NTNU

Norwegian University of
Science and Technology

Unsupervised Learning of Motion Patterns for Object Classification in Aquaculture

Øyvind Rognerud Karlstad

Master of Science in Cybernetics and Robotics
Submission date: June 2018
Supervisor: Annette Stahl, ITK
Co-supervisor: Christian Schellewald, SINTEF Ocean

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Problem Description

Recent approaches towards image classification have been driven by supervised learning algorithms requiring significant amount of data to be trained for detecting one specific object. For applications aimed to be used in aquaculture the amount of quality data is still limited.
This thesis aims to study an unsupervised method to detect and classify the main objects of interest in a fish cage, which are pellets and fish. By nature, fish and pellets have distinct motion patterns which creates the hypothesis of being able to classify the two types of objects based on their motion.

The tasks of this thesis consists of:

- Study suitable unsupervised learning algorithms to be able to classify objects in a fish farm
- Implement the algorithm and perform tests on provided video
- Study the effect of adding object sizes to the classification

Abstract

In recent years, computer vision has been used in increasing amounts in aqua culture and will be essential for the development of automated solutions for fish farming. In this thesis we study the possibility of using unsupervised learning based on motion patterns to classify main groups of objects in a fish farm. The main focus will be separating fish from feed. The approach is based on the hypothesis that fish and feed have distinct motion patterns that are suitable to use as classification criteria.

The approach is based on optical flow by using KLT-tracking to estimate motion in the image. Similar motion patterns are grouped together using cluster analysis. Mean shift and DBSCAN were chosen as the algorithms to be used in the experiments, based on a preliminary analysis of the motion data. Mean shift is centroid based, while DBSCAN is density based which gives a useful combination of differing properties to compare. Further, the effect of adding object sizes to the clustering was studied. Object sizes were estimated by using image segmentation. The segmentation algorithm is based on edge detection, using a Sobel operator to create gradient images that can be used as basis for finding contours of objects.

Results showed that classifying fish and feed based on motion patterns is plausible under certain conditions. There are some requirements for the camera position that improves the classification. For instance the clustering performance increases when numerous objects is visible in the image. Accurately selecting clustering parameters are also necessary to avoid cluster merging. In cases where several clusters are merged together, all valuable information about the objects are lost.

The effect of adding object sizes to the clustering proved to be as expected. It resulted in improved separability of the motion patterns, although the segmentation accuracy required, made the proposed approach not robust enough to calculate the object sizes automatically. The main issues were incomplete contours because of too low contrast towards the background and overlapping objects. As a cause of the inaccurate segmentation, the number of data samples were too small to draw any conclusions, although the tendencies are that by using a better suited segmentation algorithm, a more consistent classification can be obtained.

Sammendrag

Datasyn har i de senere årene blitt brukt i økende grad innen akvakultur og kommer til å spille en viktig rolle i utviklingen av automatiserte løsninger for oppdrettsnæringen. I denne masteroppgaven undersøker vi muligheten for å bruke uledet læring basert på bevegelsesmønstre til å klassifisere hovedgrupper av objekter i en fiskemerd. Hovedfokuset vil være å skille fisk fra fòr. Hovedhypotesen bak masteroppgaven er at bevegelsesmønstrene til fisk og pellets er ulike nok til at de kan brukes som et klassifiseringskriterie.

Metoden utviklet i dette prosjektet baseres på optical flow ved å bruke KLT-tracking for å estimere bevegelser i bildet. Bevegelsene grupperes ved hjelp av klyngeanalyse for å finne objekter av lik art. Basert på en analyse av bevegelsesdataen ble det besluttet å gjøre eksperimenter basert på algoritmene mean shift og DBSCAN. Mean shift er sentroidebasert og DBSCAN er tetthetsbasert, noe som ga en god kombinasjon av ulike egenskaper for sammenligning. I tillegg ble effekten av å inkludere objektstørrelser i klyngeanalysen studert. Objektstørrelsene ble funnet ved hjelp av bildesegmentering. Segmenteringen ble gjort ved hjelp av kantdeteksjon basert på en Sobel operator. Operatoren brukes til å kalkulere gradientbilder som danner grunnlaget for å finne konturer av objekter i bildet.

Resultatene viste at fisk og pellets kan klassifiseres basert på bevegelsesmønstre under visse forhold. Det er noen krav til kameraplassering som bidrar til en forbedret klassifisering. For eksempel vil klyngeanalysen gi et mer nøyaktig resultat dersom mengden av synlige objekter i bildet er mange. Det er i tillegg nødvendig med nøyaktig valg av klyngeanalyse-parametere for å unngå sammenslåing av klynger. I tilfeller hvor flere klynger slåes sammen på grunn av dårlig parametervalg, vil all nyttig informasjon om objektene gå tapt.

Effekten av å inkludere objektstørrelser i klyngeanalysen viste seg å være lovende. Det ga en større separering mellom klyngene, men kravet til nøyaktigheten i segmenteringen gjorde at den foreslåtte metoden ikke var en robust løsning. Ufullstendige konturer på grunn av lav kontrast mot bakgrunnen og overlappende objekter var hovedproblemene. På grunn av unøyaktighetene i segmenteringen ble ikke datagrunnlaget for å trekke konklusjoner om å inkludere objektstørrelser stort nok, men med en bedre segmenteringsalgoritme kan det bidra til en mer konsistent klassifisering.

Preface

This master thesis is written as a part of my Master of Science degree in engineering cybernetics at the Norwegian University of Science and Technology (NTNU) in cooperation with Sealab AS. It is a continuation of the pre-project submitted during the fall semester of 2017, where an algorithm for estimating motion patterns for objects in underwater videos was proposed.

I would like to thank my supervisor Annette Stahl and co-supervisor Christian Schellewald for guidance towards possible solutions to the problem at hand, as well as advice on how to structure the report. Thanks are also due to Sealab AS for providing high quality underwater videos from fish cages. In addition, Sealab AS provided an office space and valuable domain knowledge about aquaculture.

Finally, I would like to thank my family for their support during my studies.

Øyvind Rognerud Karlstad
Trondheim, June 4, 2018

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background and Motivation

Computer vision has grown to be a large research field in recent years and opens up for a vast variety of applications. With the decreasing cost of processing power in addition to the development of artificial intelligence, computer vision has expanded from pure industrial usage to appear in everyday applications. Ranging from putting a dog filter on your face in Snapchat to sophisticated artificial intelligence systems used to detect cancer at an early stage. The applications are endless.

The aquaculture industry will be fundamental for Norway's future, both economically and environmentally. With the growing population and decreasing arable land, it is crucial to utilize more of the ocean space. Some issues the industry currently tries to address includes sea lice, feeding optimization and high mortality. According to a study from the Norwegian Environment Agency published in 2011 [1] (Norwegian article), a fish farm that has an annual production of 1000-2000 tonnes salmon is estimated to have an emission of 150 - 300 tonnes dry feed, 15 - 30 tonnes nitrogen and 5 - 10 tonnes phosphorus. This is an issue that needs to be addressed. Computer vision can contribute to improvements for these issues. Sea lice and feed can be improved with detection algorithms and the mortality can be studied with an attitude analysis to try to discover the causes.

In this thesis we will investigate how suitable motion patterns are for classification. If motion pattern estimation can be combined with unsupervised machine learning to separate objects into main groups like pellets, fish and nets, it can be an effective tool for determining regions of interest or extract objects in an image for further processing. This could lead to a classifier that does not need to be pre-trained and will adapt to the scene it is placed in. Possible applications could be pellet detection to improve the feed conversion ratio, salmon detection to create a region of interest to scan for sea lice or maybe even detect different motion patterns based on the health of the fish.

1.2 Thesis Outline

In section 2 some theoretical background about unsupervised learning, clustering and image segmentation will be given. Section 3.1 contains an analysis of the raw data provided to determine which approaches that were suitable. The chosen clustering algorithms will be presented in section 2.3 and 2.4. In section 3.3 the required input parameters and approaches to dynamically adjust them to fit the input data is discussed.
Section 3.2 presents the proposed algorithm to use for classifying different objects in a fish farm. Section 4 describes how the algorithm was implemented. Section 5 presents the experiments that were done using the algorithm. The results from the experiments are presented in section 6. In section 7 a discussion of strengths and weaknesses of the proposed algorithm is given, including suggestions for future work.

1.3 Related Work

Unsupervised learning of motion patterns is not a new research field. There are some previously published researches based on the concept [2], [3], [4] used for learning movement patterns for cars. Although any research related to using motion patterns as classification criteria in aquaculture has not been found.

Chapter 2

Applied Methods

2.1 Machine Learning

Machine learning is the process of using statistical techniques to give computers artificial intelligence and the ability to learn based on input data. With the development of artificial intelligence, computers can be given the ability to progressively improve performance on a specific task without being explicitly re-programmed. Machine learning enables the use of automated solutions for predictions and analysis of data. Machine learning can be separated into two main approaches, supervised and unsupervised learning algorithms.

Supervised learning algorithms create models based on labeled input data. Supervised learning can be compared to learning with a teacher telling you if you answer is right or wrong. For each data sample presented to the algorithm, an output is created based on the current model. If the output does not correlated to the teachers answer, the model is adjusted to be better suited to answer the question when similar data is presented again. In this sense the algorithm learns based on input data to be suited for classifying new data. An issue related to supervised learning is overfitting. Overfitting is related to poor generalization. When a model is trained using data with low variance, the model will generally be too specified and will not be able to classify new data with small deviations from the training data accurately, which is known as overfitting.

2.1.1 Unsupervised Learning Algorithms

Unsupervised learning is the process of trying to describe hidden structures from unlabeled data. In comparison to supervised learning, evaluating the outputs of unsupervised algorithms are not as trivial, because ground truth labels may not exist. This means quantitative results may not be available for evaluation of the method and data interpretation will be more important.
The advantages of unsupervised approaches are that no data preparation is needed, thus it

can be used in new environments without any knowledge of what to discover.

Cluster analysis is a common unsupervised learning method. Cluster analysis is used for exploratory data analysis to find hidden patterns in data. Clusters are detected using a similarity measure which is defined by metrics such as Euclidean distance or Manhattan distance.

Several approaches towards generating clusters exists. In this thesis we will focus on centroid-based and density-based clustering algorithms.

Centroid-based clustering uses an iterative approach to cluster data where the similarity is derived by the distance of a data point to the centroid of the cluster. The optimization problem associated with centroid-based clustering algorithms are known to be NP-hard, thus a common approach is to search for approximate solutions. Popular algorithm that fall into this category are k-means [5], k-medians [6] and mean shift [7]. A common attribute for two first algorithms is that the number of clusters, k, have to be specified in advance, which makes it important to have domain knowledge about the dataset. Hence giving mean shift an advantage compared to the other centroid-based algorithms.

Density-based clustering searches the feature space for areas with higher density than the rest. These denser areas are defined as clusters. Sparse regions in the feature space is either considered noise or border points. Popular density-based algorithms are DBSCAN [8] and OPTICS [9].

2.2 Image Segmentation

Image segmentation is a large research field within computer vision. It is the process of partitioning an image into parts for further analysis. Image segmentation is typically used to separate objects or other interesting regions in images. Even though image segmentation have been researched for decades, it is a complex problem and a global solution fitting all images are yet to be found. With that said, some recently developed neural networks [10] [11] trained for semantic segmentation are performing very well. Prior methods have mainly consisted of approaches based on region growing, clustering or edge detection. In this thesis we will assume training data is limited, hence give preference to unsupervised algorithms. That means neural net algorithms will not be used to solve the problem in this thesis.

2.2.1 Edge Detection Segmentation

One popular approach to image segmentation is based on detecting edges using discontinuity of intensity levels in an image. Edges can be described as local changes of intensity and typically occur on the boundary between two regions. This makes it useful to separate objects from a background, for instance.
In 1968 Irwin Sobel and Gray Feldman presented an 3×3 image gradient operator [12].

The operator is referred to as the Sobel operator and is commonly used for edge detection. The Sobel operator is defined as

$$S := \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{2.1}$$

It is used as a discrete differentiation operator and can be used for computing an approximation of the image gradient. The process of detecting discontinuities of intensity in an image, **I**, using the Sobel operator can be defined by the following sequence of mathematical operations:

$$\begin{aligned} \mathbf{G_x}[x,y] = (\mathbf{I} * \mathbf{S})[x,y] = \sum_{j=-1}^{j=1} \sum_{k=-1}^{k=1} \mathbf{I}[x-k, y-j]\mathbf{S}[k,j], \\ \mathbf{G_y}[x,y] = (\mathbf{I} * \mathbf{S^T})[x,y] = \sum_{j=-1}^{j=1} \sum_{k=-1}^{k=1} \mathbf{I}[x-k, y-j]\mathbf{S^T}[k,j], \\ \mathbf{G}[x,y] = \sqrt{\mathbf{G_x^2}[x,y] + \mathbf{G_y^2}[x,y]} \end{aligned} \tag{2.2}$$

where **I** is a $w \times h$ matrix and w and h is the width and height of the image. $[k, j]$ represents integer coordinates in the Sobel operator **g**. $[x, y]$ are integer pixel coordinates in **I**.
The resulting image gradient, **G**, represents edges with areas of increasing intensity.

