UNIVERSITY OF OSLO
Department of Physics

Tracking and classification of divers in sonar images

Martin August Brinkmann

November 8, 2010

Abstract

The need for harbor protection systems have increased over the last decade. One vital component of harbor surveillance are the use of sonar to detect underwater threats such as divers. In order to detect such threats, algorithms for detection, tracking and a robust classification of underwater objects is needed.

This thesis uses known methods to detect, track and classify objects recorded from real sonar data. A temporal cell averaging filter is used to detect objects in sonar images and a tracking method based on the Probabilistic Data Association Filter (PDAF) is used to track an object over time.

A set of object features, derived from a sequence of sonar images, is used to compute a set of static and temporal features. The features tested in the thesis are compared to each other to measure their ability to distinguish divers from marine life such as seals and dolphins. A linear discriminant function is used as a classifier.

Acknowledgements

First of all I would like to thank my supervisor at Kongsberg Defence Systems, Endre Marken for all the good input given during the work of this thesis. Your thoughts and inputs have been a guideline throughout this entire work.

A special thanks to my co-supervisor at the University of Oslo, Andreas Austeng. Thanks for all the support during the last two years of completing my masters degree. Thank you for putting me in contact with Kongsberg Denfence Systems and making this thesis possible.

A would also like to thank my fellow student Jo Inge Buskenes. You have been my mentor in the world of LaTeX, and been a pleasure to study with.

My wife Gina, thank you so much for putting up with me during this stressful period. Without your motivation and understanding, the process of completing this thesis would have been overwhelming. You are truly an inspiration for me.

Martin A. Brinkmann

Oslo, September 2010

Contents

Chapter 1

Introduction

1.1 Motivation

The need for harbor protection systems has emerged as the threat of terrorist attacks against ships and harbors has increased over the last decade. Over the history of time, sea transport has been the largest carrier of freight, and ships play an important role in a modern society. Harbors are large facilities containing large amount of people and goods. They are often situated in areas with dense population and with a lot of passengers and cargo circulation, hence the harbors are potentially vulnerable for terrorist attacks.

On October 12, 2000, the American naval destroyer USS Cole entered the harbor of Aden, Yemen for refueling. During the refueling a small boat approached and placed itself alongside the destroyer. Shortly after the small boat exploded making a 15 times 15 meter wide hole in the destroyers side. 17 of the ship's crew where killed and 39 injured. This attack showed how vulnerable vessels are while docked. In 1995, Sri Lankan Navy lost a number of vessels due to suicide divers from the Tamil Tigers, and Hamas has also used divers to attack Israeli installations.

In a harbor protection system the goal is to have control of every object that may be a threat to the facility both over and under the water level. The use of sonar are one method of monitoring activity under the surface of the sea. In a normal port there are a lot of activity, and it can be a challenging and time consuming job for a sonar operator to separate potential threats and normal activity.

By finding an effective way to detect and correctly classify objects and distinguish potential threats from normal activity, the workload of the sonar operator will be reduced and the reaction time for an underwater threat will be increased.

1.2 The Problem

This thesis evaluates known features to see if they are suitable for distinction and classification of divers and marine life in sonar images. The features tested consists of shape descriptors and echo strength measurements. Features are extracted from a sequence of sonar pings resulting in a set of features describing an object of interest. These *static* features from each ping are combined to characterize an object over time. This results in a new set of *temporal* features. Both static and temporal features are used to train a *classifier* to classify two classes, diver or marine life.

To extract these features, the object of interest must first be detected in the sonar image. To collect a sample set of each feature, the object has to be observed over a period of time. In order to do this a tracking algorithm has been implemented. To separate the object in the sonar image from the background, a segmentation algorithm has also been implemented. The static features are extracted from the segmented object.

At the end a simple *linear discriminant function* has been used to classify the objects, and a comparison of the different features are made.

The thesis has used real sonar data of diver and marine life. The sonar data are provided by Kongsberg Defence Systems.

1.3 The Process

The purpose of this thesis was to find features that could correctly classify objects of interest in a harbor environment and find a robust classification of such targets.

In order to extract the features needed for the classification a detection and tracking method was needed. This became as an important and consuming task as the actual classification.

The thesis describes two types of detection methods. The use of a Constant False Alarm Rate (CFAR) filter and a temporal cell averaging filter. Both methods where tested and the temporal cell averaging filter proved to be most accurate and therefor implemented.

Two types of tracking algorithms have been implemented and tested; the Nearest Neighbor Kalman filter and the Probabilistic Data Association filter. Both methods where able to track objects over multiple pings, but the PDAF method appeared to be more robust to noise and clutter.

The thesis tests known features that has proved good classification ability in similar tasks [3, 4, 12]. The features consists of a set of static features and a set of temporal features.

A simple linear classifier have been used to test the features. A selection of the most efficient features have been found, based on the features individual ability to correctly classify between the two classes.

The results show that on the given training set, the temporal features are able to classify the training objects with 0% error rate while the static features gives an error rate of 1.2%.

Chapter 2

Short introduction to sonar and divers

This section will give a short introduction to the concept of sonar and different type of divers and marine life that can be expected in an harbor environment. The effect different diving equipment can have on the sonars ability to detect divers will be discussed, as well ad the effect sonar can have on marine mammals such as dolphin and whales.

2.1 Sonar

Sound Navigation and Ranging, commonly referred to as SONAR is a technique that uses sound propagation in water to navigate or detect various objects. There are mainly two types of sonar, passive sonar and active sonars. The main difference between the two types is while the passive sonar listens for sound produced by other objects like vessels, marine life etc, active sonars emit sound pulses and listens for the echo of the pulse. The use of passive sonar are most often used by submarines that cannot emit any active sound pulses due to the risk of detection.

Active sonars have both transmitters and receivers. Each sound pulse the sonar emits is called a *ping* and the reflected pulse is called an *echo*. In harbor protection system both types can be used.

2.2 Different types of divers

One of the threats you can expect against ships and harbor infrastructure is attack from divers. Since the second world war military divers has been used to carry out covert operations like sabotage missions and intelligence gathering. By swimming under water equipped with SCUBA[1] gear, a person can easily enter an unprotected port facility unnoticed and do a lot of damage.

There are different types off SCUBA equipment, Open, semi-closed an closed circuit system. The first type is widely used by civilian sport divers, the two latter are mainly used by professional and military divers.

2.2.1 Open circuit

This is the most common type of SCUBA set. It consist of an steel or aluminum tank filled with compressed air or other compressed gas such as *nitrox*[2], *heliox*[3] and *trimix*[4], a first stage that reduces the air pressure and a second stage at the mouthpiece. The diver breathes compressed air and exhales the used air directly out in the water giving a large trail of bubbles behind.

2.2.2 Semi-closed circuit

Less common are the closed or semi-closed systems called rebreathers. This type of SCUBA gear is more technical and complicated than the open circuit. A person absorbs approximately 5% of the oxygen he breathes and produces carbon-dioxide instead. Instead of letting the exhaled air from the diver which still contains oxygen, directly out in the water, the rebreather circulate the used air by removing the carbon-dioxide and adding new gas. As the breathing gas circulates the diver uses more and more of the oxygen decreasing the precentage of oxygen and increasing the precentage of nitrogen. To keep the oxygen level steady, a constant flow of new breathing gas is added to the system. To prevent a build up of gas in the system some gas is released into the water through a exhaust valve. This system lets only out a small amount of bubbles. Types of breathing gases commonly used in rebreathers are nitrox and heliox.

[1]SCUBA stands for Self Contained Underwater Breathing Apparatus

[2]Normal air consists of approximately 78% nitrogen, 21% oxygen and 1% other gases. Nitrox refers to a gas containing nitrogen and oxygen but unlike normal air it has usually a lower presentage of nitrogen and higher precentage of oxygen. This will give a diver a reduced risk of decompression sickness but will limit the depth that gas can be used.

