough, or because we are missing 2D detections in either top or front view. Then after examining the results, even more, seeing that we have a large portion of false-positive detections, this supports the idea that maybe we match the wrong detections in between views. If we choose a pair of detections (top and front) which does not belong to each other, we get a skewed 3D estimate and the reprojection error becomes large. This can happen if the identity of a tracked fish switches with another fish, then we would try to triangulate the detections with two different fish which results in noisy 3D estimates. According to the authors of the benchmark dataset [30] and the corresponding public available pipeline[3], ID swaps in 2D views is very harmful to the pipeline. The pipeline is built by assuming shorter but confident tracklets, meaning that by observing an ID swap, the assumption is not met. If so, the detections used for triangulation will be incorrect and may result in FP detections and then lots of correct 2D detections are lost.

To accommodate the assumption of a minimal number of ID swaps in the 2D tracklets, we are interested in a method for finding ID swaps in the tracklets before estimating 3D positions. If we could find ID swaps then it might be possible to correct the tracklet by

[3] `www.github.com/mapeAAU/3D-ZeF`

switching back the identity or simply break the tracklet up into two short tracklets with different IDs. In the work, by Pedersen et al. [30], it is stated that the pipeline does not handle many short tracklets very well. Therefore, the solution by switching back identities is of interest.
In the next chapter, a method will be proposed to detect and correct ID swaps in 2D view by taking advantage of the opportunities given by recording the fish from multiple views.

Chapter 6

Multi-view Information Sharing

In Sections 5.3 and 5.3.1 we found that ID swaps are a major problem when projecting tracklets into 3D, as the framework assume shorter but confident trackelts and then greedy assemble larger tracks by comparing the temporal overlap and intersection between views, which makes ID swaps extremely harmful. In this section, we will investigate if we could use 3D information to correct mistakes made in 2D, by using information from the other view. Swapping identity between fish, for instance during occlusions is not uncommon, but prevent consistent tracks, especially in 3D. In Fig. 6.1 the problem is illustrated, this illustration will be used as the example when describing the method.

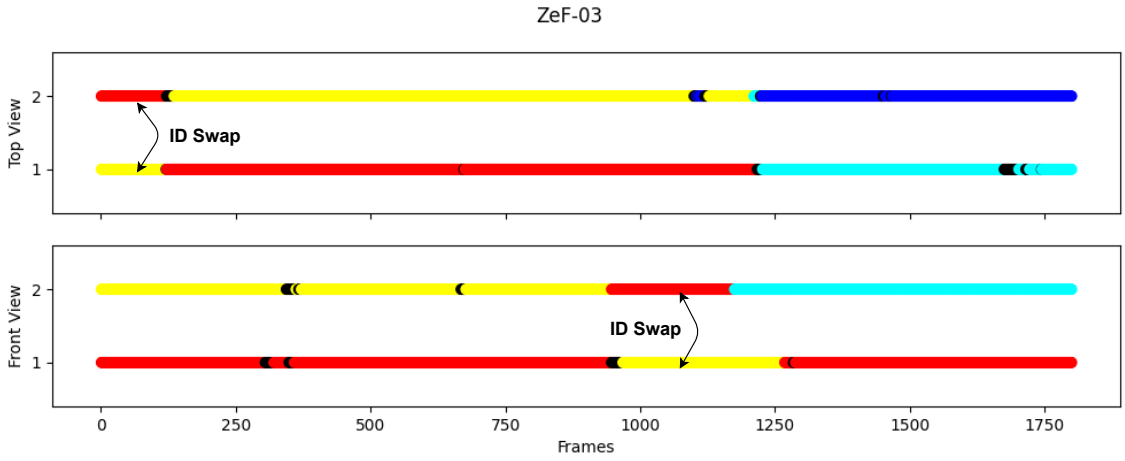


Figure 6.1: Illustration of the desired result, correcting ID swaps, maintaining consistent IDs for each ground truth ID.

In Fig. 6.1, we see ground truth object ID on the vertical axis and each color represents a unique ID. When two ground truth ID switches color, as marked on the figure, we consider this as identity swap, and it undesired. When a GT track changes color to a new color, means that the tracklet is killed and a new tracklet is initialized. It should be noted that for instance the red tracklet in the top view, is one long tracklet and we cannot directly from the data see that an ID swap occur.

6.1 Motivation

In Fig. 6.1, it can be seen that swapping the identity of the first 120 frames in the top view is highly desired. This is apparent when ground truth data is available but for an automated system, this is not known beforehand. Therefore a need for a system to automatically analyze a set of tracklets and detect ID swaps is needed. Often ID swaps occur during occlusion, caused by missing detections in that view and due to missing information, the identities could be swapped during tracking. This is hard to accommodate, as the occluded fish is close and it is hard to distinguish the individual fish. It is here, that the major advantages of tracking in various views come into play. If we lose track of a fish in one view, we might still have information about the particular fish in the other view.

The major problem when retrieving information about an unspecified fish detected in one view is to find the fish in the remaining view. As described in Section 5.2, triangulation could be used to construct a 3D position and then reproject the position back to 2D. From that, we can get an estimate of the match, by calculating the reprojection error. Doing so for all the detections in the top view and triangulating with all the detections in the front view, we could find a match by using the matches with the smallest reprojection error. This assumption could be used to analyze all the tracklets in each view and calculate the reprojection error. In Fig. 6.2, an illustration of how reprojection information can be found, where highlighted tracklet in top view is compared with the overlapping tracklets in front view one by one.

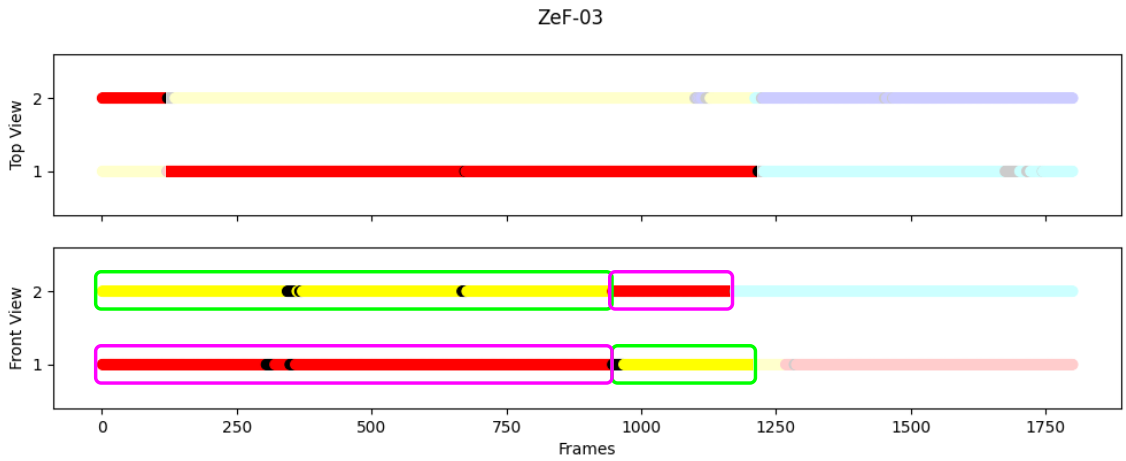
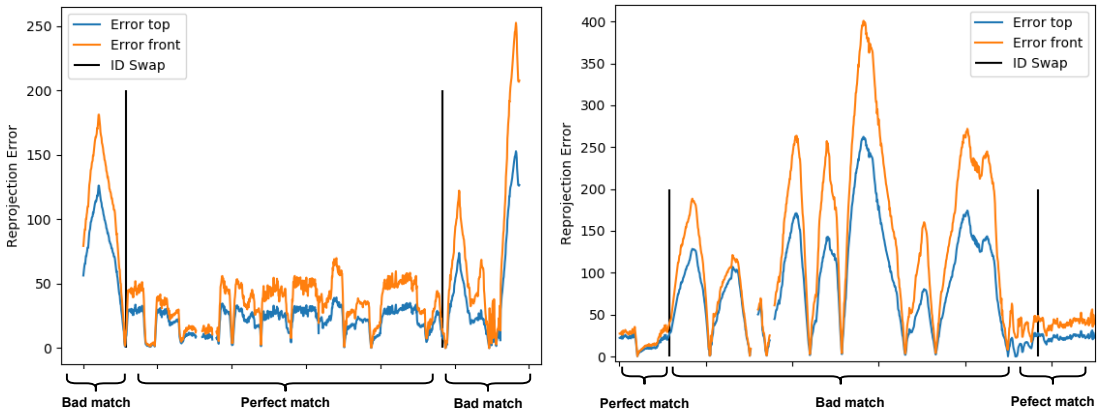


Figure 6.2: Illustration of how the reprojection errors are calculated, where overlapping tracklets will be compared. The red track in top view will be triangulated with the red and yellow track in front view, marked with purple and green respectively.

It should be noted that in a real case we do not have ground truth information, and therefore we don't know beforehand that an ID swap has occurred, as the red track is continuously, but belongs to two distinct fish. By extracting information from all the detections of the given tracklet, and triangulate with all overlapping tracklets in the other view, we can gain knowledge of how detections match. If the reconstruction error starts to rise, it may mean that the identity of two or more fish has been swapped. In Fig. 6.3 the reprojection error for the red and yellow tracklets in Fig. 6.2 is plotted to investigate any correlation between ID swaps and reprojection error.



(a) (b)

Figure 6.3: Reprojection error for the red track in top view triangulated with the red and yellow track in front view, from Fig. 6.1. Observed ID swaps are marked with vertical black line and reprojection errror is plotted for both views, with interpretation added below the curly brackets.