2.3 DBSCAN - Density-Based Spatial Clustering of Applications with Noise

In 1996 Ester, Kriegel, Sander and Xu proposed a density-based clustering algorithm [8]. Previous partitioning algorithms like k-means clustering, depend on the number of clusters k as an input parameter. This means some domain knowledge is required which may not be available for all applications. Instead DBSCAN takes two parameters as input, ϵ and $minPts$. $minPts$ represents a threshold of how many points are needed to form a cluster. ϵ is a distance threshold, defining the density of points required for being considered a cluster.

To be able to evaluate the clusters, a formalization of what a cluster is, is needed. For the DBSCAN algorithm, the following definition were made in [8, p. 228]:

1. The *Eps-neighborhood* of a point p, denoted by $N_{Eps}(p)$, is defined by $N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\}$

2. A point p is *directly density-rechable* from a point q wrt. Eps, MinPts if $p \in N_{Eps}(q) and | N_{Eps}(q)| \geq MinPts$

3. A point p is *density reachable* from a point q wrt. Eps and MinPts if there is a chain of points $p_1, ..., p_n, p_1 = q, p_n = p$ such that p_{i+1} is directly density-reachable from p_i.

4. A point p is *density-connected* to a point q wrt. Eps and MinPts if there is a point o such that both, p and q are density-reachable from o wrt. Eps and MinPts.

5. Let *D* be a database of points. A *cluster C* wrt. Eps and MinPts is a non-empty subset of D satisfying the following conditions:

   (a) $\forall p, q$: if $p \in C$ and q is density-reachable from q wrt. Eps and MinPts, then $q \in C$

   (b) $\forall p, q \in C : p$ is density-connected to q wrt. Eps and MinPts

6. Let $C_1, ..., C_k$ be the clusters of the databse D wrt. Eps_i and $MinPts_i, i = 1, ..., k$. Then we define the *noise* as the set of points in the database D not belonging to any cluster C_i, i.e. $noise = \{p \in D | \forall i : p \notin C_i\}$

The DBSCAN algorithm starts with an arbitrary point p. If any points are density-reachable from p with respect to ϵ and MinPts, these points form a cluster. Border points will be classified by not having any density reachable points with respect to ϵ and MinPts, but be density-reachable from core points, hence part of the cluster. After all points in the database have been evaluated, all points not part of a cluster are considered as noise.

Recently there have been some discussions about the run time of DBSCAN. In 2015 Gan and Tao [13] published a paper claiming the time complexity of $O(nlogn)$ presented in the original paper was a mis-claim. According to Gan and Tao the algorithm required $O(n^2)$ and moved on to claim DBSCAN should not be used . This paper won the SIGMOD [14] 2015 best paper award and thus gained a lot of attention. In 2017 a response paper were published [15] pointing out inaccuracies and showing the criticism were directed towards wrong assumptions. It was shown that a running time of $O(nlogn)$ cannot be guaranteed, but that does not imply, as Gan and Tao stated, that

"DBSCAN in $d \geq 3$ is computationally intractable in practice even on moderately large n." [15, p. 18]

2.4 Mean Shift

Mean shift is an algorithm used for locating the maxima of a density function. The algorithm uses a sliding-window-based approach to search for the region with the highest density. The search window is defined by a kernel function $K(x_i - x)$, often referred to as a Parzen window [16]. The kernel function determines the weight of nearby points for

re-estimation of the point density.
By considering a dataset *D*, mean shift uses an iterative hill climbing approach to search for the highest density of points as described in the following list.

1. Choose an arbitrary point, x, in the dataset *D* that will be the center of the initial sliding window.

2. Calculate the weighted mean $m(x)$

3. Shift the sliding window according to $x^{t+1} = x^t + m(x^t)$

4. Repeat step 2 and 3 until convergence and local or global maximum density is found

5. Repeat step 1 through 4 until all points in *D* have been assigned to a cluster

The weighted mean of a window is defined by

$$m(x) = \frac{\sum_{x_i \in N(x)} K(x_i - x)x_i}{\sum_{x_i \in N(x)} K(x_i - x)} \tag{2.3}$$

where $N(x)$ is the neighborhood of points within a given radius of x. The weighted mean $m(x)$ is a vector pointing in the direction of the largest increase of point density from x, commonly called the mean shift vector. This process is well suited to be computed using parallel programming to speed up the process.
The number of clusters is estimated based on how many convergence points are found and does not have to be predetermined, which is an advantage of mean shift. When all convergence points are discovered, a cluster is defined by all data points in the attraction basin of a convergence point. The attraction basin is the region where all searching trajectories lead to the same convergence point.

Figure 2.1 shows the main components in the hill climb search. The region of interest is the Parzen window defined by the bandwidth and the current searching position. Center of mass is calculated for each iteration and becomes the next position to search from. The mean shift vector is the vector defining where the search window is shifted to in the next iteration and is also used as a convergence criteria. When the mean shift vector length is below a defined threshold, convergence is reached and the hill climb search is terminated.

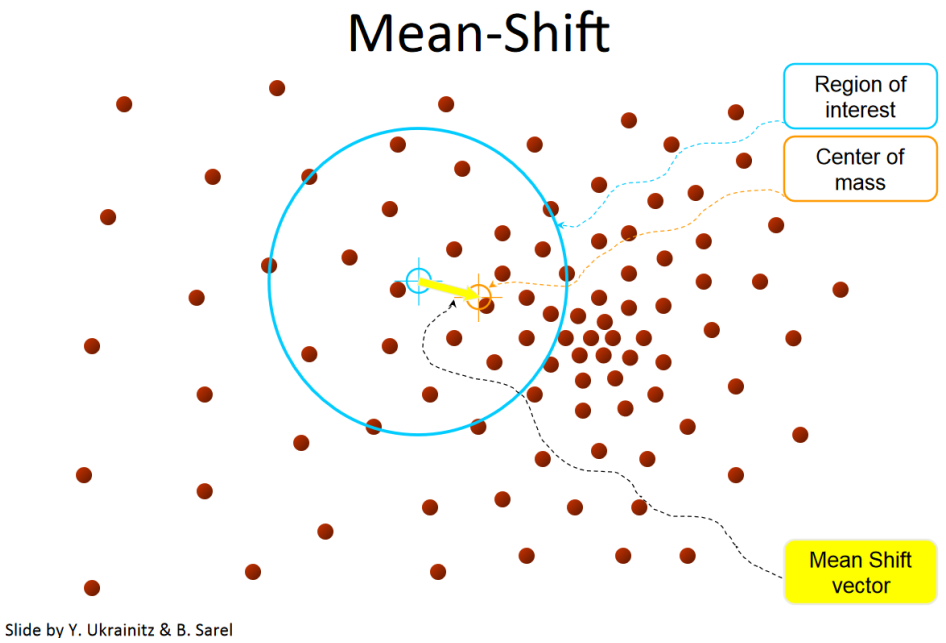


Figure 2.1: One iteration of the mean shift hill climb showing main parts.
Image source: `http://vision.stanford.edu/teaching/cs131_fall1314_nope/lectures/lecture13_kmeans_cs131.pdf`, Accessed June 3, 2018

Chapter 3

Approach

3.1 Preliminary Analysis

Prior to applying any algorithms for classifying the dataset, a raw data analysis was performed to find potential patterns and similarities between different groups of objects. The underlying assumption behind this procedure is that in certain environments, for instance in fish cages, separation of object classes can be done using motion patterns as classification criteria.

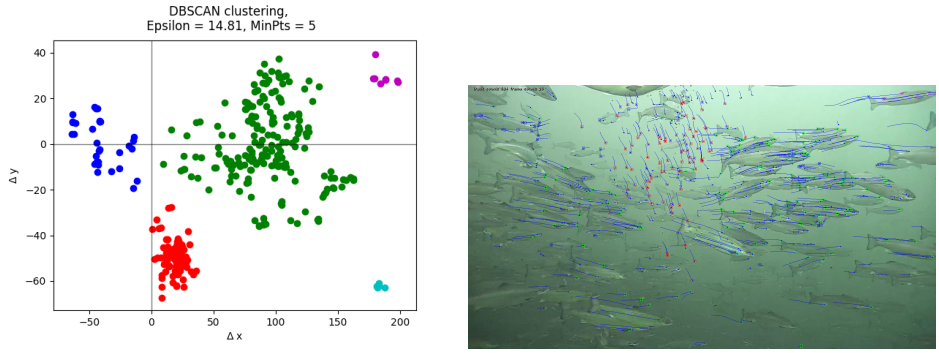


Figure 3.1: Clustered motion data created by tracking feature for 20 frames

Figure 3.1 shows a plot of the distribution of relative motion for each object successfully tracked over a time sequence of 20 frames. The relative motion is computed by subtracting the stored position in the image of a feature when it is first detected, by the current position of the feature. Hence the axes in figure 3.1 represents the number of pixels a feature has moved during a time sequence of 20 frames. This axis representation will be used in all subsequent plots of motion patterns.

By analyzing the plot of motion patterns it can be seen that for most cases, the different classes of objects are quite spread out.

Figure 3.2 shows that the underlying distribution of motion are close to independent of the number of frames in the tracking sequence. The distance between the cluster centers is obviously lower with fewer frames in the tracking sequence.

Based on the distribution of motion patterns seen in figure 3.1 and 3.2, it seems objects of different classes can be separated based on the density of points. Because of the proposed area of usage of the algorithm, the number of clusters to search for should not be predetermined, hence the number of suitable clustering algorithms gets limited. In this thesis we have decided to look further into the centroid based algorithm mean shift and the density-based algorithm DBSCAN.

In the pre-project a discussion was brought up regarding the reliability of feature tracking:

> *Because of the inaccuracy of using a pure frame-to-frame optical flow, using it for classification of motion is unlikely to give a reliable result. Therefore we propose a way to filter out unreliable flow estimations. We decided to include a backtracking algorithm.*
> *Motion estimated for feature n is accepted if*
>
> $$|m(f_n, f_{n+1}) - m(f_{n+1}, f_n)| < d$$
>
> *where m is the estimated motion of a feature in frame n. By using backtracking we can make sure that each feature is consistently tracked for a given time series before considering it as a reliable motion estimate.*

Consequences of this design choice affects the amount of data that can be extracted and used as input to the clustering. The tracking is not perfectly consistent for all cases, hence tracked features will be discarded. Therefore it follows that a larger tracking length threshold, will reduce the amount of features to the input of the clustering.

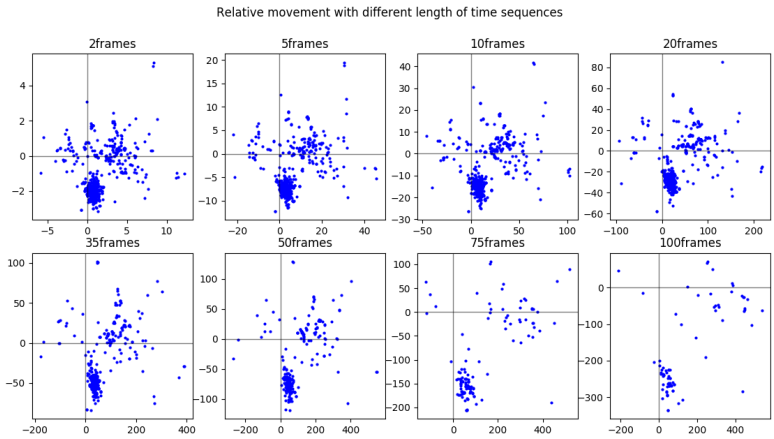


Figure 3.2: Differences in relative movement using varying number of frames

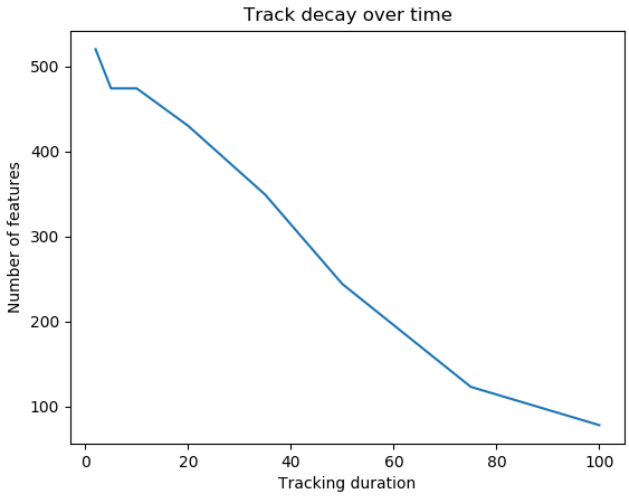


Figure 3.3: Decay of number of features tracked based on number of frames

Figure 3.3 shows how the number of successfully tracked features decay with longer tracking sequences. Of the 520 initial tracked features, only 78 are left after 100 frames. The videos used have been recorded using 25 frames per second, which means after 4 seconds, 84.4 % of the tracked features have been discarded. By observing figure 3.2 one can see that for the features successfully tracked for 100 frames, the resulting feature space is sparse and can easily be separated into clusters. The effect of different tracking lengths will be explored in section 5 and discussed in section 6.