[3]Breathing gas consisting of helium and oxygen.

[4]Breathing gas consisiting of helium, nitrogen and oxygen.

2.2.3 Closed circuit

The closed circuit system is a rebreather that uses 100% oxygen as breathing gas. With no nitrogen in the breating gas there is no need to have a constant flow of new gas added to the system. Instead new oxygen is added through a demand valve as the circulation oxygen is absorbed by the diver. Having only oxygen as the breathing gas enables the system to be fully closed, with no exess gas leaving the system. This type of breathing system is primarily used by military divers for covert operations.

2.2.4 Effect of different systems in sonar

The different systems will give a different effect in the sonar image. Air bubbles gives a good source for echo. A diver using an open SCUBA system will give a far larger echo than a diver using a closed or semi closed system. A diver using a closed or semi closed system may therefor be more difficult to detect than one with an open system. The sonar data used in this thesis are of a diver using a semi closed rebreather system.

2.3 Marine Life

In harbor environments there will always be a element of marine life such as fish, seals, dolphins and whales. There are two reasons why detection and classification of such life are desirable. Seals and dolphins can often be mistaken for a human diver. Both size and often swim pattern can be similar to that of a human diver. In order to prevent false alarms a harbor protection system must be able to distinguish a diver from a seal or dolphin.

Marine mammals such as dolphins and whales rely on their hearing to navigate and communicate. Sonar sound waves can do serious damage to marine mammals [1]. Feeding and other vital behavior can also be disturbed by the presence of a sonar. Many countries have therefore imposed regulations for use of SONAR in areas where mammals are present. The ability to detect and classify this type of marine life enables the use of sonar in regulated areas as the sonar can reduce the signal strength or be shut down in the presence of mammals.

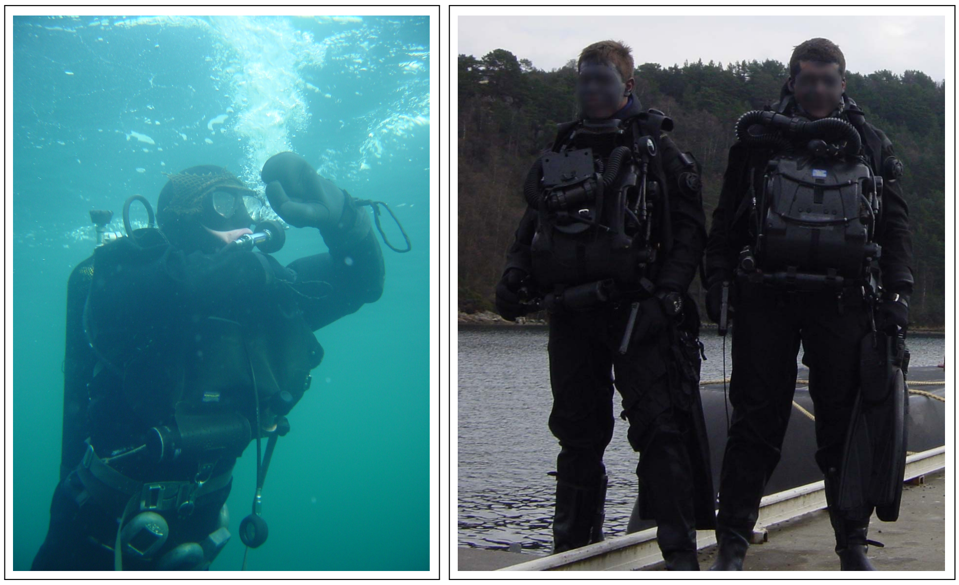

Figure 2.1: **Left:** Diver with open circuit system. **Right:** Two divers with closed circuit Siva 24 rebreathers.

2.4 Summary

A brief explanation of the concept of sonar has been presented along with a short description of the different types of divers and which effect different diving equipment may have for sonar detection.

The negative effect sonar may have on marine mammals such as dolphins and whales gives motivation for the ability to detect such mammals in order to shut down the sonar if they are present.

Chapter 3

Theoretical Background

The classification system described in this thesis consists of the following blocks,

- Target detection
- Tracking
- Feature extraction
- Feature selection
- Classification

This section will give a theoretical description of the different functional blocks of the classification system. A block schematic of the overall system can be seen in Fig. 3.1. The sonar data used in this thesis has first been matched filtered before further processing.

3.1 Object Detection

In order to extract features from an object, the object must first be detected in the sonar image. A sonar image consists of a matrix of cells, just like pixels in an ordinary picture. A detection of a possible object are hereafter referred to as an *event*. Objects in sonar images are often hard to detect due to signal noise, clutter and interference. In order to detect events in a sonar image a filtering process is needed. A common approach is to find an average echo strength for each cell either by temporal or spatial averaging.

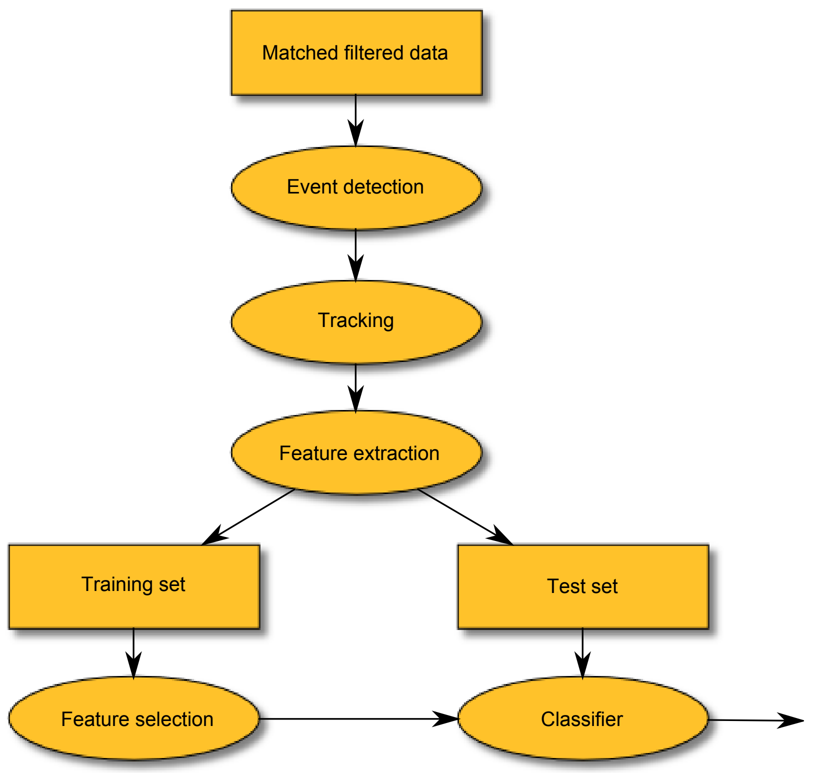

Figure 3.1: Overall system

3.1.1 Constant False Alarm Rate

The Constant False Alarm Rate (CFAR) filter is used to distinguish measurements from valid objects to background noise. Target echoes can often be buried in signal noise or clutter which has unknown power. A threshold is used to detect object measurements as events. The CFAR filter sets the threshold adaptively based on the local information of the total noise power. Based on the average noise in neighboring cells, the CFAR filter sets the local threshold for each cell. To avoid that the amplitude from the cell under test influencing the estimation of the average noise, a band of guard cells are often set up around the cell[9, 11]. Using this method to adaptive find the local threshold for each cell, the filter that a constant false alarm rate is kept.