From Fig. 6.3a it is seen that the reprojection error is very large for the first part of the sequence, but suddenly the reprojection error decreases, and stay below a certain limit, and then quickly raise again. Similar for Fig. 6.3b, a complementary pattern is seen. Where we have large reconstruction error in Fig. 6.3a we have low error in Fig. 6.3b and vise versa. It can be seen that the IDs in the first couple of frames in the sequence is swapped, and followed by a swap in the last frames of the sequence, this can be seen due to the large reprojection error. Automating this process could eventually find and fix ID swaps in the tracklets without the need for ground truth data. It is assessed that no particular valuable information is achieved by handling reprojection error in the top and front view separately, as they seem to follow each other very well. Therefore, in future analysis, when referring to reprojection error it simply is the sum of the two errors $e = e_t + e_f$. It has been chosen to implement such a system in modules, where the first module takes tracklets from both views as input and outputs potential candidate frames where swaps could have happened, this is inserted into a module that attempts to correct these identity swaps. This system is shown in Fig. 6.4.

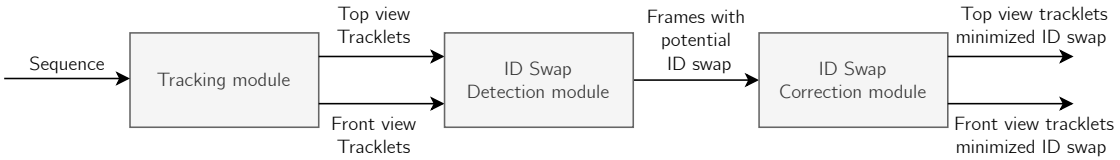


Figure 6.4: Module structure of the proposed method for detecting and correcting identity swaps.

In the following sections, each module will be described and designed with the sequence ZeF-03 as the sequence making the foundation for the analysis. When the method is designed and tested, new results will be generated with the corrected tracks.

6.2 Detection Module

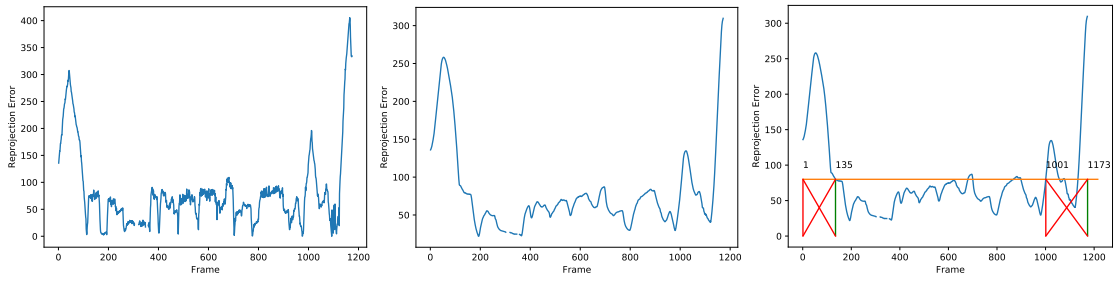
The first module in the system is the detection module. This is a novel idea, and no previous work dealing with multi-view ID swap detection was found. The system builds upon the aforementioned theory about reconstruction error.

The proposed algorithm for this module is built for a minimum working example with opportunities to improve further, this means that thresholding is used to determine if we have a good or bad match, a more sophisticated method might be suitable for the method, and such methods will be suggested in Section 6.5. The generated reprojection errors are smoothed using exponential smoothing. The errors are smoothed to reduce the impact of small high-frequency noise in the reprojection error caused by imperfect detections. These are tiny spikes around the original reconstruction error and are undesired. The smoothing is carried out by the following formula.

$$f(t) = \alpha \cdot x(t-1) + (1-\alpha) \cdot f(t-1) \tag{6.1}$$

Where $f(t)$ is the exponential smoothed version of the error at time t, α is a parameter determine the degree of smoothing, $x(t-1)$ is the reprojection error at time $t-1$ and $f(t-1)$ is the previous smoothed signal.

To accommodate for the smaller spikes, that eventually can touch the threshold, we have introduced a patience. This parameter determines a minimum number of consecutive frames above the threshold is needed before a swap can be detected. In Fig. 6.5 an example use of the module can be seen.



(a) (b) (c)

Figure 6.5: Example use of the module (a) Reprojection error for a track in top view and an overlapping track in front view is generated. (b) The reprojection error is smoothed using exponential smoothing. (c) Regions are found using thresholding, the threshold is marked by the horizontal orange line.

This module comes with two essential hyperparameters, the threshold and the patience, to tune the module a simple grid search has been conducted. We are measuring the F1 score, based on the ability to correctly detect ID swaps. To make this work, the ID swaps have been manually annotated. A detected swap within 15 frames is considered true positive, where a detection further away is considered false positive. To recap the F1 score is calculated as follows:

$$F_1 = \frac{\text{TP}}{\text{TP} + 0.5(\text{FP} + \text{FN})} \tag{6.2}$$

Where TP is true positive detection, FP is false positive and FN is false negative.

Conducting the grid search by running the module on the annotated sequence, with a predefined set of parameter values {60, 80, 100, 120, 140} for the threshold and {20, 30, 40, 50} for patience allow us to investigate the best performing set of parameters. The results from the grid search on the validation data can be found in Table 6.1. In the table to the left results from ZeF-03 is shown and to the right ZeF-04 is presented. The training set ZeF-01 and ZeF-02 is omitted from the test to avoid making decisions based on data that has been trained on.

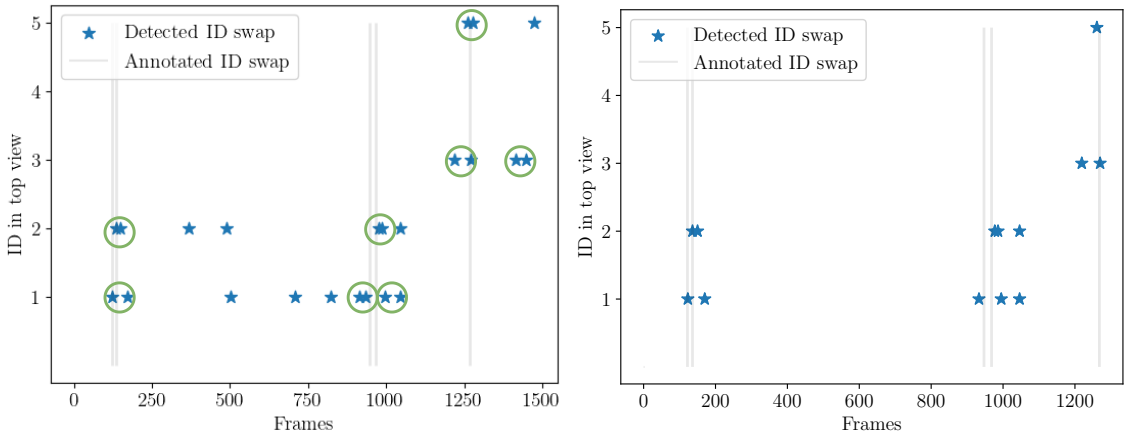
Patience \ Threshold	60	80	100	120	140
50	**0.44**	0.4	0.31	0.17	0.15
40	0.4	0.38	0.31	0.15	0.14
30	0.35	0.3	0.24	0.13	0.11
20	0.25	0.29	0.21	0.1	0.09

Patience \ Threshold	60	80	100	120	140
50	**0.36**	0.26	0.26	0.3	0.26
40	0.25	0.33	0.32	0.31	0.27
30	0.28	0.3	0.32	0.26	0.3
20	0.22	0.28	0.29	0.23	0.28

Table 6.1: F1 scores for the gridsearch on the validation sequences ZeF-03 (left) and ZeF-04 (right).

From the results in Table 6.1 it can be seen that a threshold of 60 and patience of 50 seems to be the best parameter set among the specified parameters, with an achieved F1 score of 0.44 and 0.36. These scores are not particularly high, and this is caused by a large amount of false positives detections probably due to fluctuations in the reconstruction error.

An interesting discovery when visualizing the predictions, together with the annotated swaps, it was found that most often true positives come in pairs, close to each other, and always with the same ID in, for instance top view. This gave the inspiration to build a filtering function in the module, that searches for pairs in detections. This is visualized in Fig. 6.6.



(a) (b)

Figure 6.6: Visualization of the detected swaps, where a pattern is found. Pairs are marked with green circles. (a) is default setting visualized with the parameters found during grid search and (b) is filtered using the pair method. Five IDs are present due to the amount of unique tracked IDs in top view.

Sorting the detections in this way, we were able to eliminate most of the false positives detections in the sequence. Intuitively it makes sense that true ID swaps come in pairs if

following a tracklet in top view, compared with two overlapping tracklets in front view. An ID swap must be visible in both front view tracklets, if they are switched, this is also apparent in Fig. 6.3a and Fig. 6.3b, where detections for the same frame would be present in both figures. Not to be confused with two consecutive tracklets where the ID is changed but not swapped. The module is build with smaller subsystems, described above, and a summary can be found in Fig. 6.7.

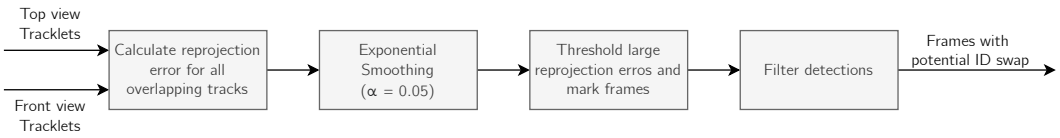


Figure 6.7: Structure of the swap detection module used in this work, providing detections for a module to correct the identity swaps.

Now the detection module is implemented, providing frames and regions of potential identity swaps in the given sequence. We can move on to designing a system for handling these detections and eventually improve performance in each view.

6.3 Correction Module

When two or more tracklets are swapping IDs, the triangulation fails for the frames that hold swapped IDs, it is highly desired to swap the affected frames back to their original identity. Such a module aims to minimize the number of ID swaps together with minimizing the number of swapped frames. To measure these parameters, a test bench has been developed and are described in detail in Appendix A.