3.2 The Proposed Algorithm

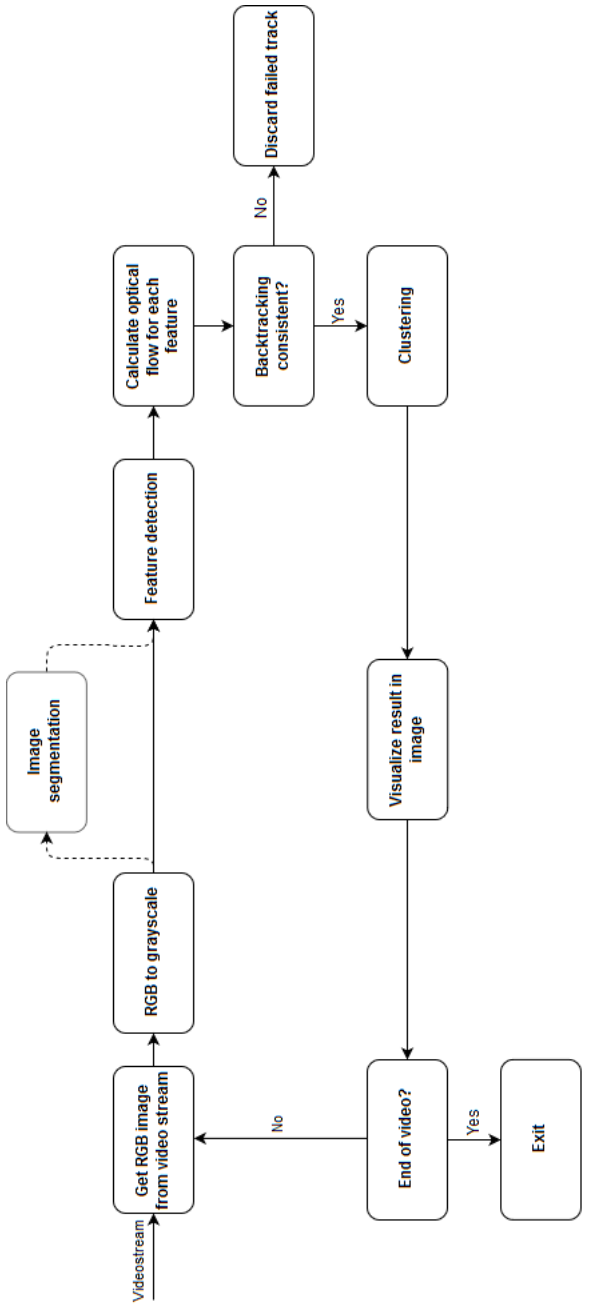


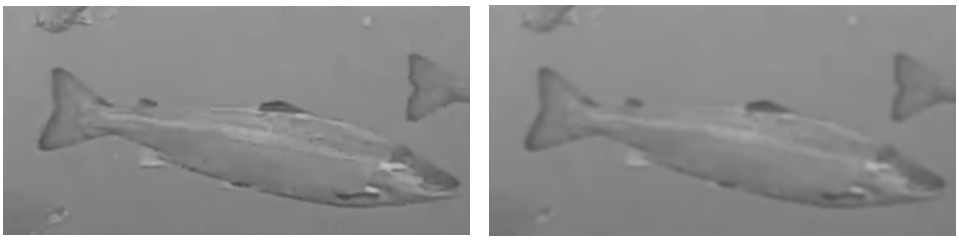
Figure 3.4: Flowchart of the proposed algorithm

Whenever a new frame is read from a video stream, it is converted to a grayscale image. Converting a RGB-image to grayscale is beneficial because it requires less computation in future steps. Information-wise, the loss is minimal as luminance is far more important than chrominance in distinguishing visual features. The resulting gray image used in the algorithm will be denoted $\mathbf{I}$.

Using $\mathbf{I}$, the user can choose whether to segment the image or not, visualized using a dotted line in figure 3.4. The image segmentation creates regions of interest and makes it possible to get information of the size of objects, but introduces a new step in the algorithm, which affects the run-time. To be able to segment out fish and pellets, $\mathbf{I}$ is first smoothed. By smoothing the image, an edge detector will be more robust towards noise. Smoothing of the image is done by convolving $\mathbf{I}$ with a Gaussian kernel, $\mathbf{g}$. $\mathbf{g}$ is a $n \times n$ matrix approximation of a Gaussian distribution. The resulting smoothed image can be denoted:

$$\mathbf{I_s}[x, y] = (\mathbf{I} * \mathbf{g})[x, y] = \sum_{j=-\frac{n-1}{2}}^{j=\frac{n-1}{2}} \sum_{k=-\frac{n-1}{2}}^{k=\frac{n-1}{2}} \mathbf{I}[x-k, y-j]\mathbf{g}[k, j] \tag{3.1}$$

where $\mathbf{I}$ and $\mathbf{I_s}$ are $w \times h$ matrices and w and h is the width and height of the image. $[k, j]$ represents integer coordinates in the Gaussian kernel $\mathbf{g}$. $[x, y]$ are integer pixel coordinates in the images $\mathbf{I}$ and $\mathbf{I_s}$.



(a) Part of an image before smoothing

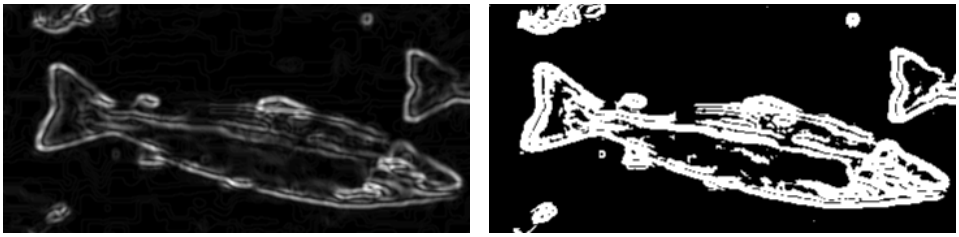
(b) Part of an image after smoothing

Figure 3.5: The effect of smoothing an image using a Gaussian kernel

By using edge detection segmentation based on the Sobel operator, the gradient image $\mathbf{G}$, can be calculated based on the smoothed image $\mathbf{I_s}$ using the approach explained in section 2.2.

From the gradient image $\mathbf{G}$, shown in sub-figure 3.6a, it can be seen that minor particles in the water, as well as some image noise is visible, hence further filtering is needed. It is apparent that the magnitude of the image gradient of the noise is smaller in magnitude, thus a threshold value for the magnitude can be used to remove it. This can be achieved by discarding all values lower than a threshold, T_g, and increasing all value higher than T_g to a $maxValue$ corresponding to the maximum intensity of a pixel in the image. The resulting image will be denoted, G_{Tr}. This can be expressed by

$$\mathbf{G_{Tr}}[x, y] = \begin{cases} maxValue & \text{if } \mathbf{G}[x, y] \geq T_g \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$



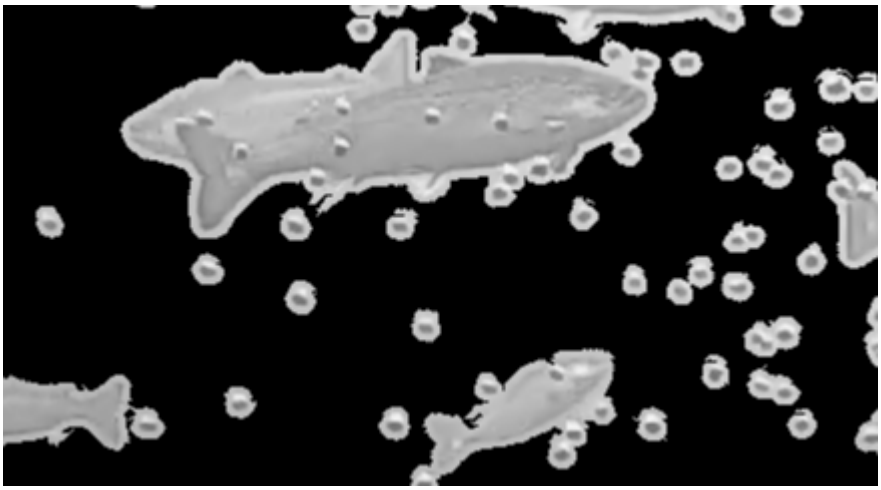
(a) Gradient image of the smoothed image **(b)** Gradient image after using a threshold of 30

Figure 3.6: Effect of applying a threshold on the gradient image

By the look of it, nothing is filtered out, but notice that there are several dim lines in the background of I_g in figure 3.6a that are removed in 3.6b.

It can clearly be seen that some of the smaller blobs in the image do not originate from any of the objects of interest. To filter them out, the area of each blob is calculated and all contours with an area below a threshold, T_a, is discarded. To calculate the area of a blob the OpenCV functions findContours [17] and contourArea [18] is used. FindContours uses a border following algorithm [19] to discover all contours in a binary image. ContourArea implements Green's theorem [20] to calculate the area of each contour by line integrals.

(a) Original frame



(b) Segmented frame

Figure 3.7: The effect of segmenting an image

Figure 3.7 shows a part of an segmented image.

After segmenting objects from the background, using the segmented image G_{Tr} as a mask, feature detection and tracking is performed as described in the pre-project [21] using KLT-tracking. A limit is set on the lowest distance between features to increase the chance of detecting features in every object in the image and reducing the risk of detecting duplicates of the same feature. The limit was implemented by masking out a circle around all currently tracked features during the feature detection.
Based on the motion data calculated using the tracked features, cluster parameters are selected using the approach described in section 3.3. All clustered feature points are marked

in the original frame with colors corresponding to the color used in the cluster plot as shown in figure 3.8.

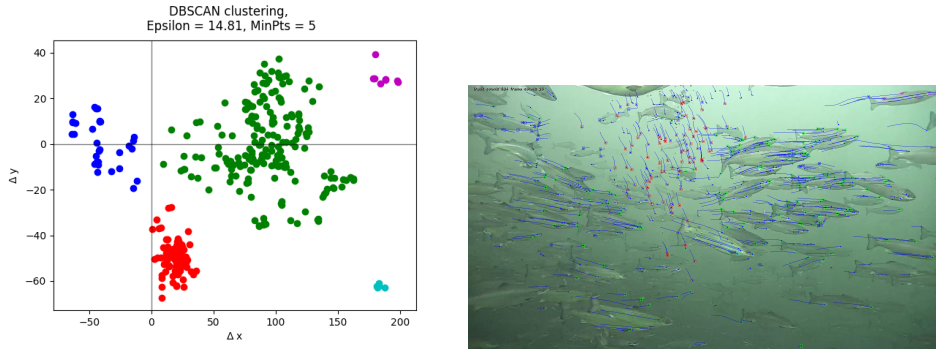


Figure 3.8: Example output from the clustering

3.3 Parameter Tuning

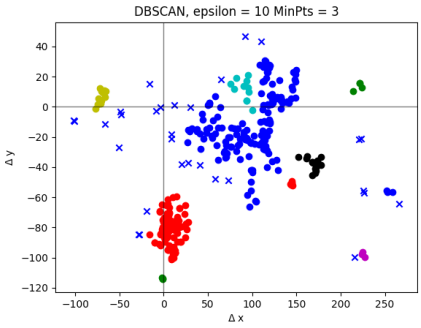
There are several parameters that needs to be adjusted for the motion clustering to work as intended. Dynamically adjusted parameters are necessary because, based on the environment, objects will move differently, leading to variations in the distribution of feature points. Hence a static parameter for cluster densities and cluster sizes will not suffice for an unsupervised algorithm adapting to the scene.

3.3.1 DBSCAN

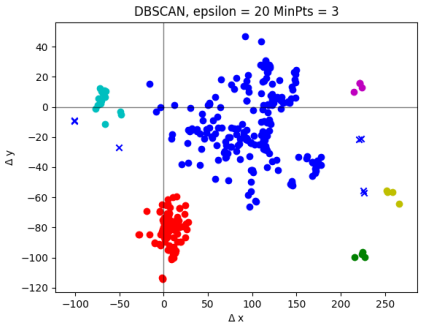
DBSCAN requires two parameters as input, ϵ and MinPts. As described in section 2.3, ϵ describes the density of the clusters. Higher values leads to more sparse clusters, lower values leads to denser clusters. Related to the motion patterns, ϵ has to be adjusted correlated to the tracking length. By tracking objects for a longer time, the feature space will be wider requiring a larger value of ϵ. MinPts is used as a noise filter. MinPts sets the limit for the minimum number of points in a cluster. This means a new cluster will be formed, only if the number of points within the epsilon neighborhood is above MinPts, else it is considered as noise.

Figure 3.9 shows a series of clustered motion data with varying ϵ. Sub-figure 3.9a shows the effect of setting ϵ too low, creating sub-clusters inside more natural clusters. Sub-figure 3.9b is an example of a more reasonable clustering, where main groups of motion patterns are clustered together and certain deviant motion patterns are creating clusters and not considered as noise. Sub-figure 3.9c shows how a too large ϵ value leads to clusters being merged.