3.1.1.1 The Algorithm

Assuming the intensity of the noise is exponentially distributed [11], the background noise for one cell is estimated by taking the average amplitude of n surrounding

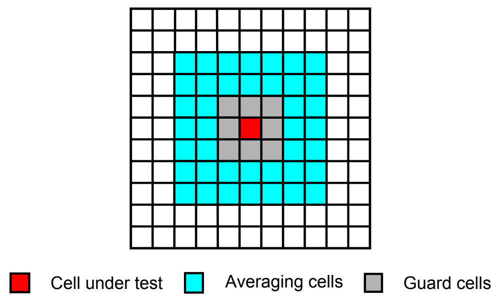

Figure 3.2: CFAR: The background noise is estimated by the averaging cells. A band of guard cells is set around the cell under test to prevent self masking.

cells

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} A_i. \tag{3.1}$$

If the amplitude of the cell under test (X) is larger than the estimate ($\hat{\mu}$), the cell will be defined as a valid event

$$\frac{X}{\hat{\mu}} > \tau, \tag{3.2}$$

where τ is a user defined constant scale factor. This scale factor is used to control false alarm rate of the system. From [7] the probability of a false alarm in a cell is given by

$$P_{fa} = \frac{1}{\left(1 + \frac{\tau}{n}\right)^n} \tag{3.3}$$

where n is the number of cells used to estimate $\hat{\mu}$. Given a false alarm rate P_{fa}, the scale factor can be calculated as

$$\tau = n\left(P_{fa}^{-1/n} - 1\right). \tag{3.4}$$

Figure 3.2 shows a CFAR filter mask. The cells surrounding the cell under test is used as guard cells separating the cell from the cells used for estimating the average noise.

3.1.2 Temporal Cell Averaging

Another approach to to cell averaging filter is to do a temporal averaging instead of spatial averaging. In this method the mean amplitude of each cell is found by averaging the cell amplitude from every ping up to k.

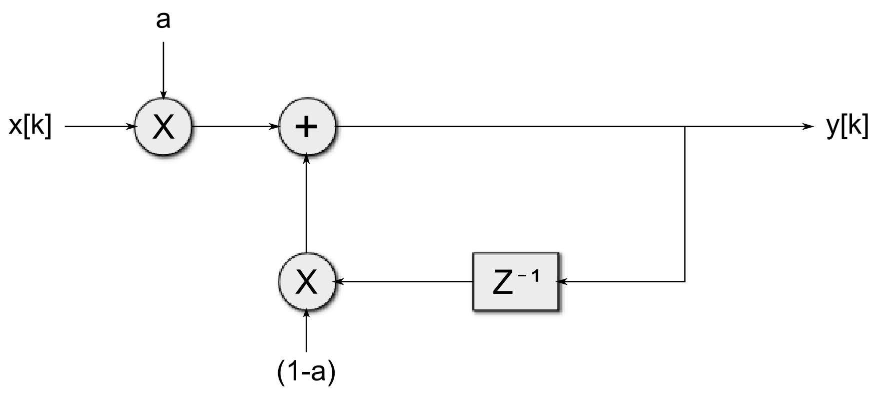

Figure 3.3: Block diagram of temporal averaging algorithm. $x[k]$ is the cell amplitude from the current ping k. The output $y[k]$ is the temporal average estimate $\hat{\mu}_k$.

The following algorithm is used in the `pinPingFilter` function described in Section 4.1. This approach has been used at Kongsberg Defence Systems (KDS) to find valid events in matched filtered sonar data, and has shown good results also in this thesis.

3.1.2.1 The algorithm

The mean amplitude of each cell at ping k is found by

$$\hat{\mu}_k = (1-a)\hat{\mu}_{k-1} + aX_k, \tag{3.5}$$

where a is a constant scale factor between zero and one and $\hat{\mu}_{k-1}$ is the average estimate for the cell at ping $k-1$. Looking at the ratio between the mean amplitude of the cell and the amplitude of the cell in ping k, if the amplitude is larger than the estimate, the cell will be defined as a valid event,

$$\frac{X_k}{\hat{\mu}_k} > \tau, \tag{3.6}$$

where τ is a constant scale factor. Figure 3.3 shows a block diagram of the temporal averaging algorithm.

3.2 Tracking

Tracking consists of estimating the current state(i.e location) of a object, based on uncertain measurements and its former state [10]. Having the last state of a object, the expected new state is an estimate based on the history of the object and associated measurements (*events*). False alarm and clutter detections may be present and give origin to non-object events. These events may not be easily distinguishable from the true object and some type of data association is needed. Data association is to determine from which object a certain event originated from.

3.2.1 Model of tracking

The goal for linear tracking is to predict the next state of an object based on previous events [11]. The standard discrete linear model in tracking is

$$x_{k+1} = Fx_k + v_k, \tag{3.7}$$

where x_k is the object state vector at time k, F is the transition matrix and v_k is the process noise. The object state vector consist of the Cartesian coordinates in two dimensions and their rate of chance between each ping

$$F = \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad x = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \end{bmatrix}. \tag{3.8}$$

The only information available from the sonar image is the position of the measurements. The relation between the measurement vector z_k and the state vector is

$$z_k = Hx_k + w_k. \tag{3.9}$$

where H is measurement matrix,

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \tag{3.10}$$

and w_k is the measurement noise. If there is only one event at each ping at time k to relate to a track, this model would be optimal. However there are often multiple events for each ping and they form a set of events

$$Z_k = \{z_k(1), z_k(2), \cdots, z_k(m)\}. \tag{3.11}$$

To determine which of these events that originates from the track, a form for data association is needed. Two common methods are the Nearest Neighbor Kalman Filter (NNKF) and the Probabilistic Data Association Filter (PDAF)[10, 2].

3.2.2 Nearest Neighbor Kalman Filter

In this method the event that is nearest to the estimated measurement is chosen for updating the track. If there exists multiple tracks, a single event can only be associated to one track. If there are no valid events in close proximity to the estimated measurement, the estimate is propagated to the next ping. This method is fairly easy to implement and do not demand a lot of computational time. However the possibility of losing the track and adapt false events are moderately high [10].

3.2.2.1 The algorithm

The algorithm [10] predicts the next state of a track by using a difference equation

$$\tilde{x}_{k|k-1} = F\hat{x}_{k-1|k-1}, \tag{3.12}$$

where $\hat{x}_{k-1|k-1}$ is the estimated state at time $k-1$ and $\tilde{x}_{k|k-1}$ is the predicted state at time k. The associated covariance matrix can be written as

$$\tilde{P}_{k|k-1} = F\hat{P}_{k-1|k-1}F^T + Q_k, \tag{3.13}$$

where $Q_k = E\{v_k v_k^T\}$. From the predicted state, we can get the predicted measurement

$$\tilde{z}_{k|k-1} = H\tilde{x}_{k|k-1}, \tag{3.14}$$

and use this to calculate the innovation vectors for all valid measurements,

$$\nu_k(i) = z_k(i) - \tilde{z}_{k|k-1} \quad \text{for} \quad i = 1, \cdots, m. \tag{3.15}$$

These vectors indicate the distance between the events and the estimated measurement. Finding the nearest event, we use its innovation to update the estimated state

$$\hat{x}_{k|k} = \tilde{x}_{k|k-1} + K\nu_k. \tag{3.16}$$

The Kalman filter gain is given by

$$K = \tilde{P}_{k|k-1}H^T S^{-1}, \tag{3.17}$$

where S is the innovation covariance matrix

$$S_k = H\tilde{P}_{k|k-1}H^T + R_k. \tag{3.18}$$

The state covariance matrix is updated by

$$\hat{P}_{k|k} = (I - KH)\tilde{P}_{k|k-1}. \tag{3.19}$$

3.2.3 Probabilistic Data Association Filter

A more robust method than the NNKF is The Probabilistic Data Association Filter (PDAF)[11]. The PDAF calculates the association probability for each valid measurement that falls inside a gate around the predicted measurement at the current time for an object. Instead of using the nearest measurement for updating the estimate, a weighted sum of all valid measurements are used for the updating process. This makes the filter less sensitive to clutter and noise and usually gives a better performance than the NNKF [10]. However since the algorithm is more complex, the computational time is greater.