This module receives the output from the detection module, which consists of the frame number, the id for the tracklet in top view and the id for the tracklet in front view. First, the module needs to group these detections such that each group consist of at least two detections. These groups are clustered by their distance, such that all closely related detection are formed in a group. Doing so, ensure that we have groups of at least two frame numbers, two ids in the top view and two ids in the front view.

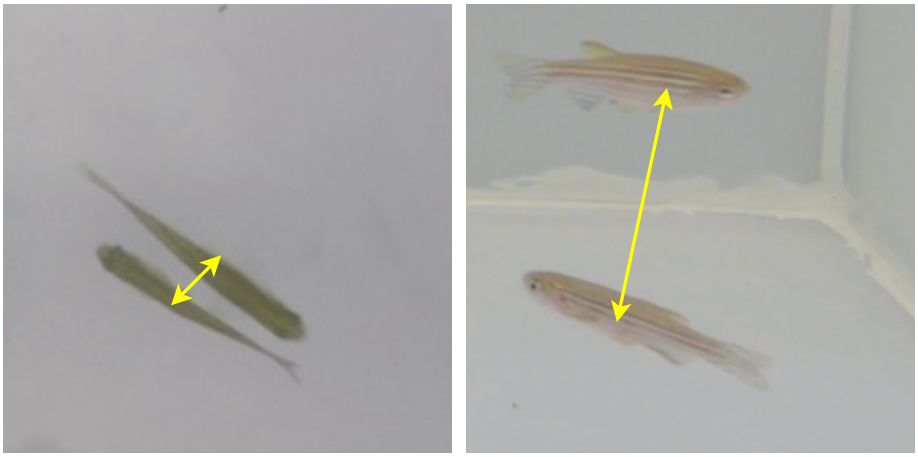
From these data, regions can be constructed wherein reconstruction errors will be calculated. For N ids in top view and corresponding N ids in front view, we can construct a $N \times N$ cost matrix, consisting of the mean reprojection error for the ID pair, then global optimization can be employed to find the minimum cost assignment of top and front view pairs.

Using this method for correcting ID swaps, we are facing an ambiguity problem. If an ID swap is detected in frame x we cannot directly tell from the data if the ID is swapped in the top or front view. From the reconstruction error, it is not apparent which view is causing the high error. Instead, it has been chosen to measure the average distance between the fish in both views for the entire overlap using Eq. (6.3).

$$d(r,v) = \frac{2}{F \cdot N \cdot (N-1)} \sum_{f=1}^{F} \sum_{i=1}^{N} \sum_{j=i+1}^{N} \|pos(i,v) - pos(j,v)\|_2 \tag{6.3}$$

where $d(r, v)$ is the average squared distance for the region r in view $v \in \{T, F\}$. F is the total number of frames in the region and N is the total number of fish in the frame. $pos(i, v)$ is the 2D position of fish i in view v and similar for fish j in $pos(j, v)$.

This approach is illustrated in Fig. 6.8 where $N = 2$.



(a) Top view (Subject to ID swap) (b) Front view (Not subject to ID swap)

Figure 6.8: Illustration of how the mean squared distance between the fish is calculated, visualizing the assumption that the distance is small in the view where the identity swap occur. This frame is frame 121 in ZeF-03 where the first identity swap in top view is seen.

Together with the average distance, the standard deviation can be calculated, and based on these data the most occluded view can be estimated. It is assessed that close to an ID swap, the fish is clustered together, and thereby the squared distance between the fish must decrease rapidly. This would be visible in both the standard deviation as well as the mean. We choose the view with the lowest mean value and largest standard deviation.

When a region with swapped identities is determined together with a set of matching id pairs, which minimizes the reprojection error, we can swap the identity of the tracklet inside that region. This module is built with smaller subsystems and an overview of the system can be seen in Fig. 6.9.

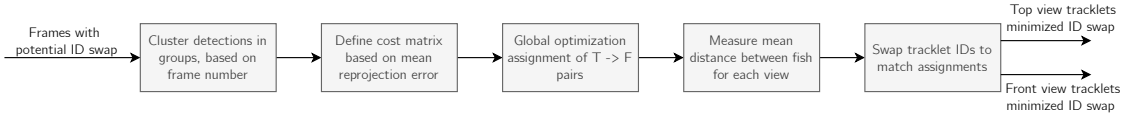


Figure 6.9: Structure of the module for correcting ID swaps, taking inputs from the detection module, aiming at minimizing ID swaps.

6.4 Evaluation of Detection and Correction modules

In this section, the modules will be evaluated for their performance individually. However, the correction module is affected by the performance of the detection module which generates regions to account for.

Detection performance is often measured with true positives, false positives and false negatives, and these are summarized in a F1 score putting more weight on the true positives. In Table 6.2, the results from the test can be seen.

	TP	FP	FN	F1
ZeF-01	18	28	114	0.202
ZeF-02	10	12	13	0.444
ZeF-03	5	6	0	0.625
ZeF-04	5	18	5	0.303

Table 6.2: Detection results for the detection module, based on annotated ID swaps with a positive margin of 15 frames.

As seen in ZeF-01 we detect 18 out of 132 ID swaps, this is a low amount of detections, in such a long sequence of 7 128 frames. Not surprisingly the best performance is found in ZeF-03 which is used as a foundation for building such a module, in this case, every swap is found, but with 6 false detection's as well. In Fig. 6.10, the detections are visualized to provide overview of how the detections are located.

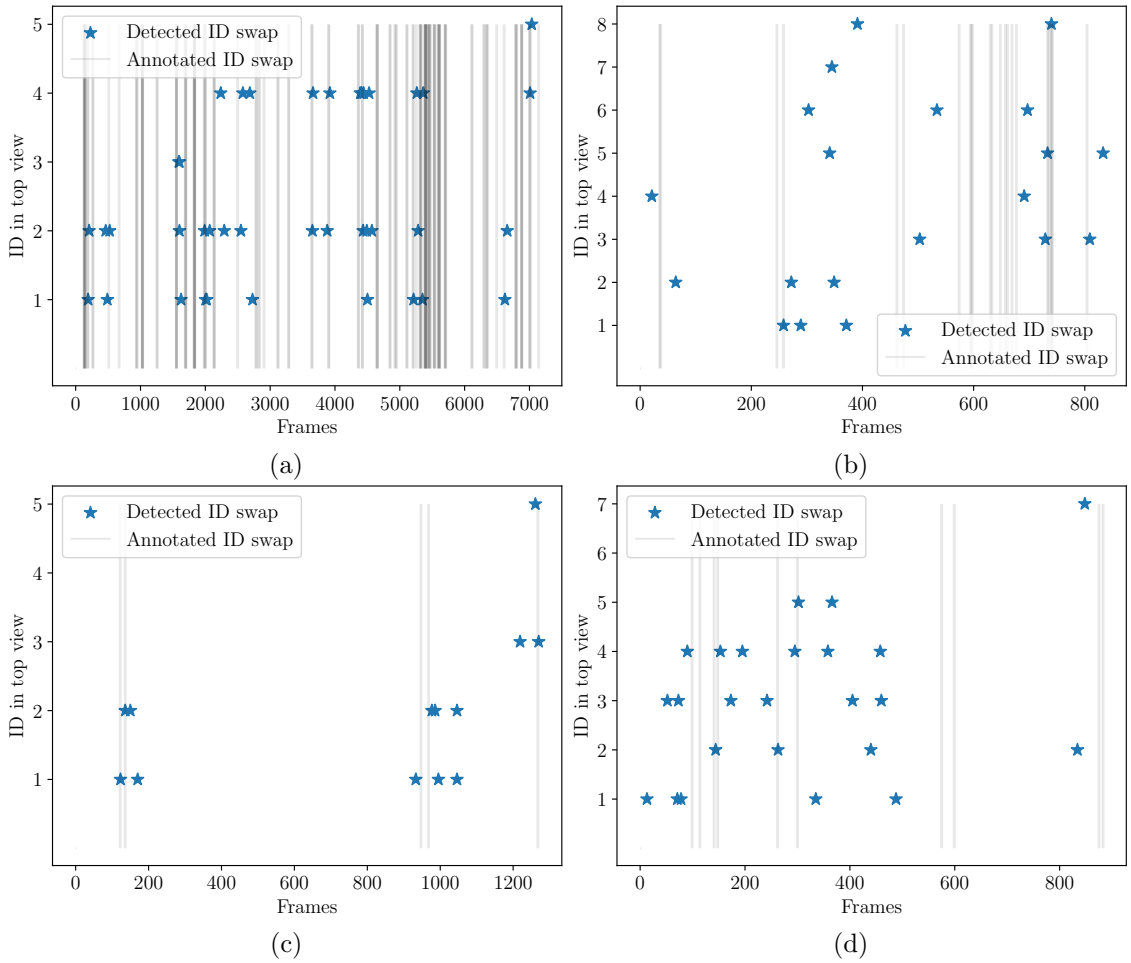


Figure 6.10: Visualization of the detected ID swaps, it is seen that in ZeF-01 and ZeF-02 many ID swaps are present, and a fraction of them are detected.

The correction module is evaluated upon two metrics, namely the number of ID swaps occurring and the amount of swapped frames in the sequence. These metrics are obtained from the test bench developed for the case. More information about the test bench can be found in Appendix A. The aim is to minimize the number of ID swaps and reduce the amount of swapped frames to optimize the performance of the 3D projection system. In Table 6.3 the results from the test can be seen.