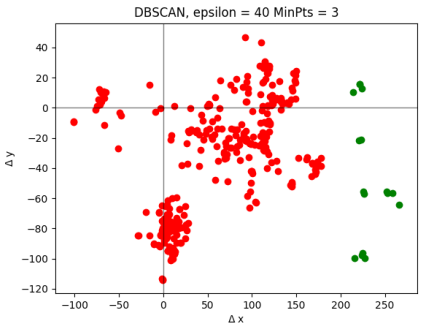
Figure 3.10 shows the effect of varying MinPts. We can see that by adjusting MinPts the amount of feature points considered as noise increase. For the application proposed in



(a) DBSCAN clustering using $\epsilon = 10$



(b) DBSCAN clustering using $\epsilon = 20$



(c) DBSCAN clustering using $\epsilon = 40$

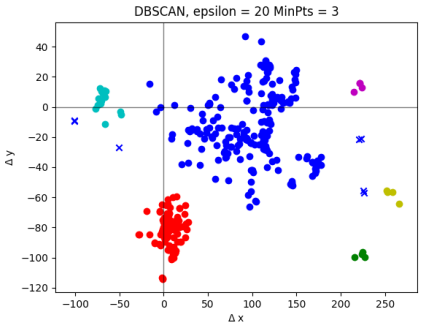
Figure 3.9: The effect of ϵ on the clustering

this thesis, MinPts has to be set according to how many features are expected within an object. For instance, if there are three feature points detected and successfully tracked within a salmon that are moving irregularly and MinPts is set to a value higher than three, this salmon will not be included in the classification and thus leading to important information loss.

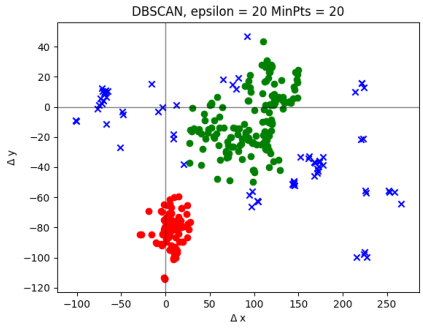
By observing the example clustering shown in figure 3.9 and 3.10, an understanding of the importance of correct and dynamic parameter selection can be established.

Ester, Kriegel, Sander and Xu developed a simple heuristic to estimate the two input parameters for DBSCAN. It is based around estimating the density of the ”thinnest” cluster in the database.
As stated in [8, p. 230]:

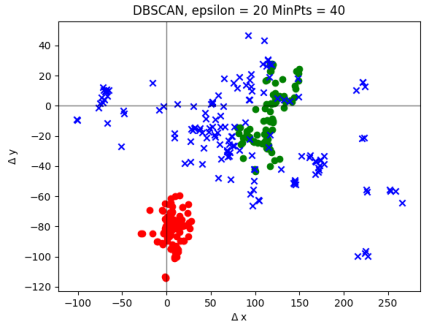
> *”This heuristic is based on the following observation. Let d be the distance of a point p to its k-th nearest neighbor, then the d-neighborhood of p contains exactly k+l points for almost all points p. The d-neighborhood of p contains more than k+l points only if several points have exactly the same dis-*



(a) DBSCAN clustering using $MinPts = 3$



(b) DBSCAN clustering using $MinPts = 20$



(c) DBSCAN clustering using $MinPts = 40$

Figure 3.10: The effect of MinPts on the clustering

> *tance d from p which is quite unlikely. Furthermore, changing k for a point in a cluster does not result in large changes. This only happens if the k-th nearest neighbors of p for k= 1,2,3 .... are located approximately on a straight line which is in general not true for a point in a cluster."*

By defining a function k-dist as the distance between a point in the database to its k'th nearest neighbor, one can compute the k-dist for each point in the database. By sorting the resulting k-dist values in descending order, the sorted k-dist can be plotted to get a graph showing the density distribution in the dataset. Figure 3.11 shows two examples of a k-dist graph.

Ester, Kriegel, Sander and Xu's proposed heuristic was to choose ϵ = k-dist(p), where p is the first point in the first "valley" of the sorted k-dist graph. Further they suggest to set $MinPts = k$ and treating all points with a higher k-dist than ϵ as noise. Although this is a simple heuristic to follow with human interaction, it is generally difficult to detect the first valley automatically. Therefore the following semi-automatic procedure was proposed:

1. Compute and display the sorted k-dist graph for the database

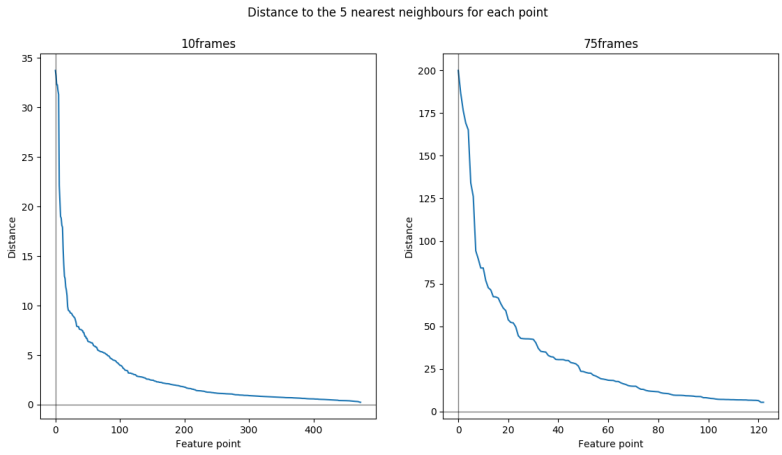


Figure 3.11: Distance to the 5th nearest neighbors in the motion data from one frame with two different track lengths

2. The user inputs the percentage of noise and the system derives a proposal for the threshold point from it
3. The user either accepts the threshold or selects another point

The approach used in this thesis is based on this heuristic, but eliminates the direct user-interaction. Instead of having a user accept or decline a proposed threshold, the percentage of noise is entered and kept until the user changes it and restarts the application. Additionally $MinPts$ was set using the following formula:

$$MinPts = max(ln(n), 4) \tag{3.3}$$

where n is the number of feature points given as input to the clustering. 4 was empirically discovered to be a lower threshold for number of points in a cluster, performing well for the aquaculture case. By choosing $MinPts = ln(n)$ in cases where it is greater than 4, better scalability is achieved for large datasets.

Another parameter that should be considered, if the algorithm is to be used with data of higher dimensions, is the distance measure. Different distance measures has a significant impact on the choice of ϵ and the clustering results. In this thesis we use the Minkowski distance measure which defines the distance between two points $\mathbf{X}$ and $\mathbf{Y}$ as:

$$d(\mathbf{X}, \mathbf{Y}) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{3.4}$$

where p is defined as the order of the Minkowski distance and n is the number of dimensions used. In this thesis p was set to 2, which leads to the more common Euclidean distance measure. When clustering data of higher dimensions, other distance measures should be considered because of the curse of dimensionality [22].

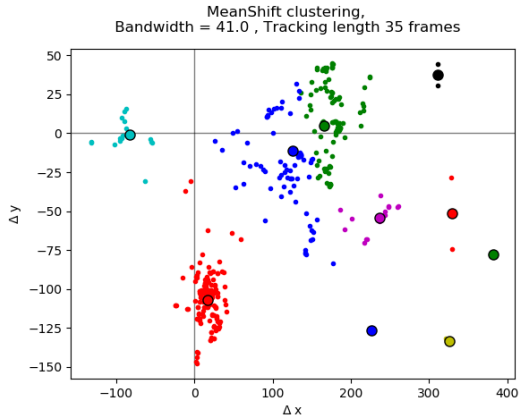
3.3.2 Mean Shift

The only parameter required by mean shift is the bandwidth. Bandwidth is the parameter that sets the radius of the Parzen window. For the implementation used, bandwidth will not be directly addressed. Instead, it is estimated based on the distance to the k nearest neighbors of batches from the dataset. k is determine by a quantile of the size of the dataset, where the quantile is the input parameter used for the algorithm. The returned bandwidth will represent the density of the dataset by summing the farthest distance between two points in each batch divided by the size of the dataset. Algorithm 1 shows pseudo-code for the process of estimating the bandwidth.

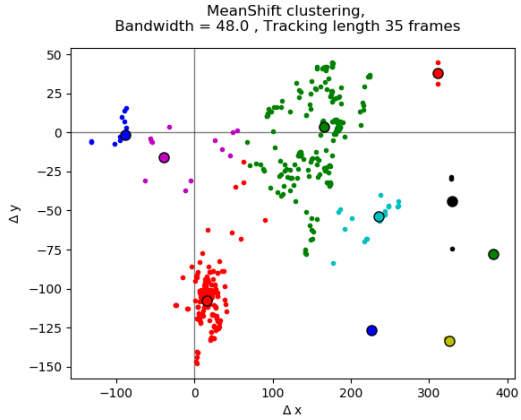
Data: Input dataset, quantile
Result: Bandwidth
batches = divide dataset into batches of 500;
bandwidth = 0;
forall *batches* **do**
    d = distance to the (size(dataset) · quantile) neighbors
    bandwidth += d
end
return bandwidth / size of dataset

Algorithm 1: Pseudo-code for bandwidth estimation

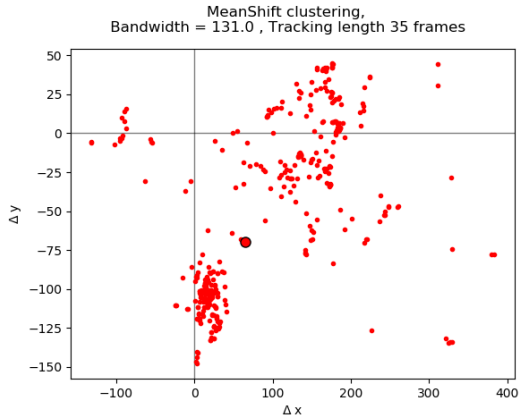
Figure 3.12 shows the effect of adjusting the quantile parameter. It can be seen that by increasing the quantile, i.e. increasing the number of points to base the size of the search window on, the density and number of clusters is reduced.



(a) Mean shift clustering using $quantile = 0.1$



(b) Mean shift clustering using $quantile = 0.2$



(c) Mean shift clustering using $quantile = 0.5$

Figure 3.12: The effect of using different quantiles of the dataset for bandwidth selection

Chapter 4

Implementation

4.1 Functional Description

The applicatoin is implemented using Python 3 [23]. Python was chosen because of it's intuitive syntax and range of external libraries for visualizing data.

Parts of the algorithm were implemented using C++. More specifically the nearest neighbor search for calculating the k-dist to select ϵ for DBSCAN clustering. This part was found to be particularly slow using Python and thus had to be implemented more computationally efficient. To be able to use C++ functions in Python, a wrapper is needed. CTypes [24] were chosen as the C++ wrapper. CTypes is a foreign function library for Python providing C compatible data types and allowing function calls from shared libraries. Note that the code appended with this thesis is built for an UNIX environment and have to be recompiled to run on Windows.

To create a clean and tidy setup of all the parameters for the algorithms, a JSON [25] configuration file is used. JSON is a lightweight data-interchange format used to store name/value pairs that are easy for humans to read and write. JSON is completely programming language independent, making it perfect for storing configuration parameters.

When running the program there are some interactivity options. Table 4.1 shows a complete list of interactive functionality available.

4.2 External libraries

Several open-source libraries have been used to implement the proposed algorithm to lessen the development time. Table 4.2 shows a list of used libraries and their licensing showing they are free to use for academic purposes. The following subsections presents the purpose of the main libraries used.

Hot-key	Function
Esc	Exit program
p	Pause
f	Toggle plotting
k	Toggle classification
s	Toggle segmentation

Table 4.1: List of hot-keys and their implemented functionality

Library	License
sklearn	BSD
PyClustering	GNU GPL
OpenCV	BSD
ccore	BSD
SciPy	BSD
NumPy	BSD
MatPlotLib	BSD
PIL	PIL Software License
TKinter	GPL

Table 4.2: List of external libraries used and their licensing

4.2.1 Scikit-Learn

Scikit-Learn [26] is an open-source library implementing several algorithms and is a great toolbox for machine learning in Python. Scikit-Learn is released under the BSD license which means it can be used freely without any restrictions. In this thesis we have used the Mean shift implementation from the Scikit-Learn library.

4.2.2 PyClustering

PyClustering [27] is a data mining library providing C++ and Python implementations of various algorithms. The appealing feature included in PyClustering's implementations is the use of CCORE, which executes the most computationally heavy parts of the algorithms using C++ instead of Python, hence making it faster.

4.2.3 Matplotlib

Matplotlib [28] can be used to create great plots in Python. In this thesis it was used to create feature plots and visualizing clusters.

Chapter 5

Experiments

Experiments were performed using the proposed algorithm with recorded video provided by Sealab AS. Videos provided were recorded at Kåholmen, Hitra during daytime at depths of 2-10 meters in April and May. This means the algorithm has not been tested using artificial lighting. The videos were recorded using 1920x1080 HD resolution at 25 frames per second.

In total, six different approaches will be tested. DBSCAN and mean shift will be used to cluster motion data based on static tracking lengths, velocity and by including object sizes to the clustering.