3.2.3.1 The Algorithm

The algorithm for the PDAF function [10, 11] are similar to the NNKF with a few modifications. The first steps are just like the NNKF [10], predicting the target state and its associated covariance. Then compute the innovation and its covariance matrix S,

$$\tilde{x}_{k|k-1} = F\hat{x}_{k-1|k-1} \tag{3.20}$$

$$\tilde{z}_{k|k-1} = H\tilde{x}_{k|k-1} \tag{3.21}$$

$$\tilde{P}_{k|k-1} = F\tilde{P}_{k-1|k-1}F^T + Q_k \tag{3.22}$$

$$S_k = H\tilde{P}_{k|k-1}H^T + R_k. \tag{3.23}$$

Rather than choosing the nearest event for updating the track, the PDAF calculates the associated probabilities for each valid event inside a gate and calculates a combined innovation,

$$\nu_k = \sum_{i=1}^{m} \beta_k(i)\nu_k(i). \tag{3.24}$$

The associated probabilities is calculated using Poisson's clutter model [10]

$$\beta_k(i) = \begin{cases} \frac{e^{-0.5\nu(i)^T S^{-1}\nu(i)}}{P_{FA}\sqrt{|2\pi S|}\frac{(1-P_D)}{P_D}+\sum_{j=1}^{m} e^{-0.5\nu(i)^T S^{-1}\nu(i)}} & i = 1, 2, \cdots, m \\ \frac{P_{FA}\sqrt{|2\pi S|}\frac{(1-P_D)}{P_D}}{P_{FA}\sqrt{|2\pi S|}\frac{(1-P_D)}{P_D}+\sum_{j=1}^{m} e^{-0.5\nu(i)^T S^{-1}\nu(i)}} & i = 0 \end{cases}, \tag{3.25}$$

where P_{FA} is the false alarm probability and P_D is the detection probability.

The state estimate is updated with the combined innovation and the Kalman gain matrix

$$\hat{x}_{k|k} = \tilde{x}_{k|k-1} + K_k \nu_k. \tag{3.26}$$

The state covariance matrix is updated by

$$\hat{P}_{k|k} = \beta_k(0)\tilde{P}_{k|k-1} + (1 - \beta_k(0))\, P^c_{k|k} + P^s_k \tag{3.27}$$

where $P^c_{k|k}$ is defined as

$$P^c_{k|k} = \tilde{P}_{k|k-1} - K_k S K_k^T, \tag{3.28}$$

and the spread of the innovations

$$P^s_{k|k} = K \left(\sum_{i=1}^{m} \beta_k(i)\nu_k(i)\nu_k(i)^T - \nu_k \nu_k^T \right) K^T. \tag{3.29}$$

3.2.4 Track management

This section describes the way the tracking algorithm is managed.

3.2.4.1 Track Initiation

Every new event initiates the start of a new track. Since a lot of the events originate from noise or clutter a track score is assigned to each track. This track score is used to eliminate false tracks and is described later on.

3.2.4.2 Track Propagation

If a track have no validated events inside the gate, it will not be updated but the predicted state will be propagated to the next ping:

$$\hat{x}_{k|k} = \tilde{x}_{k|k-1}, \tag{3.30}$$

$$\hat{P}_{k|k} = \tilde{P}_{k|k-1}. \tag{3.31}$$

For each time a track is propagated the track score falls. When the track score fall below a predefined threshold the track is eliminated. The threshold is a fixed system parameter and is based on scenario and performance requirements.

3.2.4.3 Gating

To exclude unlikely events from the innovations, gating is performed. A normalized distance

$$d^2 = \nu^T S^{-1} \nu, \tag{3.32}$$

is used. If the normalized distance is lower than the predefined gate value G, the event is considered valid. For the NNKF the closest event inside the gate is used to update the track. For the PDAF method, all measurements inside the gate contributes to the innovations.

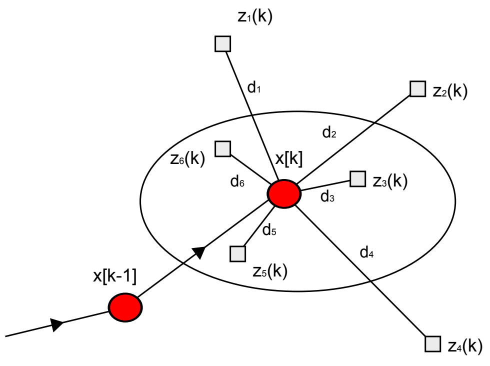

Figure 3.4: Diagram showing the gating principle. The red circles are the predicted states and the squares are the events. The events inside the circle are considered valid.

3.2.4.4 Track Fusing

To avoid redundant tracks, similar tracks are fused. Tracks that share the last n observations are combined. For two tracks i and j with estimated state vectors and covariance matrices, the combined state vector can be expressed as

$$x_c = \hat{x}_i + \hat{P}_i \hat{P}_{ij}^{-1} \left(\hat{x}_j - \hat{x}_i \right) \tag{3.33}$$

The combined covariance matrix can be expressed as

$$P_c = \hat{P}_i - \hat{P}_i \hat{P}_{ij}^{-1} \hat{P}_i, \tag{3.34}$$

where

$$\hat{P}_{ij} = \hat{P}_i + \hat{P}_j. \tag{3.35}$$

The history of track i will follow the new combined track and track j will be eliminated.

3.3 Feature Extraction

Being able to track a target enables the possibility to gather information about the target over a period of time. The information gathered is called *features* and is used to describe the target in such way that it easily can be distinguished from other objects. In this thesis static features will be extracted from each ping and later combined with the feature changes over time. This will produce a set of both static and temporal features.

In order to extract these features, the object of interest must first be isolated. This is done by identifying which of the sonar image cells that is a part of the object and which cells are background noise. This is done by segmentation.

3.3.1 Segmentation

For each state estimate used for updating the track a chip from the sonar image is made. This chip takes the track position and its n surrounding cells to make a small sonar image containing the object that caused the event.

Segmentation is to separate the image into regions which shares similar criteria. In this case there are two regions, the object of interest and background cells. In order to separate the object from background a segmentation method called *region growing* [8] is used.

3.3.1.1 Region growing segmentation

Region growing is a procedure that groups cells into regions based on predefined criteria and cell based properties. The approach is to start with one or a set of *seed* points. Neighboring cells to the seed cell who share the same criteria are included in the region and the region grows until there are no more neighboring cells who meets the criteria. An example of the approach can be as follows.

1. Start with finding the seed points. Make a binary image s with ones in the seed point and zeros in the rest of the cells.
2. Form a binary image f containing ones for each cell that meets the criteria. Cells that does not meet the criteria are labeled zero.
3. Append all the 1-valued points in f that are neighboring points with the seed points
4. Label each connected cell in the output image with a region label.

3.3.2 Features

The choice of features in this thesis has previously shown to give a good classification performance on divers [3, 12]. The same set of features will be tested to see if they are able to distinguish between divers and marine life.

3.3.2.1 Static features

The static features are listed and explained below.

1. *Area:* This is the surface of the object after segmentation. It is defined as

$$A = \sum_{i,j} = T(i,j) \tag{3.36}$$

   where $T(i,j)$ has a value of one for a cell inside the object and zero for cells outside.

2. *Perimeter:* The perimeter can be defined as the number of boundary cells. A boundary cell is a cell that is 4-connected[1] with any background cell.

[1]4-connected cells are any cell that touches one of the sides of a background cell.

3. *Compactness:* Is defined as

$$Compactness = \frac{4\pi \cdot A}{P^2}, \tag{3.37}$$

where A is the area and P the perimeter.

4. *Mean:* This feature measures the average reflected amplitude of the segmented object.

$$Mean = \frac{\sum_{i,j} G(i,j)}{N}, \tag{3.38}$$

where G is the amplitude of the cells in the object and N is the number of cells within the object.