	Before		After	
	Swaps	Swapped Frames	Swaps	Swapped Frames
ZeF-01	133	9 659	177	8595
ZeF-02	23	1 563	28	1946
ZeF-03	6	776	6	540
ZeF-04	16	704	38	916

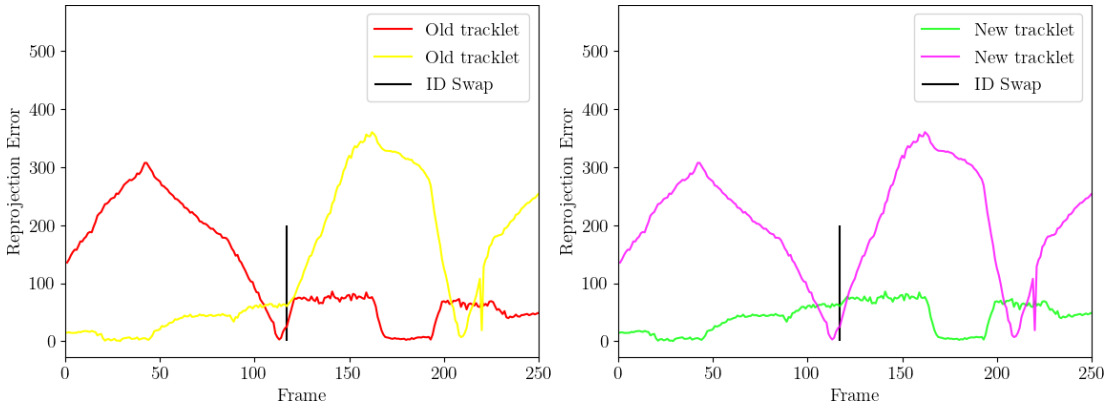
Table 6.3: Test results of the correction module for each sequence in the training data.

From the test results in Table 6.3, it can be seen that the correction module does not perform very well. In all cases, it increases the number of swaps in the given sequence and half of the time it increases the amount of swapped frames. This is not a desired behavior of the module. In Appendix B the sequences can be seen before and after the correction module to visualize the impact of the module.
In the following section errors and observations will be discussed, followed by suggestions for changes in future work eventually making the modules more robust.

6.5 Discussion and Future Work

In this section, observations and considerations made during the design and evaluation modules will be discussed.

Concerning the detection module, the decision of whether or not an ID swap has occurred or not has to be taken. Currently, thresholding the reprojection error together with a patience parameter is used to determine if the reprojection error is large enough to claim a swap at that particular frame. This is a simple and non-adaptive method that is assessed to work sufficiently, but a more sophisticated method could be interesting to investigate further. For instance, an interesting pattern is observed when plotting the reconstruction error for a single top view tracklet against all overlapping front view tracklets. When the ID swap happens, it seems like the reprojection error is drastically switching and two or more consistent lines could be found. This is very complex interpretation, and is illustrated in Fig. 6.11. The reprojection error is plotted for the red top view tracklet in Fig. 6.2, together with the red and yellow tracklet in the front view. The reprojection error is marked with the respective front view color ID. Attention should be paid to the purple and green line, as this seems to be the true original tracklet, if the ID swap had not happened.



(a) Top view (Subject to ID swap) (b)

Figure 6.11: Illustration of the hypothesis that two swapped tracklets seems to swap pattern in reprojection error. It should be noted that the green and purple line is not tracked but are used to illustrate the interpretation.

Close to the ID swap, the reprojection error quickly changes direction and inherit the pattern of the other tracklet. This information might be useful for developing a more robust ID swap detector. If a system to detect these sudden changes could be developed the performance may be improved even further. A large rate of change of the reprojection error may be an indicator for such a switch, but no unambiguous methods were found during the work.

In regards to the correction module, ambiguities are discovered, such as the view of interest when swapping identities. Deciding which view to swap is a difficult problem. Currently, the mean and standard deviation of the distance between the fish is used to choose the view with the lowest average distance or highest variation. This might not be the best solution for determining the view, and more work should be invested in analyzing for any pattern which makes the relevant view more apparent. For instance, calculating the view with the most missing detections, under the assumption that missing detections often leads to ID swaps.

Another interesting problem that needs more attention, is which direction makes the most sense swapping, before or after the detection frame where ID swap occurs. This depend on the tracklet and eventually other swaps later in the sequence. To visualize this problem Fig. 6.12 has been drawn.

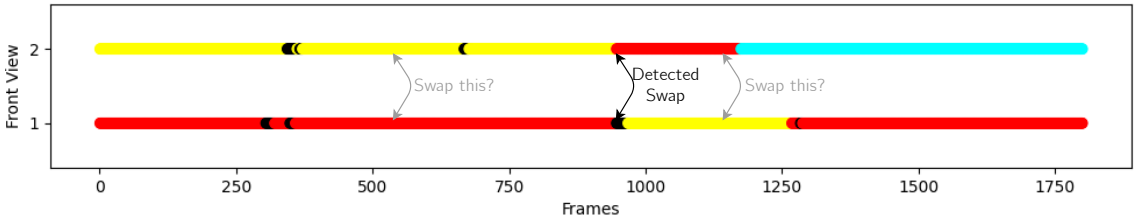


Figure 6.12: Visualization of the direction problem, where both before and after the detected ID makes up a suitable solution of swapping the tracklet.

Swapping the region before the detected ID swap would solve the ID swap for the first part of the sequence, by swapping the yellow tracklet with the red tracklet. But a more optimal solution would be to swap the red tracklet after the detected ID swap, as this

completes the tracklet entirely. Currently, a solution to this problem could not be found, and only selecting the region before the detection is implemented.

Evaluating the sequences is an important aspect of the pipeline, for instance, the measurement of ID swaps in the given sequence. In this work, a test bench for evaluating the performance is developed in a controversial manner, by counting every swap as an ID swap. Illustrated in Fig. 6.13.

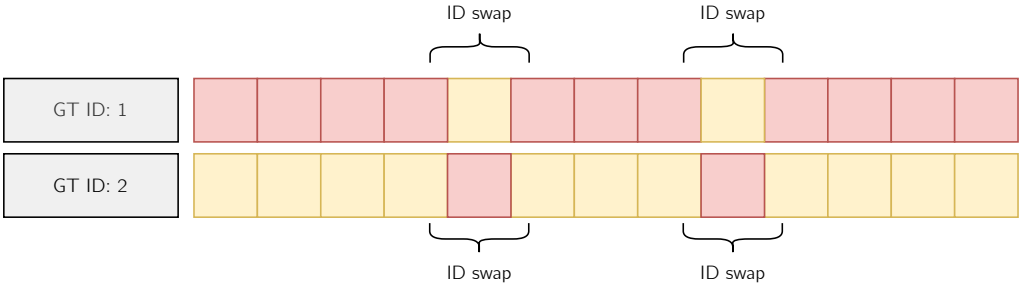


Figure 6.13: Illustration of the method for count ID swaps, which has been used for evaluating the detection and correction module.

A swap of a single frame is indeed undesired but after all not that harmful, and cannot be detected due to the patience of the developed detection module. Furthermore, when correcting for ID swaps, it is not ensured that we correct all the swapped frames, but only a subset of the frames, hence leaving a minor swap. This swap still contributes to the score but the outcome of the module would still be improved. It could be considered to implement a more soft test bench that can handle very short ID swaps differently. Counting every single-frame swaps might skew the performance report. Another aspect could be to quarantine x number of frames close to the detected ID swap, as we cannot confidently determine the identity of these detections. In Appendix B the minor swaps are visualized, putting these in quarantine and deciding their ID later when most of the tracklet is completed might be a solution.

In the next section, the corrected 2D tracklets will be triangulated again, and new 3D tracklets will be formed to test upon the benchmark test set to investigate for any improvements.

6.6 Evaluation on 3D-ZeF20 Testset

In this section the new results will be presented together with the current best performing method on the benchmarks test set. In Table 6.4 the results for every test sequence can be found, the results on the top is the benchmark score marked with an asterisk (*) and the score below is the new results.

	MOTA	IDF1	MOTP	MT	ML	FP	FN	Rcll	Prcn	FAF	ID Sw.	Frag	MTBFm
ZeF-05*	79.4	88.9	34.6	1	0	28	157	82.6	96.4	0.0	0	30	12.2
ZeF-05	79.11	89.6	33.25	1	0	94	94	89.56	89.56	0.1	0	20	19.7
ZeF-06*	78.4	88.2	36.6	1	0	45	344	80.9	97.0	0.1	0	44	16.2
ZeF-06	32.72	43.9	32.81	0	0	582	623	65.39	66.91	0.6	6	33	16.8
ZeF-07*	40.3	54.9	32.8	0	0	577	2,104	53.2	80.6	0.6	6	200	5.9
ZeF-07	8.20	42.6	31.42	0	0	1971	2145	52.33	54.44	2.2	15	134	8.4
ZeF-08*	48.0	58.6	34.6	0	0	720	3,947	56.1	87.5	0.8	12	246	9.8
ZeF-08	-26.99	31.6	31.83	0	3	5642	5779	35.79	36.34	6.3	8	184	8.2

Table 6.4: Test results obtained from MOT Challenge. Results marked with asterisk (*) is the current top performance on 3D-ZeF20 benchmark testset, taken from MOTChallange.

Despite the fact that the detection and correction module is not working as perfect as expected, the results from the ZeF-05 is acceptable and compared to the results before correction we see minor improvements in performance for some of the sequences where, for instance, ZeF-08 has a major drawback of using the correction module. It seems like the correction module is performing better with less number of fish.

It can be concluded that the benchmark score is currently not beaten, and that the correction module does not work as intended, especially not for sequences with a large number of fish present.

Chapter 7

Discussion

In this chapter, observations and consideration made throughout the work will be discussed and provides the reader with information and topics for future improvements. This chapter will be separated into two sections, one regarding the work in two dimensions and a section about working in three dimensions. In the first section, thoughts about Tracktor and the entire tracking pipeline will be discussed and in the next section topics related to triangulation and 3D projection will be mentioned.

7.1 Tracking in 2D

In this work, the main focus has been to improve 2D tracking performance, under the assumption that better 2D tracklets would yield better performance in 3D. Different from the benchmark, we have employed Tracktor for tracking zebrafish. Various optimization of the models has been tested and sub-optimal models have been trained. As mentioned throughout the evaluation of the Tracktor framework, the provided motion model did not show its potential due to the erratic movement of zebrafish. Current state-of-the-art methods within the field of zebrafish tracking have shown that Kalman filters could be used to estimate the position of zebrafish when detections are unavailable. This has yet not been tested in this work, but in future work, it could be beneficial to implement. Getting information about the zebrafish's location during occlusions through estimates might help the framework to keep track and consistent identities, as identities often switch when we miss one or more detections.