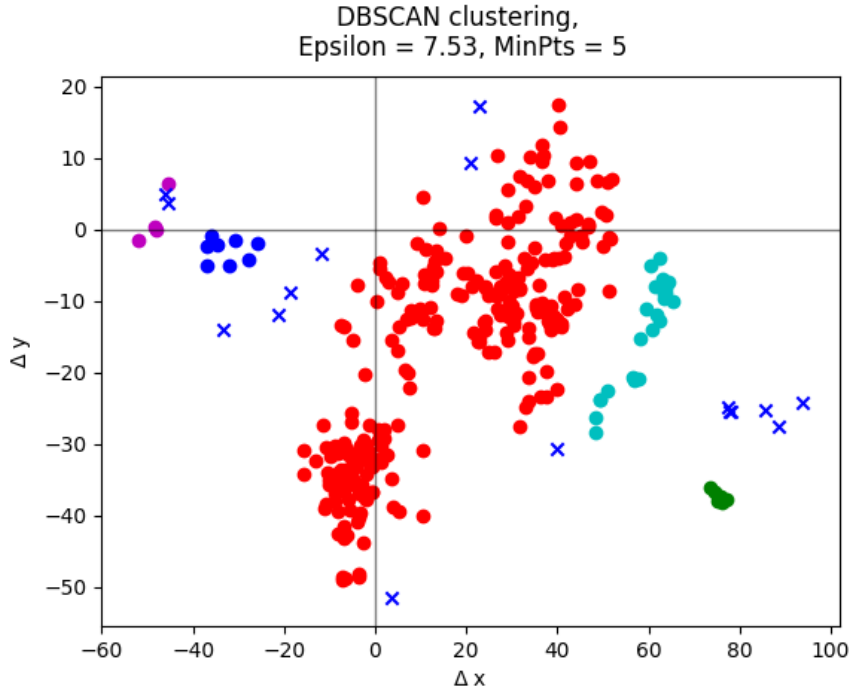
In this chapter we will present a few frames from a video showing the clustered motion patterns. Because of the size of the data only frames found irregular or particularly interesting will be presented.

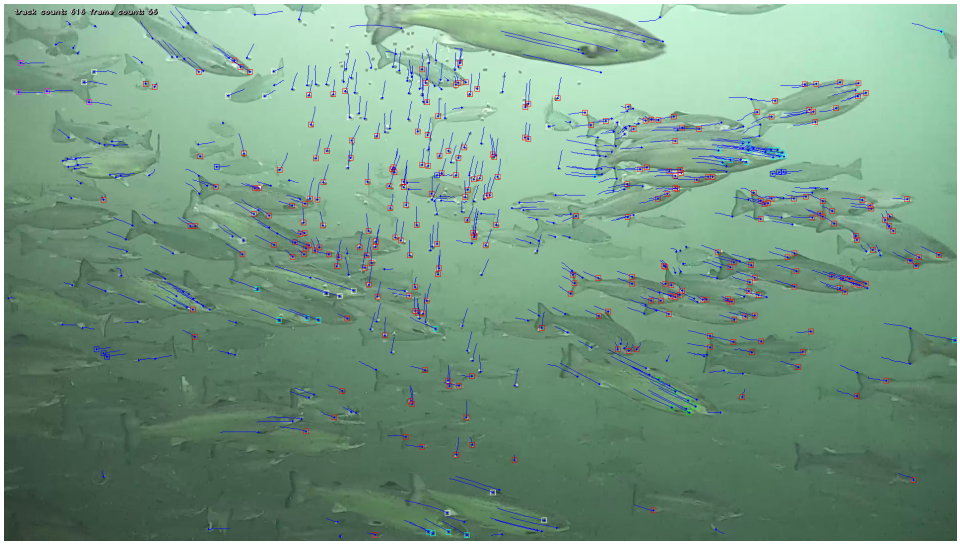
Some algorithm parameters not directly affecting the clustering, were kept static during the experiments. All functions and parameter values can be found in the appended source code and configuration file. Parameters related to the clustering are specified in the figures.

All data are visualized using a plot of the motion patterns and the image they originate from. To get a better understanding of which object that corresponds to the different motion patterns, a rectangle encapsulating the feature point is drawn, using the color representing the cluster. The following figures showing results clustered by DBSCAN, visualizes motion patterns clustered as noise using a white rectangle in the image and with an ”x” in the plots. Additionally, lines showing the tracked motion pattern of each feature is shown in the image, using blue lines, to be able to get a better understanding of the clustering.

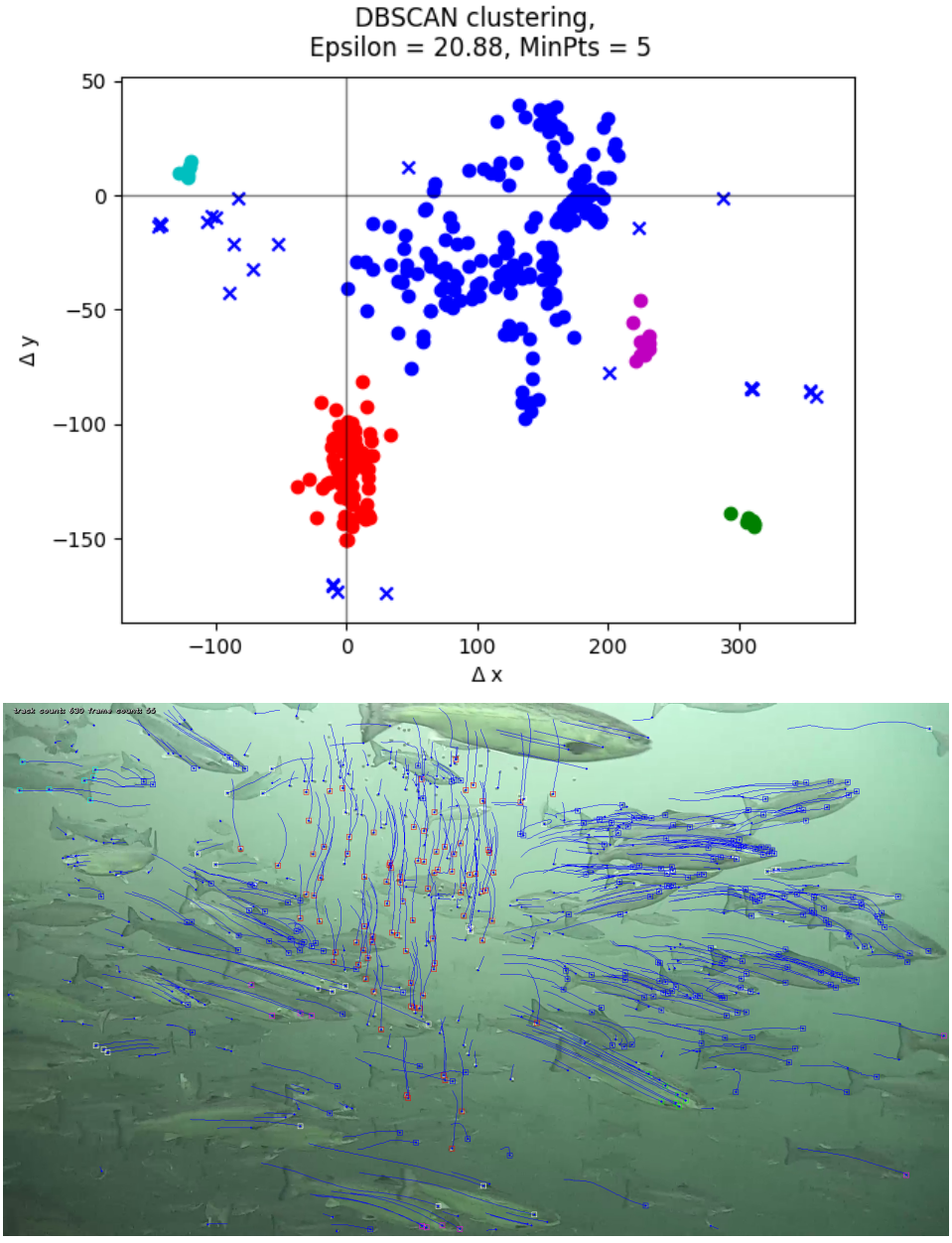
5.1 Using Static Tracking Lengths

5.1.1 DBSCAN

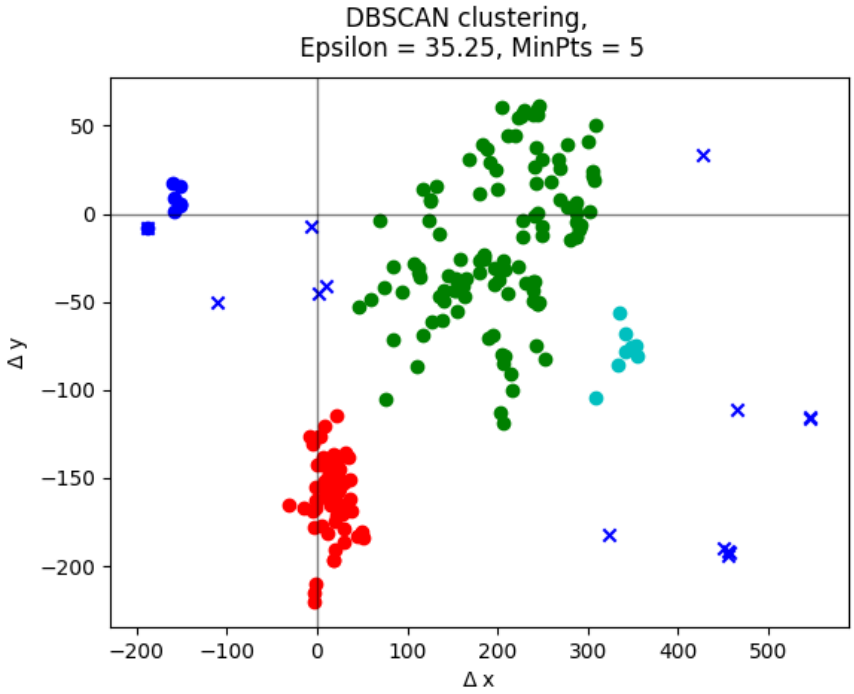


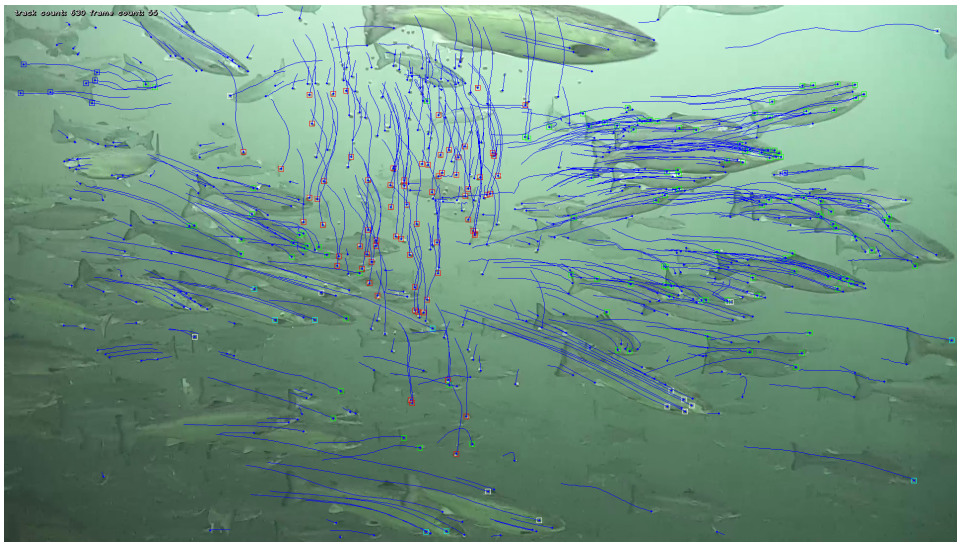


(a) DBSCAN clustering using 10 frame tracking length



(b) DBSCAN clustering using 35 frame tracking length





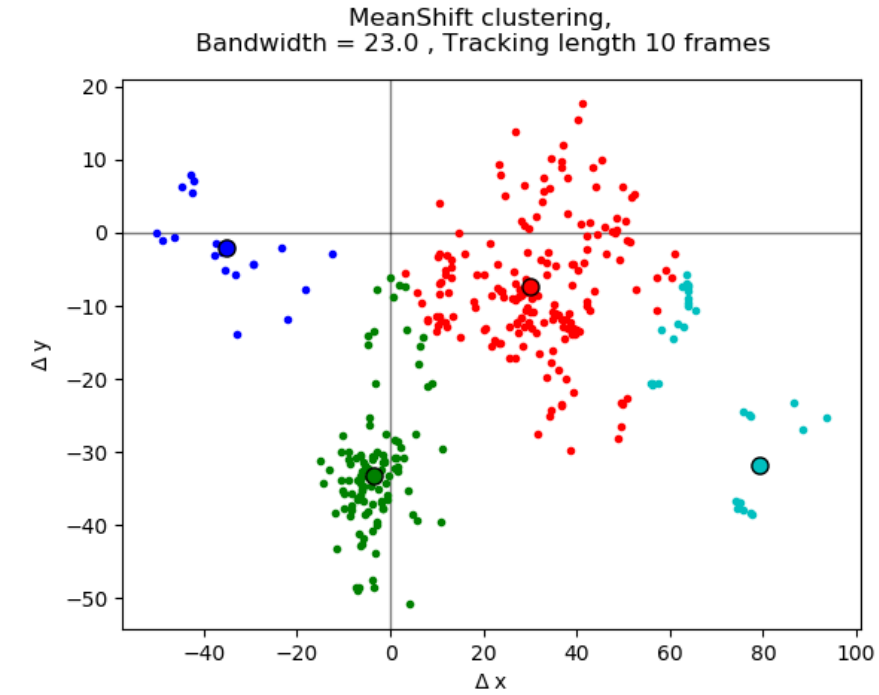
(c) DBSCAN clustering using 50 frame tracking length