5. *Variance:* The variance of the amplitude is defined as

$$Variance = \frac{\sum_{i,j} \left(G(i,j) - Mean\right)^2}{N}. \tag{3.39}$$

6. *Major Axis:* Length of the major axis of the smallest enclosing ellipse of the segmented object.

7. *Minor Axis:* Length of the minor axis of the smallest enclosing ellipse of the segmented object.

8. *Eccentricity:* Scalar that specifies the eccentricity of the ellipse that has the same second moment as the object. The eccentricity is the distance between the foci of the ellipse and its major axis length. The eccentricity of an ellipse lies between 0 and 1, where a circle has eccentricity 0 and a line segment eccentricity 1.

9. *First and second invariant moment:* A moment m_{pq} is of order $p+q$ and for a digital image $f(x,y)$ defined as

$$m_{pq} = \sum_x \sum_y x^p y^q f(x,y), \tag{3.40}$$

and the central moments are

$$\mu_{pq} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q f(x,y), \tag{3.41}$$

where

$$\bar{x} = \frac{m_{10}}{m_{00}} \quad \text{and} \quad \bar{y} = \frac{m_{01}}{m_{00}}. \tag{3.42}$$

The normalized central moments are then defined as

$$\eta_{pq} = \frac{\mu_{pq}}{\mu_{00}^{\gamma}}, \tag{3.43}$$

where

$$\gamma = \frac{p+q}{2} + 1. \tag{3.44}$$

This leads to the first and second invariant moment and can be expressed as

$$\phi_1 = \eta_{20} + \eta_{02} \tag{3.45}$$

$$\phi_2 = (\eta_{20} - \eta 02)^2 + 4\eta_{11}^2 \tag{3.46}$$

3.3.2.2 Temporal features

These static features can be used to produce temporal counterparts. The temporal features are as follows

1. *Mean:* The mean value of the static feature f is

$$\mu_i = \frac{1}{n}\sum_{j=1}^{n} f_{i,j}, \tag{3.47}$$

   where n is the number of pings and i is feature number

2. *Variance:*

$$\sigma_i^2 = \frac{1}{n}\sum_{j=1}^{n}(f_{i,j} - \mu_i)^2. \tag{3.48}$$

3. *Mean rate of change:*

$$\mu_{r,i} = \frac{1}{n-1}\sum_{j=2}^{n}(f_{i,j} - f_{i,j-1}). \tag{3.49}$$

4. *Variance rate of change:*

$$\sigma_{r,i}^2 = \frac{1}{n-1}\sum_{j=2}^{n}((f_{i,j} - f_{i,j-1}) - \mu_{r,i})^2. \tag{3.50}$$

By using these temporal features on the static features the amount of features increases from 10 to 40.

3.4 Classifier

The task of the classifier is to use the features extracted from an object to define the object to a specific class or category [5]. In order for a classifier to do this, it has to learn how the features for each class is distributed. There are two ways of training a classifier [5]. By having a *training set* of feature vectors witch are labeled by class, the classifier can learn how the features of each class is distributed. This is called *supervised learning.* The other approach is to train the classifier with an unlabeled training set. Giving no information of the number of classes and class distribution, the classifier groups the features by their statistics making the different classes. This is called *unsupervised learning.* In this thesis supervised learning is used.

3.4.1 Bayesian Decision Theory

In Bayesian decision theory, the statistics for each class is used to set the probability that an object belongs to a certain class. If the distribution of a feature is dependent of class, it is said to be a *class conditional density.* Having the knowledge how the conditional densities and *prior* probabilities for each class, a decision rule can be made by choosing the class with the highest *posterior* probability. This can be found using the Bayes formula.

3.4.1.1 Bayes formula

A decision rule uses the information given by the features to decide which class a feature vector most likely belongs to. Having information about the prior probability for each class $P(\omega_i)$ and the conditional densities $p(\mathbf{x}|\omega_i)$, the posterior probability can be found using Bayes formula [5]:

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})}, \tag{3.51}$$

where

$$p(\mathbf{x}) = \sum_{i=1}^{c} p(\mathbf{x}|\omega_i)P(\omega_i) \quad \text{for classes } i = 1, \cdots, c. \tag{3.52}$$

The term $p(\mathbf{x})$ can be viewed as a scale factor that ensures that the sum of all posterior probabilities sum up to one. This term is unimportant as far as decision making is concerned.

3.4.2 Discriminant Functions

In this thesis the choice of classifier has fallen on the use of discriminant functions. Discriminant functions use feature statistics derived from the training set to classify an object. Given a feature vector $\mathbf{x}$ from an unknown object and a set of discriminant functions $g_i(\mathbf{x})$ for classes $1, \cdots, c$, the classifier labels the object to class ω_i if

$$g_i(\mathbf{x}) > g_j(\mathbf{x}) \quad \text{for all} \quad j \neq i. \tag{3.53}$$

There are many choices for a discriminant function [5]. Bayes rule can be one choice

$$\begin{aligned} g_i(\mathbf{x}) &= P(\omega_i|\mathbf{x}) && (3.54)\\ &= \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})}, && (3.55) \end{aligned}$$

Since any scaling to the discriminant function will not influence the decision, the function can be simplified to

$$g_i(\mathbf{x}) = p(\mathbf{x}|\omega_i)P(\omega_i). \tag{3.56}$$

Any monotonically increasing functions used on $g_i(\mathbf{x})$ will not change the decision, so it can be re-written to

$$g_i(\mathbf{x}) = \ln p(\mathbf{x}|\omega_i) + \ln P(\omega_i) \tag{3.57}$$

This may seem to complicate the function, however as will be shown later this will lead to computational simplification.

3.4.2.1 Discriminant function for a Gaussian model

The Gaussian probability density function or a feature x is defined as,

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}, \tag{3.58}$$

with $\mu = E[x]$ being the expected value of x, and $\sigma^2 = E[(x-\mu)^2]$ being the variance. For the multivariate case the feature vector $\mathbf{x}$ has a dimension d equal to the number of features used. If each of the d components, x_i are independent and normally distributed with their own mean and variance, their joint density has the form

$$p(\mathbf{x}) = \prod_{i=1}^{d} p(x_i) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi}\sigma_i} \exp\left\{-\frac{1}{2}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2\right\} \tag{3.59}$$

$$= \frac{1}{(2\pi)^{d/2}\prod_{i=1}^{d}\sigma_i} \exp\left\{-\frac{1}{2}\sum_{i=1}^{d}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2\right\}. \tag{3.60}$$

The exponent can be written as

$$-\frac{1}{2}\sum_{i=1}^{d}\left(\frac{x_i-\mu_i}{\sigma_i}\right)^2 = -\frac{1}{2}\left(\mathbf{x}-\mu\right)^T \mathbf{\Sigma}^{-1}\left(\mathbf{x}-\mu\right), \tag{3.61}$$

where $\mathbf{\Sigma}^{-1}$ is the inverse covariance matrix. The determinant of the covariance matrix $|\mathbf{\Sigma}|$ is the product of all the variances, hence the general multivariate normal density in d dimensions can be written as,

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right\}. \tag{3.62}$$

Inserting this into (3.57) leads to the following discriminant function

$$g_i(x) = -\frac{1}{2}(\mathbf{x}-\mu_i)^T\mathbf{\Sigma}_i^{-1}(\mathbf{x}-\mu_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}|\mathbf{\Sigma}_i| + \ln P(\omega_i). \tag{3.63}$$

Because the term $\frac{d}{2}\ln 2\pi$ is independent of i it acts only as a constant and can be dropped. The resulting discriminant function is quadratic and can be written as:

$$g_i(\mathbf{x}) = \mathbf{x}^T\mathbf{W}_i\mathbf{x} + \mathbf{w}_i^T\mathbf{x} + w_{i0}, \tag{3.64}$$

where

$$\mathbf{W}_i = -\frac{1}{2}\mathbf{\Sigma}_i^{-1}, \tag{3.65}$$

$$\mathbf{w}_i = \mathbf{\Sigma}_i^{-1}\mu_i \tag{3.66}$$

and

$$w_{i0} = -\frac{1}{2}\mu_i^T\mathbf{\Sigma}_i^{-1}\mu_i - \frac{1}{2}|\mathbf{\Sigma}_i| + \ln P(\omega_i). \tag{3.67}$$

3.4.2.2 Linear Discriminant Function

It is possible to make a quadratic discriminant function linear. A linear discriminant functions are more easy to compute and are popular to use as trial classifiers [5].