Concerning the re-identification network, TriNet used to re-identify lost tracklets within a given period. It was chosen to run with the model unconstrained because we assumed no new fish will appear in the aquarium. Increasing the similarity threshold forced the network to assign a newly detected object to an existing tracker that has been lost. This was under the assumption, that if we have a lost track and detected a new fish then these must match. This might be an incorrect assumption, or at least handled badly. Because a large number of ID swaps was found, near occlusions, meaning that when two objects occlude each other we must assign the detections to one of the trackers. This assignment could be correct, but it is prone to go wrong too. The unconstrained model showed good performance on the MOT metrics, but visualizing the tracks it was clear that ID swaps occurred quite often. It might be beneficial to make the model more

conservative, and instead of re-identify lost objects, it would be a more optimal solution to kill the tracker and initialize a new reducing the risk of ID swaps.

Another interesting approach to consider, is tracking the sequence backwards, or even in both directions. This could eventually help by providing information about the individual fish from both directions and could possibly increase performance near occlusions. In Fig. 7.1, this concept is illustrated.

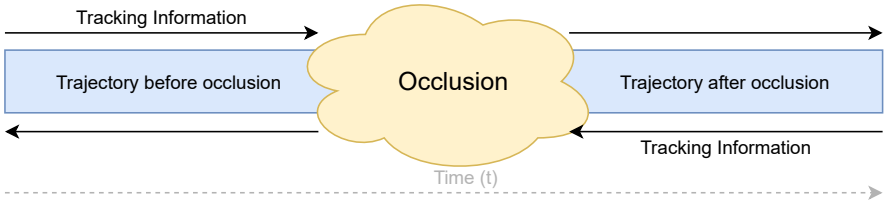


Figure 7.1: Illustration of how information is accessible when tracking in both directions.

This could eventually help to solve problems where sparse information about the fish is accessible, for instance, in the first few frames of the sequence. If errors occur in the first couple of frames, we have not initialized a good visual representation (sufficient embeddings) of the individual fish to re-identify them correctly. For instance, the first frames of ZeF-04, containing five fish, is very crowded in the top left corner in the front view. Through the visualization of the tracklets, it is seen that lots of errors occur in the starting frames, but after a couple of frames the cluster disappear and the fish is spread out. After that, the tracklets are more consistent and few errors are present. Tracking backwards could eventually help because then a proper visual representation of the individual fish is established already. Making the network able to identify the fish correctly. In Fig. 7.2, the aforementioned cluster can be seen.

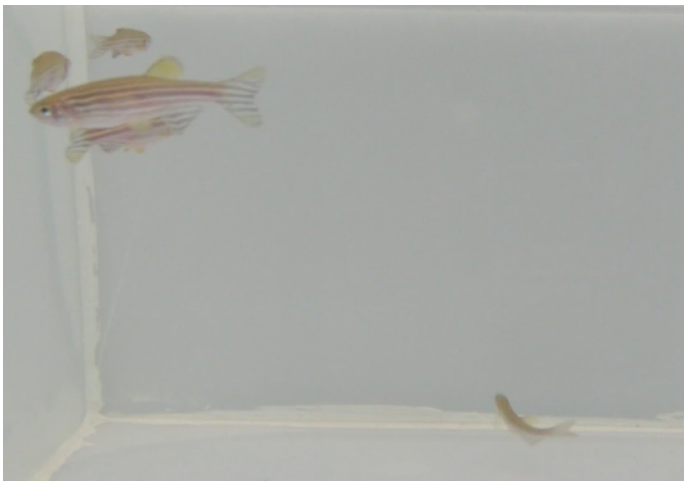


Figure 7.2: Frame number 1 from the sequence ZeF-04, in the top left corner of the tank a cluster of four fish is seen, making it hard to learn a good visual representation of the individual fish.

Overall, Tracktor showed good performance on the validation set. There is a possibility that the entire framework (Object detection and re-identification) is somewhat overfitted

to the training data. The test set is mutual exclusive from the training data, by using new zebrafish, which is a bit smaller and looking slightly different. If the model is overfitted to the large zebrafish in the training data, we would see a decrease in the performance on the 2D tracklets. Unfortunately, ground truth 2D information of the test set is not available and we cannot evaluate the 2D tracklets separately without projecting them into 3D. However, considering regularization techniques to improve generalization might be of interest.
Tracktor was able to provide great results in 2D but when entering 3D space, the performance decreased drastically. In the following section, the work in 3D will be discussed.

7.2 Tracking in 3D

Comparing the performance on validation sequences in the front view, Tracktor was able to provide better tracklets than any method in [30]. In the top view, no big changes compared to the benchmark performance was found. It is assessed that the front view was the largest source of errors in the benchmark results as it was the view with the lowest tracking performance. Tracktor was able to solve this issue, but moving to 3D failed tremendously.

No good explanation for the unsatisfying results was found, besides the number of ID swaps and the number of frames with swapped IDs. The pipeline used in previous work for estimating head positions and triangulation was employed to ensure consistency in the 3D reconstruction part. In this way, we were able to ensure any improvement in the performance directly corresponds to improved 2D performance. However, this was not the case.

The modules for detecting and correcting potential ID swaps did not improve the performance of the 3D trajectories. The module for correcting ID swaps, by swapping the ID of a tracklet with the tracklet with the lowest reprojection error did not perform well and more work is needed to get that to work. However, the detection module did show potential for detecting ID swaps in the tracklets by examining the reprojection error. This is a novel approach and more work within this field is of interest. From the visualization of the reprojection error, it was able to see a pattern between good and bad matches, and a more intelligent methods for deciding is needed. The current method utilizes thresholding and it might not be optimal. In Section 6.5 this is elaborated in more detail, together with suggestions for future approaches. It is difficult to improve the performance of a correction module before we can consider the detection module stable and robust. Overall, investigating the relationship of tracklets in-between views using the reprojection error shows great potential and should be considered in future work.

Chapter 8

Conclusion

Automatic tracking of multiple zebrafish for behavior analyzing studies is not a trivial task. Current solutions for tracking zebrafish is either solely in 2D or limited to a single fish in 3D. Furthermore, the setups required are often highly specialized and difficult to generalize. Benchmark dataset in 3D with multiple fish are publicly available and can be used to test different method and algorithms in the strive of a fully automated 3D tracking system for multiple fish. The purpose of this thesis is twofold. The first purpose was to investigate if current state-of-the-art methods within pedestrian tracking could be used together with domain knowledge to improve existing performance on the benchmark dataset 3D-ZeF20. Secondly, the purpose of using a dataset with multiple views was to investigate if information between views could be used to improve the performance of the two views, for instance, during occlusions.

2D Tracking with Tracktor

Relating to the first part of the thesis, concerning tracking in 2D, problems was stated:

How can multiple zebrafish be tracked in two distinct camera views?

How can SOTA in MOT be adapted for tracking of zebrafish?

It can be concluded that tracking multiple zebrafish in two views can be carried out by using state-of-the-art MOT frameworks such as Tracktor. Tracktor tackles the tracking in a different way than traditional tracking approaches, by exploiting the associated object detection model to detect and track each object. Substantial performance improvements are found for the front view, wherein top view a minor drawback was found. The impressive results in the front view might be due to the associated re-identification model, as the IDF1 score is increased greatly. Re-identification benefits from the textural information in the front view and does not show its potential in the top view. In Table 8.1 a comparison to the benchmark data is seen for the top view, and in Table 8.2 results for the front view is presented.

	MOTA	MOTP	IDF1	Prc	Rcll	FP	FN	MT	ID_{swap}	Frag.	MTBFm
$FRCNN_{ZeF\text{-}03}$	**96.8**	0.978	**76.3**	**0.992**	**0.979**	**30**	**76**	2	10	17	92.737
$Our_{ZeF\text{-}03}$	95.22	**0.131**	59.7%	0.978	0.976	79	86	2	**7**	**14**	**103.4**
$FRCNN_{ZeF\text{-}04}$	**96.4**	0.972	48.8	**0.995**	**0.972**	**22**	**128**	5	15	**23**	**83.528**
$Our_{ZeF\text{-}04}$	88.66	**0.178**	**87.34%**	0.95	0.936	222	293	5	**1**	40	48.0

Table 8.1: Comparison of 2D tracking performance on validation data top view. FRCNN is best score from benchmark [30].

	MOTA	MOTP	IDF1	Prc	Rcll	FP	FN	MT	ID_{swap}	Frag.	MTBFm
$FRCNN_{ZeF\text{-}03}$	93.6	0.958	13.1	**1**	0.974	**0**	95	2	136	24	21.372
$Our_{ZeF\text{-}03}$	**96.58%**	**0.114**	**66.32%**	0.989	**0.977**	36	**83**	2	**4**	**8**	**159.9**
$FRCNN_{ZeF\text{-}04}$	79.0	0.957	17.8	0.996	0.818	**14**	826	3	117	72	17.402
$Our_{ZeF\text{-}04}$	**88.18%**	**0.146**	**74.14%**	**0.98**	**0.901**	81	**446**	**5**	**11**	**38**	**53.3**

Table 8.2: Comparison of 2D tracking performance on validation data front view. FRCNN is best score from benchmark [30].

In all cases, the number of ID swaps has been decreased significantly and most of the time the number of fragmentations has been decreased greatly. From above it is concluded that Tracktor outperformed the current performance on the benchmark in front view, but not in top view. Tracktor is handling each view separately, and can be used for the front view in further studies and other methods could be employed for top view.