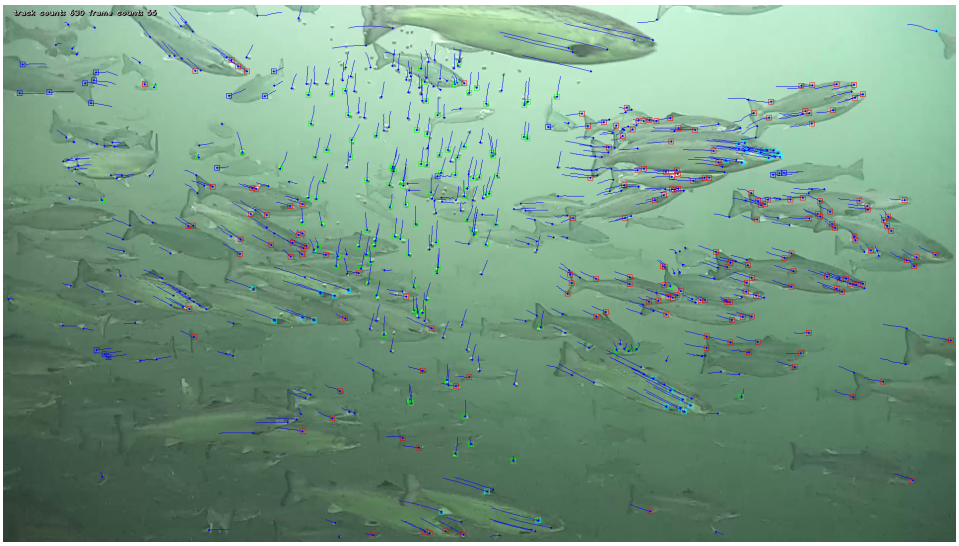
Figure 5.1: DBSCAN clustering using static 14% noise

Figure 5.1 shows clustering using DBSCAN for several tracking lengths using the heuristic presented in section 3.3.1. The amount of noise was found empirically to be $\approx 14\%$

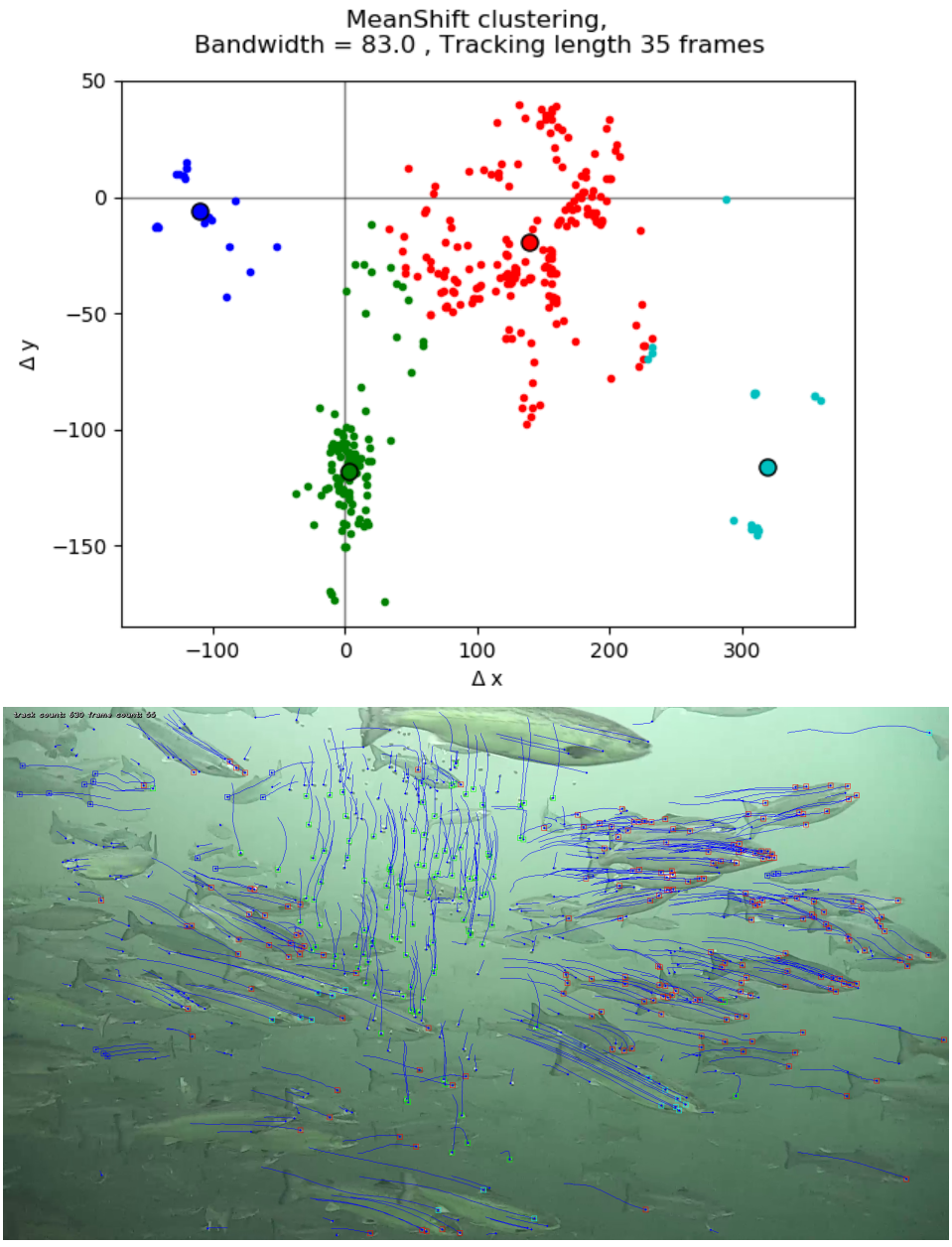
for the video used.
It can be seen that longer tracking lengths creates a larger feature space at the cost of number of features consistently tracked. A larger feature space makes it easier to separate real-world object groups into clusters, because larger inter-cluster distances allows for more inaccurate clustering parameters. Since there are no deterministic clustering parameter estimation methods, allowing more inaccuracy may lead to more consistency in the resulting classification.
One issue that has been detected using this approach is that the separability of the clusters is varying. By mainly observing the two largest clusters which corresponds to fish and pellets, visualized by red and green in figure 5.1b, one can see that the distance between the two clusters is significant. This is not always the case. By observing sub-figure 5.1a there are a few points in between the clusters that merge them together, making pellets and fish ending up in the same cluster.

5.1.2 Mean Shift

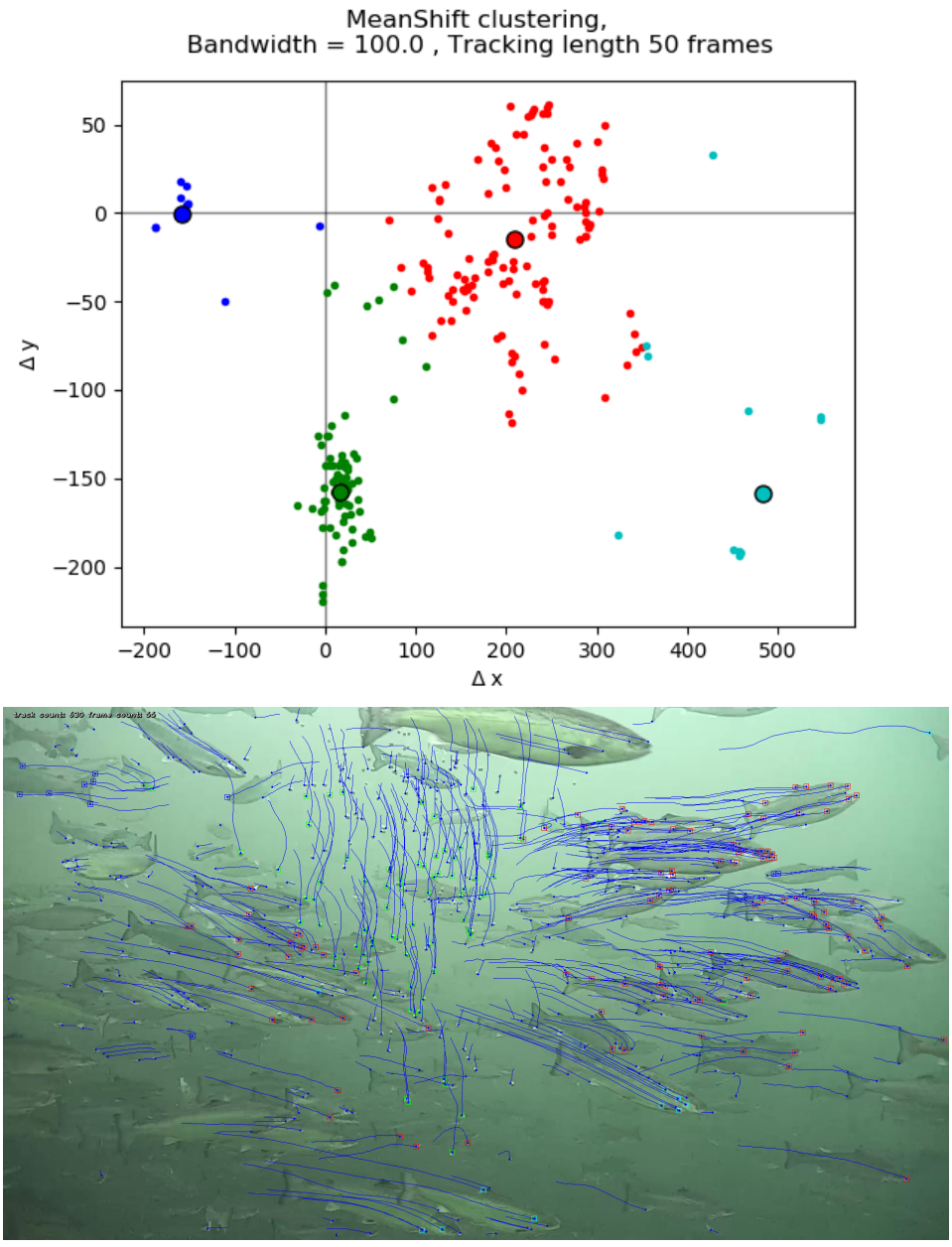




(a) Mean shift clustering using 10 frame tracking length



(b) Mean shift clustering using 35 frame tracking length



(c) Mean shift clustering using 50 frame tracking length

Figure 5.2: Mean shift clustering showing the effect of different tracking lengths

Figure 5.2 shows clustering using the mean shift algorithm for different tracking lengths using the same frames as in the DBSCAN results. The bandwidth parameter used is shown in the figure and was found using the method described in 3.3.2 by using a quantile of 30 %.
By inspecting both the plots and frames presented in figure 5.2, it can be seen that a majority of the tracked pellets and salmon are clustered separately. Notice that fish swimming from right to left are separated from the ones swimming from left to right. In addition a few salmon with a motion pattern deviating from the rest are clustered separately, visualized using a cyan color.
Notice how feature points close to the borders of the clusters are classified different from what would seem to be the closest centroid. This phenomenon can be explained by the work flow of the algorithm. During the mean shifting, in the case of overlapping shifting windows, the window containing the most points is preserved.

5.2 Using Velocity

Figure 5.1 and 5.2 shows that the separability of the clusters improves with longer tracking lengths, at the cost of number of features tracked. This introduces an information loss that is not optimal and ideally should be avoided. As a step towards improving the algorithm and avoiding the information loss introduced by having a static tracking length, we will explore the performance of using the velocity instead of relative distances as the clustering data. The velocity is found by dividing the relative movement used in section 5.1 by the number of frames the feature have been successfully tracked. Hence the data input to the clustering will be a vector describing the average velocity defined by the following expression

$$\overline{\mathbf{V}} = \frac{\mathbf{X}[n] - \mathbf{X}[0]}{n} \tag{5.1}$$

where *n* is the current tracking length and $\mathbf{X}$ is a list of stored tracked positions for a feature point in a 2D space defined by

$$\mathbf{X}[i] := [x, y], [x, y] \in \mathbf{I}, i \in [0, n] \tag{5.2}$$

where $\mathbf{I}$ is the current image evaluated from the video stream.

The expected outcome of using this method will be a denser feature space, thus requiring a more accurate parameter selection for the clustering algorithms. Further, a more rapid classification is possible as a consequence of not having to wait for the tracking of a feature to reach a certain length. This is beneficial because, by using a static threshold of tracking length, there is a risk of not detecting objects of interest that are only visible for a brief amount of time.

One concern associated with classifying all tracked features is ending up with a feature space that is inseparable. There are tendencies of having overlapping clusters in the previous models where the tracking length has been low, for instance 5.1a, resulting in merged

clusters. Mean shift tolerates overlapping clusters and would be able to separate the feature space into reasonable clusters as long as there are detectable density differences. At the same time, the number of falsely clustered points increases with decreasing cluster boundaries.

DBSCAN does not handle overlapping clusters as well. Clusters will be merged as long as there are features in the epsilon neighborhood between them. This could be avoided by under-compensating ϵ, but at the risk of ending up with more clusters than what is desirable.