Because of their simplicity they can also give better classification performance than more complicated functions [4].

In order to make this quadratic discriminant function linear, the *joint covariance matrix* is found by taking the weighted average of the class covariance matrices,

$$\boldsymbol{\Sigma} = \sum_{i=1}^{c} P(\omega_i)\boldsymbol{\Sigma}_i. \tag{3.68}$$

This simplifies the quadratic form in (3.63) to

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x} - \mu_i)^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \mu_i) + \ln P(\omega_i). \tag{3.69}$$

Expanding the quadratic form in the first term leads to

$$g_i(\mathbf{x}) = -\frac{1}{2}\left(\mathbf{x}^T \boldsymbol{\Sigma}^{-1}\mathbf{x} - 2\boldsymbol{\Sigma}^{-1}\mu_i\mathbf{x} + \mu_i^T \boldsymbol{\Sigma}^{-1}\mu_i\right) + \ln P(\omega_i). \tag{3.70}$$

Dropping the $\mathbf{x}^T\boldsymbol{\Sigma}^{-1}\mathbf{x}$ term which is independent if i, the discriminant function becomes linear and can be written as:

$$g_i(\mathbf{x}) = \mathbf{w}_i^T\mathbf{x} + w_{i0}, \tag{3.71}$$

where

$$\mathbf{w}_i = \boldsymbol{\Sigma}^{-1}\mu_i \tag{3.72}$$

and

$$w_{i0} = -\frac{1}{2}\mu_i^T\boldsymbol{\Sigma}^{-1}\mu_i + \ln P(\omega_i). \tag{3.73}$$

3.4.2.3 Discriminant functions for two class problems

If there are only two classes to classify, the discriminant functions can be combined to define a single discriminant function

$$g(\mathbf{x}) = g_1(\mathbf{x}) - g_2(\mathbf{x}) \tag{3.74}$$

For this case the decision rule will choose ω_1 if $g(\mathbf{x}) > 0$ otherwise ω_2. In the special case where a feature vector lies on the decision border $g(\mathbf{x}) = 0$, the vector has equal class probability and can be assigned to either class. Using the linear discriminant function from (3.71) for each class, the new single discriminant function can be written as follows:

$$
\begin{aligned}
g(\mathbf{x}) &= g_1(\mathbf{x}) - g_2(\mathbf{x}) && (3.75)\\
&= \left(\mathbf{w}_1^T\mathbf{x} + w_{10}\right) - \left(\mathbf{w}_2^T\mathbf{x} + w_{20}\right) && (3.76)\\
&= (\mathbf{w}_1 - \mathbf{w}_1)^T\mathbf{x} + (w_{10} - w_{20}) && (3.77)\\
&= \mathbf{w}^T\mathbf{x} + w_0, && (3.78)
\end{aligned}
$$

where

$$\mathbf{w} = \mathbf{w}_1 - \mathbf{w}_1 \qquad (3.79)$$

and

$$w_0 = w_{10} - w_{20}. \qquad (3.80)$$

3.5 Summary

In this chapter the theoretical background needed to solve the thesis problem is presented. The topics covered are object detection, tracking, feature extraction and classification methods.

For object detection two types of CFAR filters has been presented, namely a spatial cell averaging filter and a temporal cell averaging filter. The spatial cell averaging filter uses the neighboring cells to estimate the average amplitude while the temporal cell averaging filter takes the amplitude of each cell over multiple pings to calculate the average amplitude.

For the problem of tracking two different methods has been introduced. The Nearest Neighbor Kalman Filter (NNKF) and a Probabilistic Data Association Filter (PDAF). While the NNKF is easy to implement it lacks robustness when there is a lot of noise. The PDAF is more complicated but is less sensitive to noise and less chance of false event adoption since the track update is done by a weighted sum of events.

A set of static and temporal features have been listed. The methods for extracting these features rises the need for a method for segmenting valid object from background in the sonar image. For this task a region growing algorithm is proposed.

A linear discriminant function is described and suggested as a classifier. Even though more complicated and sophisticated classifiers exists, the linear discriminant function often shows good results when it comes to classification.

Chapter 4

Methods

This section describes the Matlab implementation of the theory described in Chapter 3. A major part of the thesis has been to solve the problem of extracting the features used to train and test the classifier. In order to extract the desired features, the objects of interest must first be detected in the match filtered data and then tracked to collect a large training set.

The methods implemented are function for *event detection*, *tracking function*, a *feature extraction* function and *classification*.

All implementation is done in Matlab. The functions used for CFAR filtering and event detection where already produced at Kongsberg Defence Systems (KDS) and has been used unaltered in this thesis. For the classification and feature selection the Matlab toolbox *prtools*[6] has been used.

Function for reading the matched filtered sonar data files is also used and provided by KDS.

4.1 Detection

Both the CFAR filter and event detection algorithms used in this thesis where already made by KDS. The files `pingPingFilter.m` and `findFMEvents.m` are used unaltered and are only given a short description.

The event detection is done by filtering the sonar image with a temporal CFAR filter described in section 3.1.2. The filter uses the cell variance over each ping to average the sonar image.

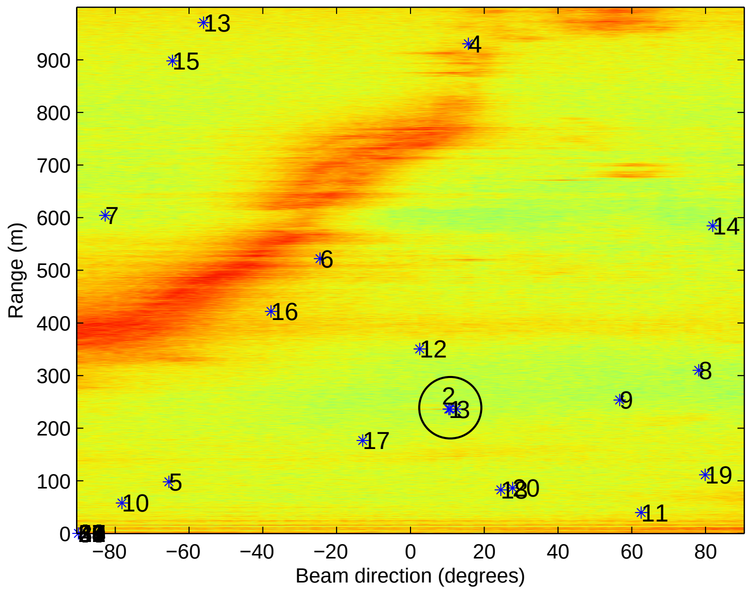

Figure 4.1: Event detection in sonar image. Events are identified by blue asterisks. A diver with semi-closed breathing system is detected by three events and marked by the black circle.

```
function [pingPingOutput, mean, variance] ...
  = pingPingFilter(currentPing, mean, variance, pingCounter, parameters)
```

Input parameters: `currentPing` are the sonar image produced by the matched filtered data from the current ping. The `mean` and `variance` are two matrices of the same size as the sonar image. They contain the current mean and variance of each cell. `pingCounter` is an integer counting the pings processed. `parameters` contains the scaling factors for the mean and variance, see Figure 3.3.
Output parameters: `pingPingOutput` is an image matrix consisting of the square root of the `currentPing` cell variance divided by the temporal mean of the cell variances,

$$pingPingOutput = \sqrt{\frac{(currentPing - aprioriMean)^2}{aprioriVariance}}. \qquad (4.1)$$

Output `mean` and `variance` are the posterior matrices which includes the current ping. They are used as input for when the function is used on the next ping.