3D Projection and Reprojection

Regarding the last part of the thesis, concerning tracking in 3D and multi-view information, problems was stated:

> ***How can information between views be shared to improve tracking performance during occlusions?***
>
> ***How can 3D trajectories be constructed from two views?***
>
> ***Is it possible to beat the current best performance on the 3D-ZeF20 benchmark?***

Regarding the first question, if information between the top and front camera can be shared to improve tracking performance. It was found that the current implementation with a swap detection module combined with a correction module for correcting swapped tracklets, did not improve the performance. Ambiguities in the correction module were found making it unreliable for now. However, great potential was found in using reconstruction error as a measure for the fit for detections in each view. A low reprojection error indicates a good match between detections. Using this measure we were able to point out ID swaps in the tracklets. However, more work is needed to build a robust module for automatize the detection process but the fundamental theory is presented and a minimal working prototype is implemented scoring an average F1 score of 0.4 on the training and

validation data. There is room for improvements, but suggested improvements are highlighted in Section 6.5.
To compare the proposed method with the benchmark best performance, 3D tracklets needs to be formed. In this work, the framework developed by the author of the benchmark, are used to triangulate the tracklets for each view to estimate the 3D position, this is working very well. Errors occur when identities are swapped between tracks and causing issues in regards to performance. In 8.3 the final result on the benchmark test set is shown.

	MOTA	IDF1	MOTP	ID Sw.	Frag	MTBFm
Benchmark	**51.0**	**63.0**	34.4	18	520	**9.0**
Tracktor to 3D	12.36	42.8	**34.12**	**12**	**343**	7.59

Table 8.3: Test results obtained from MOT Challenge. Benchmark is the current best performance on 3D-ZeF20 benchmark testset, taken from MOTChallange.

The results show that the current best performance on the benchmark could not be outperformed at this time. A reduction in number of ID swaps and fragmentations are found, but MOTA and IDF1 is unfortunately lower than benchmark.

After all, it can be concluded that Tracktor is outperforming the benchmark in front view, but when entering the three-dimensional space, tracking fails due to ID swaps in the 2D tracklets. More work should be invested in reducing ID swaps before 3D triangulation.

Appendix A

ID Swap Test Bench

In this appendix a brief description of the developed test bench for providing test results for the ID swap detection and correction modules. The aim is to minimize the number of ID swaps in the sequence together with reducing the number of swapped frames. Therefore a test bench to measure these parameters is important.

Measuring Amount of Swaps

One of the key metrics to measure is the number of swaps in the sequence. When talking about swap amounts, we are measuring the ID associated with a ground truth fish. Every time a new id is found for the annotated fish, it checks whether or not this id has been associated with another fish. If the ID had already been used, it means that a new tracker is not created and an existing id has been swapped. This is illustrated in Fig. A.1

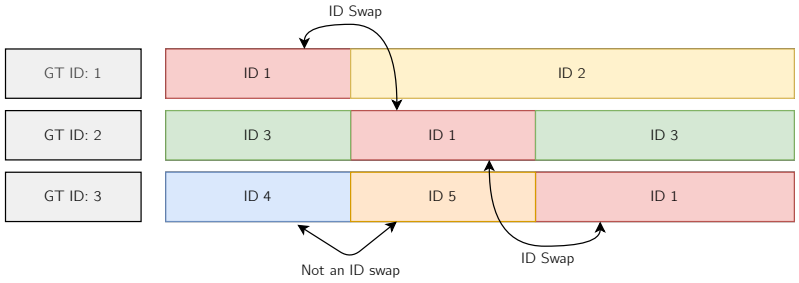


Figure A.1: Illustration of how ID swaps are measured. Each color is a distinct tracked ID.

Even a single detection with a wrong ID results in an ID swap, and switching back again counts as a new swap, this is a rather conservative method of counting swaps, but will be used in this project. The number of ID swaps is not the only important measure, but the length of the swaps, or more precisely how many frames has been swapped is important too.

Measuring Number of Swapped Frames

Before measuring how many frames that have a swapped identity, it is crucial to understand how to count these frames. It has been chosen to first assign an identity to the

ground truth ID with the largest part of the tracklet for every tracked id in the tracking. Then counting all the frames with an ID outside of the set of IDs assigned to the ground truth ID will give a measure of how many frames that is not correct. This is illustrated in Fig. A.2.

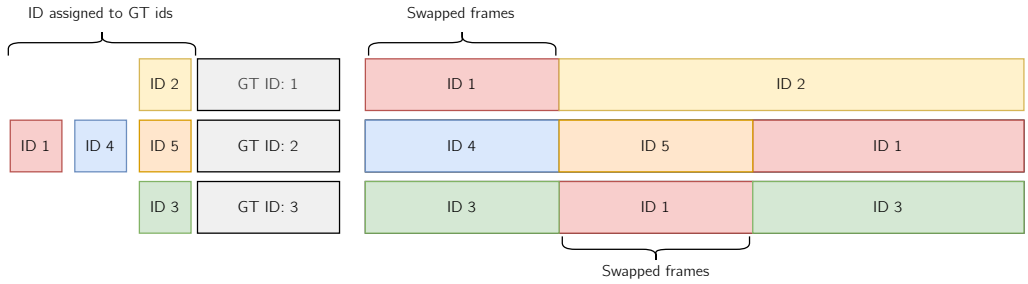


Figure A.2: Illustration of how number of swapped frames are measured. Each color is a distinct tracked ID. One ground truth ID is assigned to every tracked ID, based on the GT ID with the largest part of the tracklet.

Implementation test

To verify the implementation of the test module, an implementation test is performed to ensure the module is working as expected. Such that mysterious results can be related to the data and not the implementation of the test bench.

To test if the test bench behaves as intended, we need to insert data for which we know the expected outcome. As the aim is to minimize swaps and the amount of swapped frames, inserting the ground truth annotated data would be an opportunity. From the ground truth data, it is expected that the outcome is 0 swaps and 0 swapped frames. The test results can be seen in Table A.1.

	Detections		Ground Truth	
	Swaps	Swapped Frames	Swaps	Swapped Frames
ZeF-01	133	9 659	0	0
ZeF-02	23	1 563	0	0
ZeF-03	6	776	0	0
ZeF-04	16	704	0	0

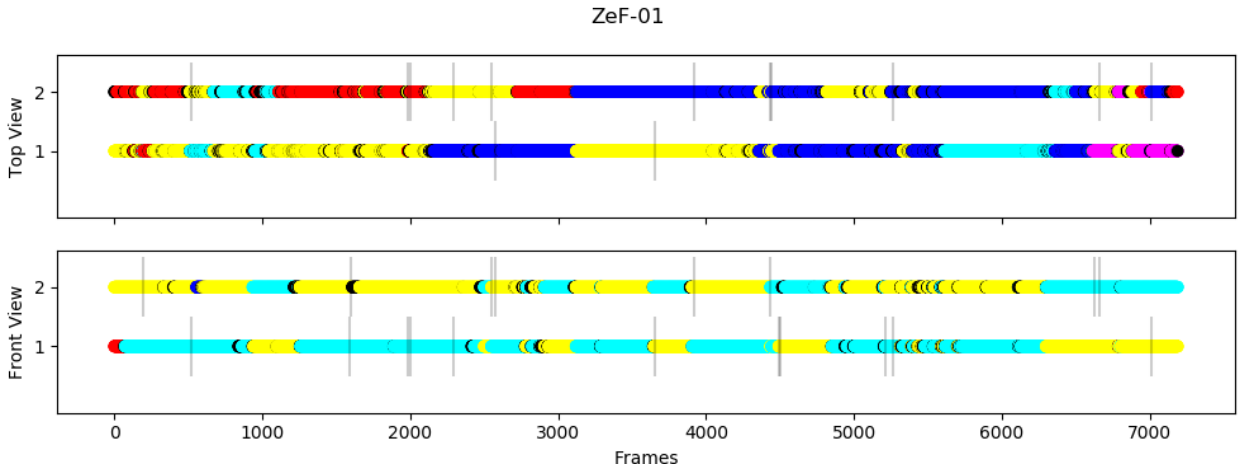
Table A.1: Implementation test to verify the test bench is working as expected. The results from the detections is manually checked to verify.

From the results in Table A.1, it can be concluded that the test bench is implemented correctly and is working as intended.