5.2.1 DBSCAN

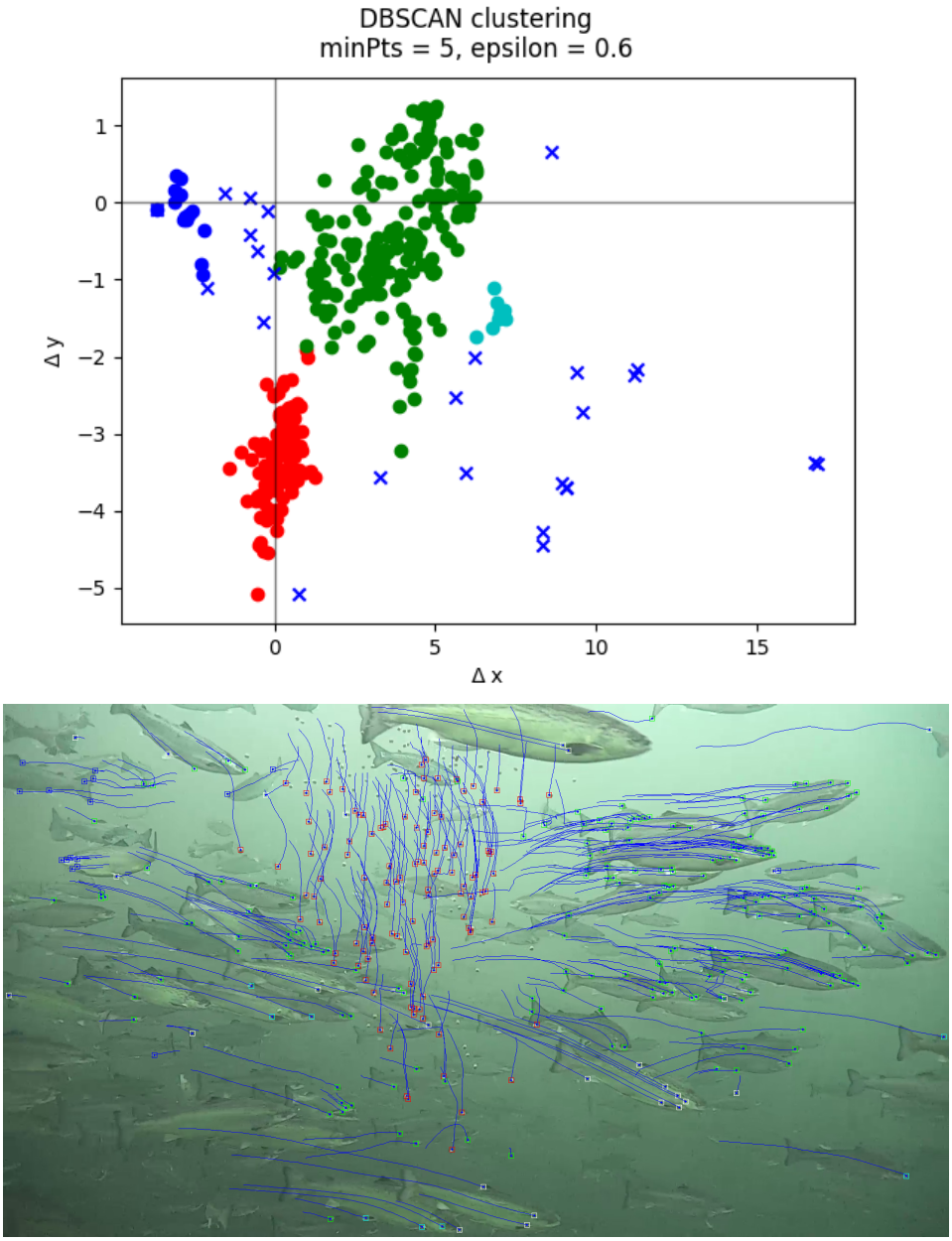


Figure 5.3: DBSCAN clustering using features regardless of tracking length with $\epsilon = 14\%$

The results shown in figure 5.3 supports the hypothesis presented earlier. Clustering succeeds using DBSCAN, but the amount of motions classified as noise is fairly high. This is a weakness of using DBSCAN on the data from the fish cage. The parameters have been adjusted aiming to primarily separate pellet and fish. These two classes, as seen in figure 5.3, are closely distributed in the feature space. Therefore a low ϵ has to be chosen to avoid any merged clusters.

5.2.2 Mean Shift

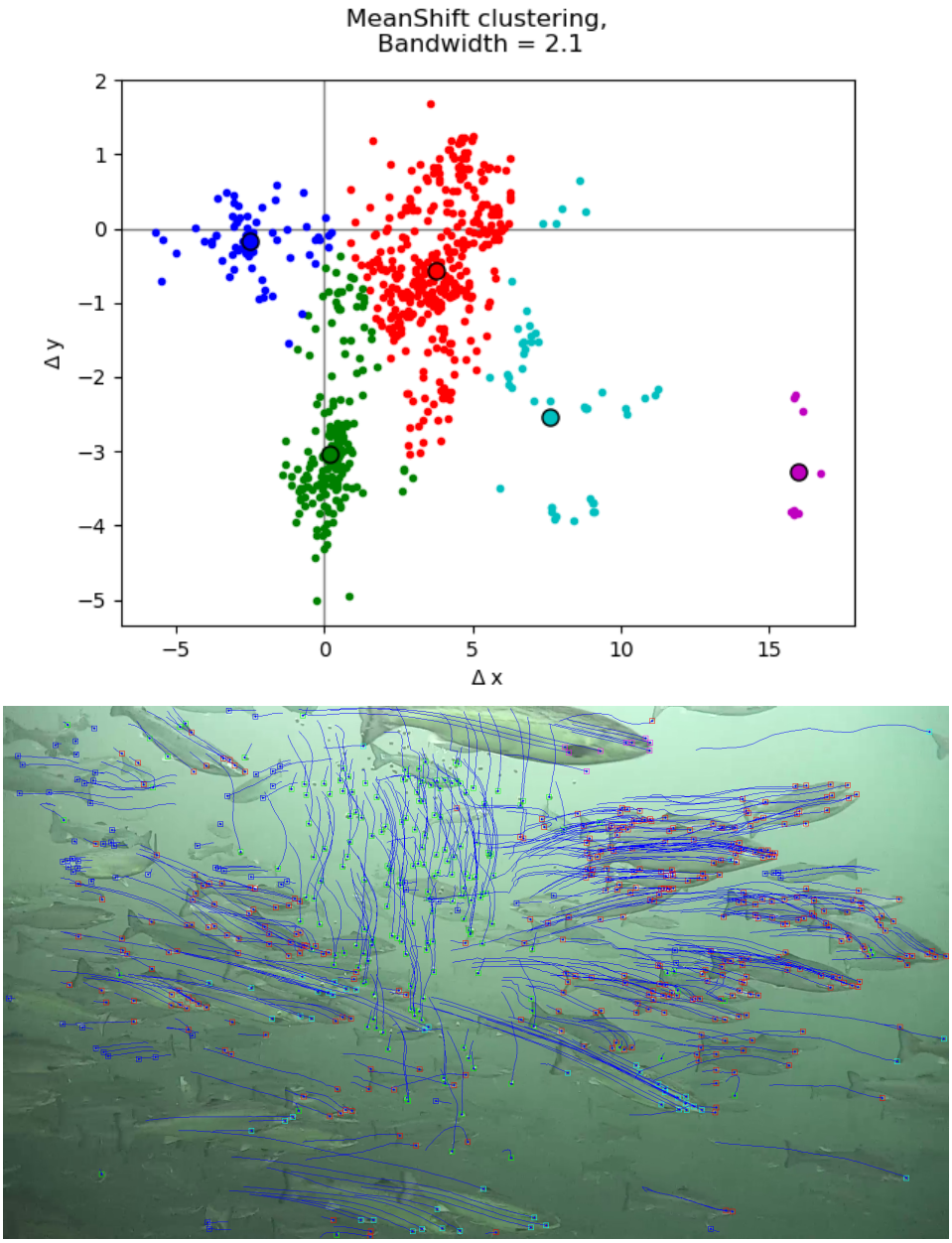


Figure 5.4: Mean shift clustering using features regardless of tracking length

The performance of mean shift does not seem to be affected by using the velocity. Comparing the results to, for instance figure 5.2b, it is clear that although the feature space is denser the clustering still performs well. Compared to the results shown in figure 5.2, the quantile of points used to estimate the bandwidth had to be lowered. Because of the closer feature space and the fairly high amount of outliers, a lower quantile have to be used to avoid the number of clusters to be too low. Figure 5.2 was created using a quantile of 30%, while a quantile of 20% was used to create the results in figure 5.4.

5.3 Including Object Sizes

Adding object sizes were done as a part of the experiments. The reason for adding object sizes into the clustering, was the hypothesis based on an observation specific for the aquaculture case, that the objects of interest have significant variations in physical size. From the tiny pellets, to the large full grown salmon. By incorporating the object size it is also theoretically possible to separate healthy fish from loser fish based on the movement and size.

Object sizes were extracted using a segmentation technique based on edge detection, as described in section 3.2. The pixel area of the objects is significantly higher than the movement from frame to frame, thus the object sizes have to be scaled down to not effect the clustering too much. For this application the object sizes were divided by 10 000 to have approximately the same scalar range as the motion patterns.

There are two major issues related to using image segmentation to extract object sizes with the proposed approach. The first is the inaccuracy of the segmentation. Figure 5.5 shows an example of a segmented frame compared to the original image. It can be seen that on fish at an angle below the camera, the contrast towards the background is so low that the fish is barely visible. The lack of contrast is also caused by larger depths and dimmer light. These parts are problematic because they are above the threshold for the edge detector, but do not capture the whole contour of the fish, as can be seen in the lower parts of sub-figure 5.5b.

A consequence of this inaccuracy could be two fish moving identically and being of the same size, but because only one complete contour is found, they get falsely clustered separately.

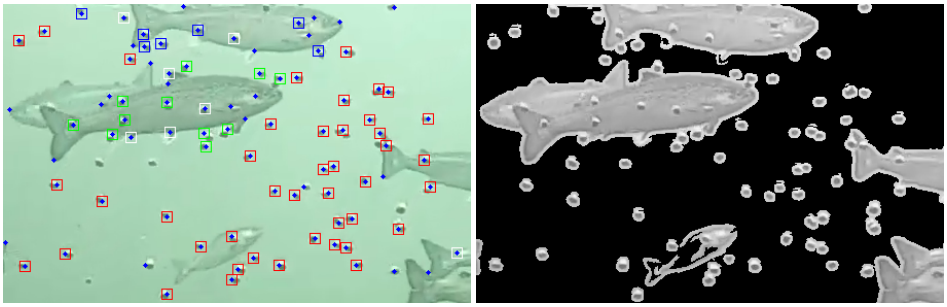
The second issue is overlapping objects. When finding contours, only the external image points found are stored. What this means, is that for contours containing both a fish and a couple of pellets, will get clustered based on the contour size of the fish. The obvious solution would be to not only store the external contours, but create a hierarchy of contours that would incorporate all overlapping contours. Such an approach would fail because of the structure of the salmon and the inaccuracies in the segmentation discussed earlier. Both fish fins, incomplete contours and possibly black dots from the side of the salmon would be included in the contour hierarchy and separating out only pellets would be impossible or inefficient without a better approach.

The following presented experiments are based on carefully picked frames, where most of the objects are positioned towards the surface or have a monotonous background and are not occluded. Therefore it was possible to study the effect of adding object sizes to the clustering even though the segmentation approach is not suited for automatic segmentation of every objects in the image.