A fixed threshold is used on the output image to detect events. Any cell in the image `pingPingOutput` higher than the threshold will be set as an event. The function `findFMEvents` is used to detect events in the filtered image:

```
function [fmEvents, eventCounter] = findFMEvents(fmSweepData,...
       fmNormalizedSweepData, NoOfScans, NoOfBeams, scansPerMeter, ...
       maxNoOfEventsPerBlock)
```

Input parameters: `fmSweepData` and `fmNormalizedSweepData` are the same as *currentPing* and *pingPingOutput*. `noOfScans` and `noOfBeams` are the length and width of the sonar image. `scansPerMeter` indicates how many cells there is per meter in the range direction of the image. `maxNoOfEventsPerBlock` sets the maximum length of the output event matrix.
Output parameters: `fmEvents` is a matrix of events containing range, bearing and amplitude. `eventCounter` is a integer containing number of events found.

Targets like divers may have a larger extent than one cell and may result in more than one event. A sonar image containing multiple events are shown in Figure 4.1.

4.2 Tracking

The tracking algorithm is divided over several functions. The functions used are:

- `getKalmanParameters`
- `trackInit`
- `predictKalmanStates`
- `dataAssociation`
- `updateTrack`
- `fuseTrack`

The tracking function is used on the CFAR filtered data. For the first ping all events will start a new track. The new tracks will be initiated by the function `trackInit` and the the next state is predicted. Data association between the tracks and new events is done by either the Nearest Neighbor Kalman Filter or Probabilistic Data Association.

4.2.1 Kalman parameters

All system specific parameters are set in the function `getKalmanParameters`. The output variable is a MATLAB structure array containing the parameters listed in table 4.1.

The transition and measurement matrices `F` and `H`, used are the same as described in equations (3.8) and (3.10). The noise covariance matrices `Q` and `R` are the same size

F	State transition matrix
H	Measurement transition matrix
Q_SCALFACTOR	Scaling factor for the process noise covariance matrix
R_SCALEFACTOR	Scaling factor for the measurement noise covariance matrix
Q	Process noise covariance matrix
R	Measurement noise covariance matrix
MAXPROPAGATE	Number of times a track can propagate in a row before termination
MAXCOMMONEVENTS	Number of common events shared between two tracks before fusing
GATE	Size of the gate determining which measurements are valid
FALSEALARMPROBABILITY	The probability for false alarm
DETECTIONPROBABILITY	The detection probability

Table 4.1: Kalman Parameters

FALSEALARMPROBABILITY	0.005
DETECTIONPROBABILITY	0.65

Table 4.2: System defined probabilities

as F and H. In this thesis both matrices are identity matrices scaled by Q_SCALFACTOR and R_SCALFACTOR. How many times a track can propagate before it is terminated is defined by MAXPROPAGATE. The GATE defines the area around a predicted state where events are considered valid. The gate variable is a constant that act as a threshold for the *normalized distances* for all the events. See Figure 3.4. MAXCOMMONEVENTS sets how many common events two tracks must share before they are fused. The parameters FALSEALARMPROBABILITY and DETECTIONPROBABILITY are parameters defined by the tracking system. In this case the parameters are set as defined in Table 4.2.

4.2.2 Track initiation

This function initiates the track. A Matlab structure array **trackObjects** are created with one entry for every event. The structure array contains all track history, event chips and feature information. This function only runs one time when the track starts.

```
function [trackObjects] = trackInit(fmEvents,eventCounter,kalmanParameters)
%Initialize tracks for all events
dX = kalmanParameters.dX;
```

```
dY = kalmanParameters.dY;
noPings = kalmanParameters.MAXPROPAGATE;

    for idObj = 1:eventCounter
        %The Current State:
        trackObjects{idObj}.x_current_state = [fmEvents(idObj,1:2),dX,dY]';
        %Martrx listing events used:
        trackObjects{idObj}.eventsUsed = fmEvents(idObj,1:2);
        %Initial State Covariance:
        trackObjects{idObj}.sCovMatrix = 1*eye(4);
        %no pings to propagate without valid event
        trackObjects{idObj}.trackScore = noPings;
        %Saving the track
        trackObjects{idObj}.fmTrack =[fmEvents(idObj,1:2)];
        %Logging number of times track has been propagated:
        trackObjects{idObj}.noTimesPropagated = 0;
    end
```

Input parameters: `fmEvents` and `eventCounter` is the array and number of events produced by `findFMEvent.m`. `kalmanParameters` consists of the parameters described in previous section (Section 4.2.1).
Output parameters: `trackObjects` is a structure array containing all the tracks and the tracking variables.

The algorithm gets the parameters for the Kalman algorithm. These parameters are found in the function `getKalmanParameters`. They are system dependent and can be changed to fit different systems. The MATLAB structure array `trackObjects` stores all data associated which each track. This includes variables used for the tracking algorithm and object features extracted during the track. All these variables are explained in the next sections. After track initiation the tracking algorithm predicts the next state, measurement, covariance matrix and innovation covariance matrix.

4.2.3 Predict states

After the tracks have been initialized the next state for each track are predicted by the function `predictKalmanStates`.

```
function [trackObject] = predictKalmanStates(trackObject,kalparam)
%Estimates the next state of the object beeing tracked using the kalman
%parameters in input kalparam.

for idObj = 1 : length(trackObject)
    %Next predicted state
    trackObject{idObj}.x_next_state = ...
```

```
     kalparam.F*trackObject{idObj}.x_current_state;

    %Update predicted covariance matrix
    trackObject{idObj}.pCovMatrix = ...
     kalparam.F*trackObject{idObj}.sCovMatrix*kalparam.F' + kalparam.Q;

    %Next predictes measurement:
    trackObject{idObj}.z_next_state = ...
     kalparam.H*trackObject{idObj}.x_next_state;

    %Estimate the invation/residual covariance matrix (S)
    trackObject{idObj}.S = ...
     kalparam.H*trackObject{idObj}.pCovMatrix*kalparam.H' + kalparam.R;

end
```

Input parameters: `trackObjects`, `kalmanParameters`.
local parameters: `x_next_state`, `x_current_state`, `sCovMatrix`, `pCovMatrix`.
Output parameters: `trackObjects`.

This function predicts the next state for the object $\tilde{x}_{k|k-1}$, the predicted measurement $\tilde{z}_{k|k-1}$, and predicted covariance matrix for the state $\tilde{P}_{k|k-1}$ and innovation matrix S as described in equations (3.20) to (3.22) and (3.18). All predictions are saved in the structure array `trackObjects`.

After the state vectors and covariance matrices for all tracks are predicted, the tracking algorithm tries to associate the events found in the next ping to each track. This is done by the function `dataAssociation`.