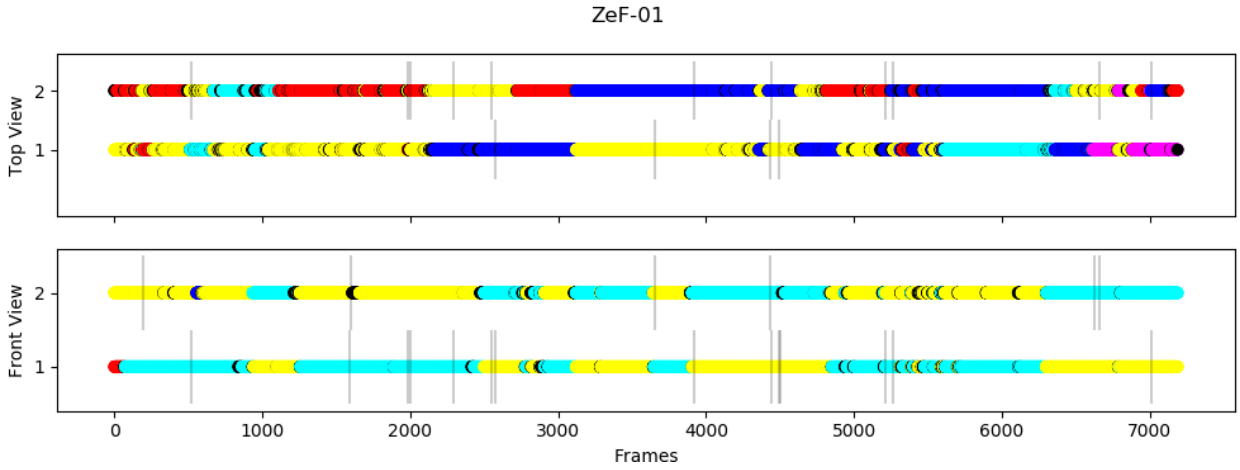
Appendix B

ID Correction results

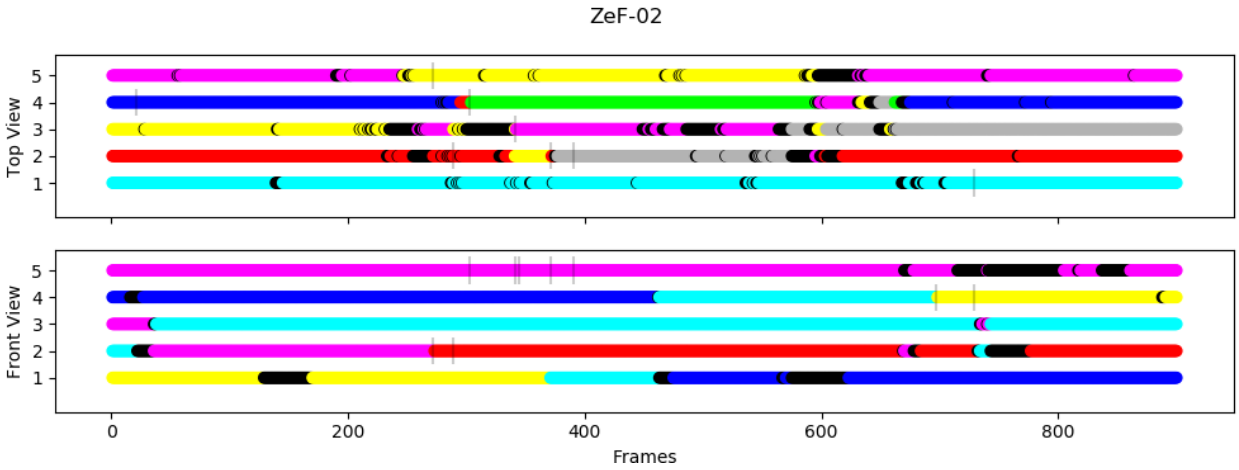
This appendix is intended to visualize the results from the correction module to visually compare the results by showing the tracklets before and after correction. Detected swaps are marked with vertical gray lines near the respective ID.