(a) Original frame **(b)** Segmented frame

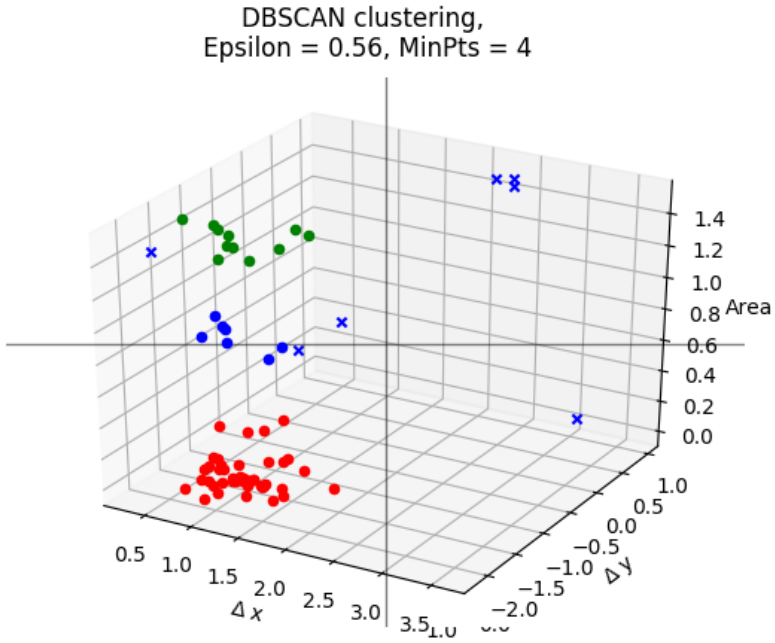
Figure 5.5: Segmented frame compared to the original colorized frame

5.3.1 DBSCAN



(a) Original frame visualized with classifications

(b) Segmented frame

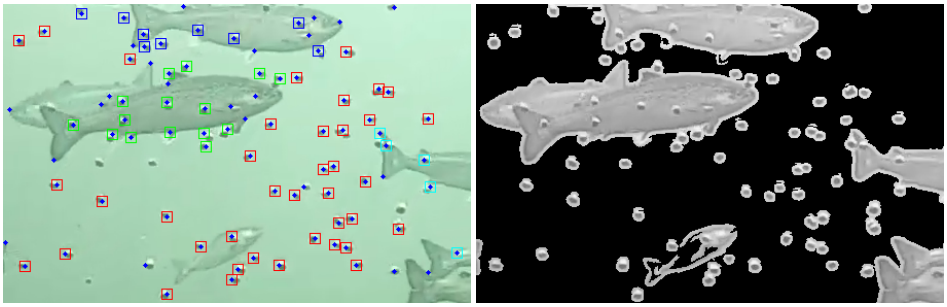


(c) Plot of feature space clustered using DBSCAN

Figure 5.6: DBSCAN clustering using features regardless of tracking length including object size

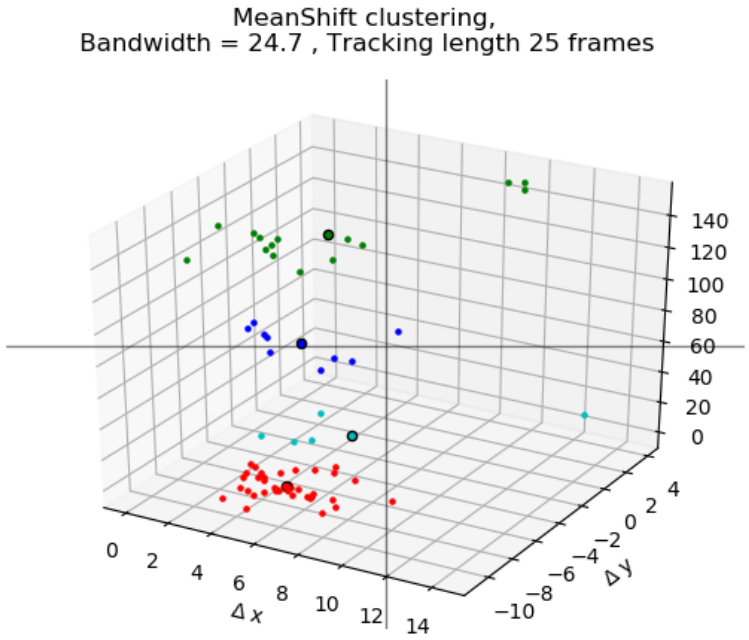
Figure 5.6 shows clustering using DBSCAN including both motion patterns and object sizes. It can be seen that the effect of including sizes accomplished what was proposed in the hypothesis. Sub-figure 5.6c shows how the added area creates a larger feature space and contribute to creating better separability of the clusters.

5.3.2 Mean Shift



(a) Original frame visualized with classifications

(b) Segmented frame



(c) Plot of feature space clustered using mean shift

Figure 5.7: Mean shift clustering using features regardless of tracking length including object size

Figure 5.7 shows clustering using mean shift including both motion patterns and object size. The same effects described for DBSCAN also affects the mean shift clustering.

Chapter 6

Results

The results from clustering motion patterns will be evaluated by the ability to separate real world object classes for each subsequent frame in a video. These results give feedback on several details of the algorithm. The main elements are:

- The ability to detect good features and consistently track them as long as there are no occlusion of the feature

- The ability to dynamically adjust clustering parameters based on heuristics described in section 3.3.1

Based on the experiments presented in section 5, some conclusions of which approach is better in certain situation can be drawn.
When dealing with a dense feature space and the continuity of separating clusters is important, DBSCAN will fall short for mean shift unless the parameter selection for DBSCAN is accurate. As shown in section 3.3, the accuracy of the chosen ϵ is important to avoid cluster merges. In the experiments it was shown that by using a static tracking length of features as input to the clustering, several features tracked got discarded because of inconsistency before being clustered. This introduced a possible information loss that is not ideal. At the same time, the advantage of using tracking lengths of a constant length is the ability to control the size of the feature space to some extent.

Mean shift is a centroid based clustering algorithm, hence it will not have issues with cluster merging as DBSCAN does, as long as the Parzen window is not too large. Combining this ability with using the velocity as clustering data, the most optimal information extraction can be obtained. By choosing this approach, the feature space is denser and the risk of classification errors is higher, but the number of clustered features will also increase. When using mean shift, the increased number of features exceeds the amount of misclassifications, hence making this approach preferable.

By including object sizes in the clustering, better separability of the clusters was achieved. The proposed segmentation algorithm was not accurate enough to be used for automatic object size estimation of all objects in an image. By carefully selecting frames, some data were gathered which showed promising results towards improving the clustering compared to merely using motion patterns.

Chapter 7

Discussion

7.1 Real-Time Performance

One concern about the proposed algorithm is the real-time performance. Ideally it should be able to run as a back-end application processing frames from a live video stream. For this to be a reality, there are some requirements for the run-time of each cycle of the algorithm. As discussed in the pre-project [21], using the optical flow tracking sets a lower limit on the frame rates that can be used, because of the small movement constraint that optical flow is based on. Low frame rates leads to large apparent motions, because objects from one image to the next have time to move significantly. Obviously this is dependent of what is being classified. Slow moving objects can be consistently tracked with a lower frame rate than a faster moving object. Generally the frame rate is not the crucial factor, but the displacement of the tracked feature from one frame to the next have to be sufficiently small.

This leads into the never-ending discussion of speed versus accuracy. In computer vision one often have to make the trade-off between speed and accuracy, which also applies to the proposed algorithm. Using the same parameters which were used to produce the results in section 6, the average frame rate was found to be between 5 and 10 frames per second using a regular desktop computer, depending on the consistency of the tracking. With increasing numbers of discarded tracked features, a more frequent feature detection is needed, which slows down the run-time. The videos provided by Sealab AS were recorded using 25 frames per second which will be the threshold for accepting the algorithm as real-time or not. Hence some tuning options have to be discussed for ways to improve the frame rate of the algorithm.

The first parameter to discuss is the resolution of the image used in the processing. The input is HD video with a resolution of 1920x1080 pixels. By resizing the input image, the next processing steps can be made much faster. An increased frame rate does not come without a cost. By decreasing the image resolution the feature space gets narrower which

leads to more falsely classified features close to the cluster borders as discussed in section 5.1.2.

The second parameter affecting the run-time is the maximum number of feature to be detected. The issue with lowering the number of tracked features is that it will lead to fewer classified objects.

In the proposed algorithm, clustering is performed for every frame which might not be necessary based on the application.
One approach could be to cluster data using a given frame interval. Related to the videos used in this thesis, a clustering every second could have given satisfying information of what is seen in the image until the next second. Assuming a frame rate of 25 frames per second would introduce a significantly lower amount of processing needed.
Another way to lower the classification rate is to assume consistency in the classification. By assuming all classifications of tracked features are correct and features will belong to the same class as long as they are visible, the clustering can be limited to only run when new tracked features reach a specified threshold length.

7.2 Camera Placement

Since the tracking is based on optical flow, which calculates the apparent motion, some assumptions have been made about the camera placement. In the case where the aforementioned approach of creating a model based on the output of the algorithm, the user should be aware that a moving camera will create a bias in the output feature space. For a more thorough discussion of the camera positions effect on the tracking we refer to [21].

Another point to make about the camera placement is the distance from the camera to the objects of interest. By including as many objects as possible in the image at all times, by observing the fish from distance is advised. By including as many objects as possible into the image, the number of feature to detect increases, which in return creates more data to base the clustering on. Using more data as input to the clustering gives more reliable clusters, as the centroids and dense areas in the feature space becomes more clear.

Chapter 8

Conclusion and Further Work

Through this thesis we have shown how an unsupervised learning algorithm can be used to classify motion patterns of objects in a fish farm. The algorithm performs well as long as the number of samples used as input to the clustering is significant. As the clustering is based on defining a cluster based on regions of higher density, a necessity is to have several objects in the image. Therefore the proposed algorithm is not suitable to detect a single pellet or a single fish in an image, but rather classify multiple objects based on motion similarities.
Two clustering algorithms were used in the experiments and were discovered to have different characteristics. Mean shift tolerates more inaccuracies in the parameter selection than DBSCAN and is less prone to cluster merging. DBSCAN performed better at completely separating clusters.

The edge detection segmentation used proved to not be accurate enough for the purpose of obtaining information about the object size of all visible objects in the image. The segmentation suffered from incomplete contours and an inability to separate overlapping objects. Some data were possible to gather by carefully selecting image areas, with small amounts of occlusion and backgrounds of high contrast towards the objects. Based on this data a clustering was performed using both motion patterns and object sizes. Although the amount of test data was not adequate to draw any conclusions, the results was promising towards contributing to more robust clustering, because of higher rates of separability, especially between fish and pellets.

The output of the algorithm consists of motion data and which cluster each motion pattern belongs to. This is information that is useful on it's own, but does not necessarily give any information about what kind of objects it corresponds to. In the case where a labeled output is of interest, a model has to be trained for the clustered output data. A model able to label the clusters will be fairly simple, if it is based on the mean value of each cluster, but have to be adjusted to the environment operated in.

By using the algorithm presented in this thesis, several use cases are possible as long as motion patterns can be used as a feature to separate the process. Example related to the aquaculture case studied in this thesis could be:

- By storing motion data over time, studies related to predicting fish health based on their movement could be achieved
- Appetite monitoring could be achieved by recording the feeding process and analyze how far the pellets drop before they are eaten
- If supervised algorithms are of preference, image annotation could be done using the proposed unsupervised approach since it does not require a pre-trained model and is able to separate fish and pellets

Further work could be directed towards finding a suitable segmentation algorithm to be able to both have a reliable object size to base the clustering on, as well as being able to reduce the amount of feature points tracked, to an amount related to the number of tracked objects. Additionally, using different algorithms for feature detection and tracking to improve the run-time of the algorithm are possible. The proposed algorithm is computationally heavy, which is problematic as the computational power of small hardware suited for placing in underwater cameras is still limited. Remote computation is possible, but having on-board hardware to do image analysis is preferred as delays associated with sending data long distances should ideally be avoided.

Bibliography

[1] Norwegian Environment Agency, "Vurdering av nye tekniske løsninger for å redusere utslippene fra fiskeoppdrett i sjø," 2011.

[2] B. Morris and M. Trivedi, "Unsupervised learning of motion patterns of rear surrounding vehicles," 12 2009.

[3] D. Klostermann, A. Osep, J. Stückler, and B. Leibe, "Unsupervised learning of shape-motion patterns for objects in urban street scenes," 01 2016.

[4] T. Nawaz, A. Cavallaro, and B. Rinner, "Trajectory clustering for motion pattern extraction in aerial videos," in *2014 IEEE International Conference on Image Processing (ICIP)*, pp. 1016–1020, Oct 2014.

[5] J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *In 5-th Berkeley Symposium on Mathematical Statistics and Probability*, pp. 281–297, 1967.

[6] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[7] D. Comaniciu and P. Meer, "Mean shift: A robust approach toward feature space analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 603–619, May 2002.

[8] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pp. 226–231, AAAI Press, 1996.

[9] M. Ankerst, M. M. Breunig, H. peter Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," pp. 49–60, ACM Press, 1999.

[10] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016.

[11] V. Badrinarayanan, A. Kendall, and R. Cipolla, "Segnet: A deep convolutional encoder-decoder architecture for image segmentation," *CoRR*, vol. abs/1511.00561, 2015.

[12] I. Sobel, "An isotropic 3x3 image gradient operator," 02 2014.

[13] J. Gan and Y. Tao, "Dbscan revisited: Mis-claim, un-fixability, and approximation," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, (New York, NY, USA), pp. 519–530, ACM, 2015.

[14] "Sigmod, special interest group on management of data." `https://sigmod.org/`, Accessed 31. May 2018.

[15] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Trans. Database Syst.*, vol. 42, pp. 19:1–19:21, July 2017.

[16] E. Parzen, "On estimation of a probability density function and mode," *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. pp. 1065–1076, 1962.

[17] OpenCV. `https://docs.opencv.org/3.4.1/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a`, Accessed 21. May 2018.

[18] OpenCV. `https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga2c759ed9f497d4a618048a2f56dc97f1`, Accessed 21. May 2018.

[19] S. Suzuki and K. be, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32 – 46, 1985.

[20] R. A. Adams and C. Essex, *Calculus 2*. Pearson, 8 ed., 2013.

[21] Ø. R. Karlstad, A. Stahl, and C. Schellewald, "Motion estimation of objects in high quality underwater video from fish cages," 2017.

[22] S. J. Russell and P. Norvig, *Artificial Intelligence, A Modern Approach*. Pearson, 2 ed., 2016.

[23] "Python software foundation." `https://www.python.org/`, Accessed 31. May 2018.

[24] "Ctypes - a foreign function library for python." `https://docs.python.org/3/library/ctypes.html`, Accessed 31. May 2018.

[25] "Json (javascript object notation)." `https://www.json.org/`, Accessed 31. May 2018.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[27] "Pyclustering." `https://github.com/annoviko/pyclustering`, Accessed 31. May 2018.

[28] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.