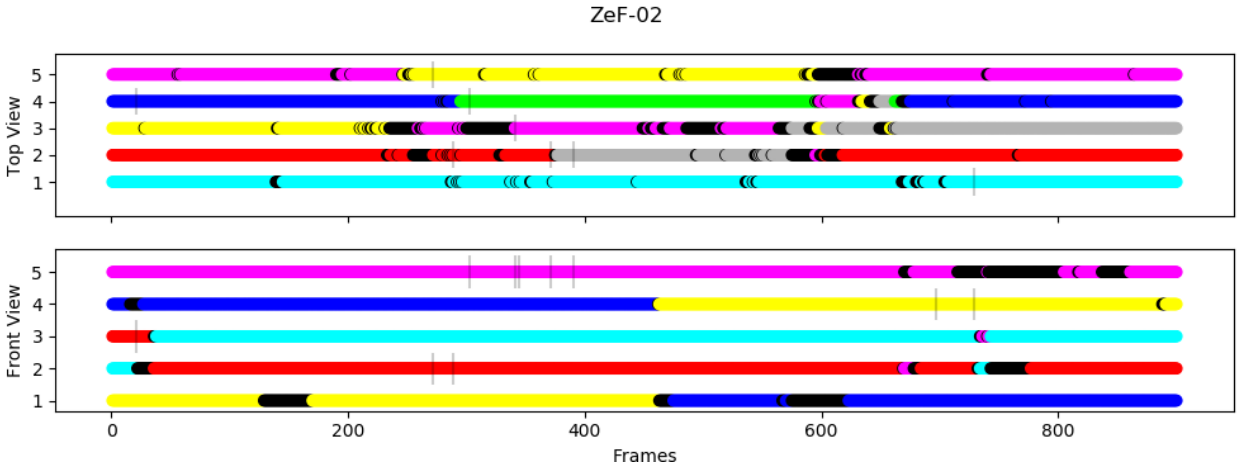
(a) After swap correction



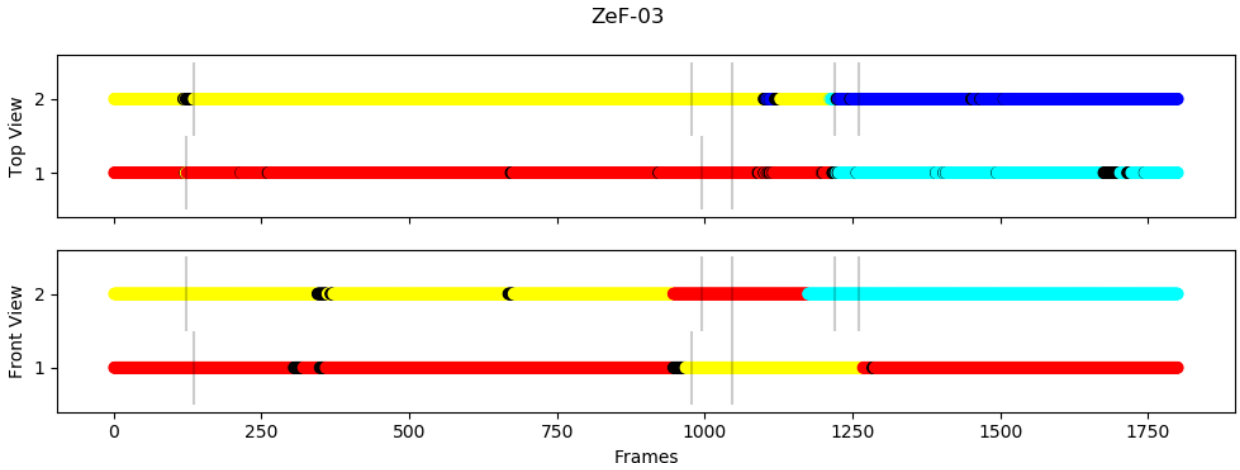
(b) Before swap correction



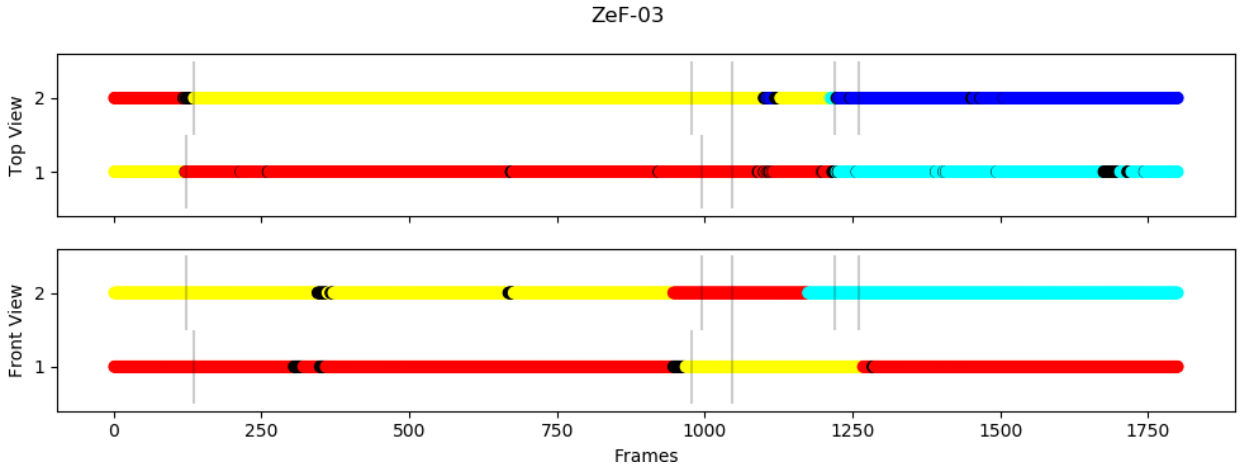
(a) After swap correction



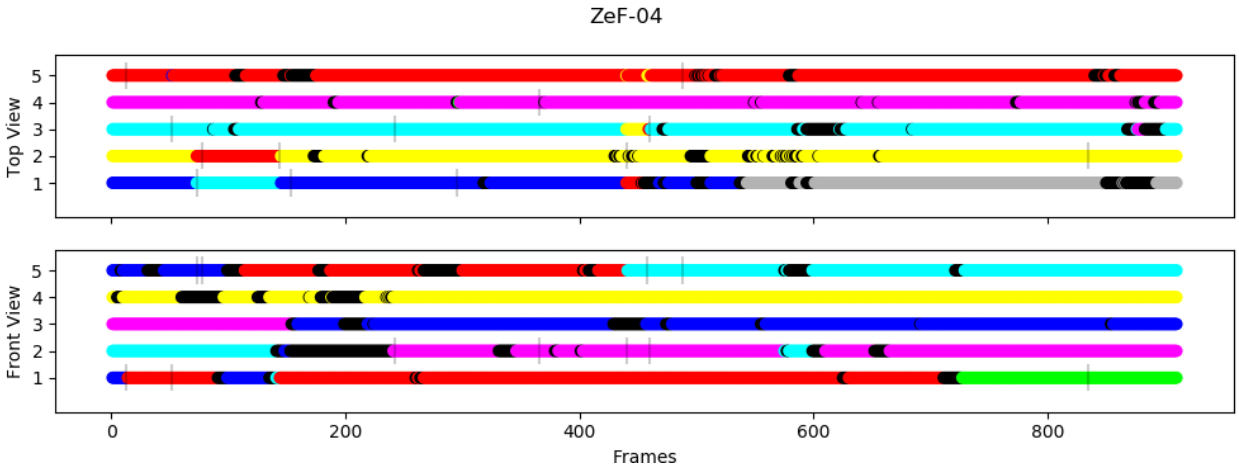
(b) Before swap correction



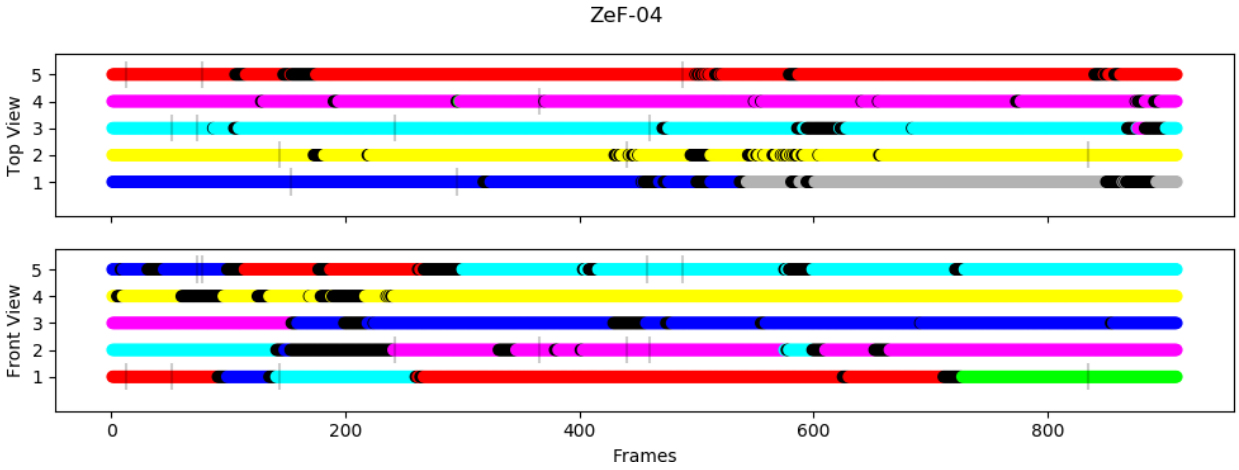
(a) After swap correction



(b) Before swap correction



(a) After swap correction



(b) Before swap correction

Bibliography

[1] Marta Barreiros et al. "Zebrafish tracking using YOLOv2 and Kalman filter". In: *Scientific Reports* 11 (Feb. 2021), p. 3219. DOI: 10.1038/s41598-021-81997-9.

[2] Sean Bell and Kavita Bala. "Learning Visual Similarity for Product Design with Convolutional Neural Networks". In: *ACM Trans. Graph.* 34.4 (July 2015). URL: https://doi.org/10.1145/2766959.

[3] Philipp Bergmann, Tim Meinhardt, and Laura Leal-Taixe. *Tracking without bells and whistles*. 2019. arXiv: 1903.05625 [cs.CV].

[4] Keni Bernardin and Rainer Stiefelhagen. "Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics". In: *EURASIP Journal on Image and Video Processing* (2007). DOI: 10.1155/2008/246309.

[5] E. Bochinski, V. Eiselein, and T. Sikora. "High-Speed tracking-by-detection without using image information". In: 2017. DOI: 10.1109/AVSS.2017.8078516.

[6] E. Bochinski, T. Senst, and T. Sikora. "Extending IOU Based Multi-Object Tracking by Visual Information". In: (2018). DOI: 10.1109/AVSS.2018.8639144.

[7] Guillem Brasó and Laura Leal-Taixé. "Learning a Neural Solver for Multiple Object Tracking". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[8] Peter Carr and Robert T. Collins. "Assessing tracking performance in complex scenarios using mean time between failures". In: Institute of Electrical and Electronics Engineers Inc., 2016. DOI: 10.1109/WACV.2016.7477617.

[9] Xi En Cheng et al. "Obtaining Three-Dimensional Trajectory of Multiple Fish in Water Tank via Video Tracking". In: *Multimedia Tools Appl.* (2018). URL: https://doi.org/10.1007/s11042-018-5755-5.

[10] Gioele Ciaparrone et al. "Deep learning in video multi-object tracking: A survey". In: *Neurocomputing* 381 (2020), pp. 61–88. DOI: https://doi.org/10.1016/j.neucom.2019.11.023.

[11] N. Dalal and B. Triggs. "Histograms of oriented gradients for human detection". In: 1 (2005). DOI: 10.1109/CVPR.2005.177.

[12] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[13] Ross Girshick. *Fast R-CNN*. 2015. arXiv: 1504.08083 [cs.CV].

[14] Richard Hartley and Andrew Zisserman. "Camera Models". In: *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004, pp. 153–177. DOI: 10.1017/CBO9780511811685.010.

[15] J. B. Haurum et al. “Re-Identification of Zebrafish using Metric Learning”. In: (2020). DOI: 10.1109/WACVW50321.2020.9096922.

[16] K. He et al. “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[17] Alexander Hermans, Lucas Beyer, and Bastian Leibe. *In Defense of the Triplet Loss for Person Re-Identification*. 2017. eprint: 1703.07737.

[18] Arne Hoffhues Jonathon Luiten. *TrackEval*. https://github.com/JonathonLuiten/TrackEval. 2020.

[19] Q. Al-Jubouri et al. “Computer Stereovision System for 3D Tracking of Free-Swimming Zebrafish”. In: (2017). DOI: 10.1109/DeSE.2017.31.

[20] Farmanur Rahman Khan and Saleh Sulaiman Alhewairini. “Zebrafish (Danio Rerio) as a Model Organism”. In: (2019). DOI: 10.5772/intechopen.81517. URL: https://doi.org/10.5772/intechopen.81517.

[21] B Y Lee et al. “Occlusion handling in videos object tracking: A survey”. In: *IOP Conference Series: Earth and Environmental Science* (2014). URL: https://doi.org/10.1088/1755-1315/18/1/012020.

[22] T. Lin et al. “Feature Pyramid Networks for Object Detection”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, pp. 936–944. DOI: 10.1109/CVPR.2017.106.

[23] Jonathon Luiten et al. “HOTA: A Higher Order Metric for Evaluating Multi-object Tracking”. In: *International Journal of Computer Vision* 129.2 (2020). ISSN: 1573-1405. DOI: 10.1007/s11263-020-01375-2. URL: http://dx.doi.org/10.1007/s11263-020-01375-2.

[24] Wenhan Luo et al. *Multiple Object Tracking: A Literature Review*. 2017. arXiv: 1409.7618 [cs.CV].

[25] Hans Maaswinkel, Liqun Zhu, and W. Weng. “Using an Automated 3D-tracking System to Record Individual and Shoals of Adult Zebrafish”. In: *Journal of Visualized Experiments : JoVE* (2013).

[26] Simone Macrì et al. “Three-dimensional scoring of zebrafish behavior unveils biological phenomena hidden by two-dimensional analyses”. In: *Scientific Reports* 7 (May 2017). DOI: 10.1038/s41598-017-01990-z.

[27] Anton Milan et al. *MOT16: A Benchmark for Multi-Object Tracking*. 2016. arXiv: 1603.00831 [cs.CV].

[28] Gregory de Oliveira Feijó et al. “An algorithm to track laboratory zebrafish shoals”. In: *Computers in Biology and Medicine* (2018). DOI: https://doi.org/10.1016/j.compbiomed.2018.01.011.

[29] Rafael Padilla et al. “A Comparative Analysis of Object Detection Metrics with a Companion Open-Source Toolkit”. In: *Electronics* 10.3 (2021). ISSN: 2079-9292. DOI: 10.3390/electronics10030279. URL: https://www.mdpi.com/2079-9292/10/3/279.

[30] Malte Pedersen et al. *3D-ZeF: A 3D Zebrafish Tracking Benchmark Dataset*. 2020. arXiv: 2006.08466 [cs.CV].

[31] Alfonso Pérez-Escudero et al. "IdTracker: Tracking individuals in a group by automatic identification of unmarked animals". In: *Nature methods* (2014). DOI: 10.1038/nmeth.2994.

[32] Zhi-Ming Qian, Xi En Cheng, and Yan Qiu Chen. "Automatically Detect and Track Multiple Fish Swimming in Shallow Water with Frequent Occlusion". In: (2014). DOI: 10.1371/journal.pone.0106506. URL: https://doi.org/10.1371/journal.pone.0106506.

[33] Zhi-Ming Qian et al. "An effective and robust method for tracking multiple fish in video image based on fish head detection". In: *BMC Bioinformatics* 17 (June 2016). DOI: 10.1186/s12859-016-1138-y.

[34] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf.

[35] Cristina Santoriello and Leonard I. Zon. "Hooked! Modeling human disease in zebrafish". In: *The Journal of Clinical Investigation* 122.7 (July 2012), pp. 2337–2343. DOI: 10.1172/JCI60434. URL: https://doi.org/10.1172/JCI60434.

[36] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2015). DOI: 10.1109/cvpr.2015.7298682. URL: http://dx.doi.org/10.1109/CVPR.2015.7298682.

[37] Connor Shorten and Taghi Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6 (July 2019). DOI: 10.1186/s40537-019-0197-0.

[38] Hyun Oh Song et al. *Deep Metric Learning via Lifted Structured Feature Embedding*. 2015. arXiv: 1511.06452 [cs.CV].

[39] Adam Michael Stewart et al. "A novel 3D method of locomotor analysis in adult zebrafish: Implications for automated detection of CNS drug-evoked phenotypes". In: *Journal of Neuroscience Methods* (2015). DOI: https://doi.org/10.1016/j.jneumeth.2015.07.023.

[40] Tsegay Teame et al. "The use of zebrafish (Danio rerio) as biomedical models". In: *Animal Frontiers* 9.3 (2019), pp. 68–77. DOI: 10.1093/af/vfz020. URL: https://doi.org/10.1093/af/vfz020.

[41] L. Zhang and L. van der Maaten. "Structure Preserving Object Tracking". In: (2013). DOI: 10.1109/CVPR.2013.240.

[42] T. Zhang and C. Suen. "A fast parallel algorithm for thinning digital patterns". In: *Commun. ACM* 27 (1984), pp. 236–239.