4.2.4 Data Association

The data association function associates all events from the last ping to the existing track. This is done either using the NNKF algorithm or the PDAF algorithm depending on what is specified in the input.

```
function [TAS] = dataAssociation ...
    (fmEvents, eventCounter, trackObject,kalmanParameters, ASS_TYPE)

for idObj = 1: length(trackObject)
    %Initialize variables:
    v = zeros(eventCounter,2);
    d_2 = zeros(eventCounter,1);

    %Calculate the innovations/residual vector(v) and normal distaces(d_2)
    for idEv = 1:eventCounter
        %Finding the innovations:
```

```
        v(idEv,:) = (fmEvents(idEv,1:2)' ...
            - trackObject{idObj}.z_next_state)';

        %Finding the normalized distances (d_2)
        d_2(idEv) = v(idEv,:)* inv(trackObject{idObj}.S)*v(idEv,:)';
    end

    %Nearest Neighbor Kalman Filter:
    if (strcmp(ASS_TYPE,'NNKF'))
        TAS(idObj).vector = v;
        TAS(idObj).d_2 = d_2;
    end

    %Probabilistic Data Association Filter:
    if (strcmp(ASS_TYPE,'PDAF'))

        %Get false alarm probability and detection probability from Kalman
        %parameters.
        fa = kalmanParameters.FALSEALARMPROBABILITY;
        pd = kalmanParameters.DETECTIONPROBABILITY;

        %Initialize arrays:
        e = zeros(eventCounter,1);
        p = zeros(eventCounter+1,1);

        %Calculate conditional probability:
        b =  fa* (det(2*pi*trackObject{idObj}.S)^0.5)*((1-pd)/pd);
        for idEv = 1:eventCounter
          e(idEv) = exp(-0.5*v(idEv,:)...
              *inv(trackObject{idObj}.S)*v(idEv,:)');
        end

        %Probability that all events are false alarms:
        p(1) = b/(b+sum(e));

        for idEv = 1:eventCounter
          p(idEv+1) = e(idEv)/(b+sum(e));
        end

        %Save probabilities, normalized distances and innovations to
        %output:
        TAS(idObj).vector = v;
        TAS(idObj).d_2 = d_2;
        TAS(idObj).p = p;

    end
end
```

Input parameters: `fmEvents` is the output matrix from the function `findFMEvents`. It consists of all the events found in the last ping. `eventCounter` is the number of

events found in the last ping. `trackObjects` contains the predicted measurements for the tracks. The input parameter `kalmanParameters` is needed to calculate the conditional probabilities for the PDAF model. `ASS_TYPE`defines if the function shall use the NNKF method or the PDAF method to calculate the track association.
Output parameters: `TAS` .

This function associates the events from the latest ping to the current tracks. This is done by either the NNKF model or PDAF. The method used is specified in the input parameter `ASS_TYPE`. The function calculates the innovations `v` (3.15) and normalized distances `d_2` (3.32) for all the events. If the PDAF method is used, the function calculates the association probabilities β_i as described in (3.25). The associated probabilities, normalized distance vector and innovations are returned in the output parameter `TAS`. If the NNKF is chosen, only innovation vector `v` and normalized distance vector `d_2` are returned.

For the PDAF method, the function need the *false alarm probability* and *detection probability* which is specified in the Kalman parameters. These probabilities are used to compute the conditional probabilities. When the innovation vector and associated probabilities are found the tracks need to be updated. This is done in the `trackUpdate` function.

4.2.5 Track Update

The track update is done by the function `updateTrack`. The function runs trough every track and updates the state estimates with the associated measurements from the last ping.

```
function [trackObjects] = ....
    updateTrack(trackObjects,fmEvents,kalmanParameters,TAS, ASS_TYPE)
%function for updating tracks by either the NNKF or PDAF method.
```

Input parameters: `trackObjects` containing all trackdata including the predicted states and covariance matrices, `fmEvents` with all events from last ping. The `kalmanParameters` contains parameters needed for the PDAF method, `TAS` is the structure array with the calculated innovations, normalized distances and conditional probabilities the events have for the different tracks. `ASS_TYPE`specifies which method to use.
Output parameters: `trackObjects` with updated track data for each track.

The function updates the track using either the NNKF method or PDAF method, which is defined in the input parameter `ASS_TYPE`. If the NNKF method is chosen, the function takes the normalized distances found in the data association function and checks if any events are inside the gate. If so, the event nearest the predicted measurement is selected and used to update the state estimate (3.16) and the track score is updated. If no events lies inside the gate, the track is propagated and the track score is reduced by one. The propagation is logged in a counter that counts how many times a track has propagated without any valid events. If the track score is zero the track is terminated. The Kalman filter gain and state covariance matrices are updated and the new state estimate is added to the track data.

```
if strcmp(ASS_TYPE,'NNKF')
   disp('Association type NNKF');
   %Update track according to nearest neighbor kalman filter:
   for idObj = 1:  length(trackObjects)
       %Get the normalized distances and innovation vectors:
       d_2 = TAS(idObj).d_2;
       v = TAS(idObj).vector;
       %Calculate Kalman filter gain:
       K = trackObjects{idObj}.pCovMatrix*kalmanParameters.H'*...
           inv(trackObjects{idObj}.S);
       %Get the measurement of the event that is closest to the predicted
       %measurement.
       k = find(d_2==min(min(d_2)));
       %Update the predicted state and state ovariance matrix:
       %Only update if event lies inside gate.
       if d_2(k)≤kalmanParameters.GATE
           trackObjects{idObj}.x_current_state = ...
               trackObjects{idObj}.x_next_state + K*v(k,:)';
           %reset track score
           trackObjects{idObj}.trackScore = kalmanParameters.MAXPROPAGATE;
           %set event is used in track
           trackObjects{idObj}.eventsUsed = ...
               [trackObjects{idObj}.eventsUsed;fmEvents(k,1:2)];
       %No events inside the gate, and trackscore is above zero:
       elseif (d_2(k)>kalmanParameters.GATE && ...
               trackObjects{idObj}.trackScore > 0)
           %Propagate predicted state
           trackObjects{idObj}.x_current_state = ...
               trackObjects{idObj}.x_next_state;
           %reduce trackScore by one
           trackObjects{idObj}.trackScore =  ...
               trackObjects{idObj}.trackScore - 1;
           %Log number of times the track has been propagated:
           trackObjects{idObj}.noTimesPropagated = ...
               trackObjects{idObj}.noTimesPropagated + 1;
       else
           %If no measurement are inside gate and track score is zero,
           %delete track:
           trackObjects{idObj}=[];
           continue;
```

```
        end
        %Update state covariance matrix:
        trackObjects{idObj}.sCovMatrix = ...
            (eye(4)-K*kalmanParameters.H)*trackObjects{idObj}.pCovMatrix;
        %Update trackData
        trackObjects{idObj}.fmTrack = [trackObjects{idObj}.fmTrack;...
            trackObjects{idObj}.x_current_state(1:2)'];
    end
    %Sort out deleted tracks
    trackObjects(cellfun(@isempty,trackObjects))=[];
```

If the PDAF model is chosen, the function follows the same procedure as with the NNKF approach. Instead of updating the state estimate with the nearest event, all events that lies inside the gate are used to calculate the combined innovation `v2` (3.24). This is used to update the state estimate and the state covariance matrix as described in equations (3.26) to (3.29). If no events lies inside the gate the track is propagated and the track score is reduced. If the track score falls to zero the track is terminated.

```
elseif strcmp(ASS_TYPE,'PDAF')
    disp('Association type PDAF');
    %Update track according to nearest neighbor kalman filter:
    for idObj = 1:  length(trackObjects)
       insideGate = 0;
       %Get the normalized distances, innovation vectors and conditional
       %probabilities:
       d_2 = TAS(idObj).d_2;
       v = TAS(idObj).vector;
       p = TAS(idObj).p;

       %Calculate Kalman filter gain:
       K = trackObjects{idObj}.pCovMatrix*kalmanParameters.H'*...
           inv(trackObjects{idObj}.S);
       %Get the measurement of the event that is closest to the predicted
       %measurement:
       k = find(d_2==min(min(d_2)));
       %Check if any events are inside gate:
       %Compute the combined inovation/residual of the measurement
       %inside the gate.
        for i=1:length(d_2)
            if(d_2(i)≤kalmanParameters.GATE)
                v2(i,:)=p(i+1)*v(i,:);
                s(i)=p(i+1)*v(i,:)*v(i,:)';
                insideGate = 1;
            end
        end
        %Update the predicted state and state ovariance matrix
        %Only update if measurement lies inside gate.
        if (insideGate)
            if (length(v2(:,1))>1)
                v2 = sum(v2);
            end
            %Calculate the spread of inovations:
            ps = K*(sum(s)-v2*v2')*K';
            %Covariance update with correct measurement:
            pc = trackObjects{idObj}.pCovMatrix - ...
                K*trackObjects{idObj}.S*K';
            %Update state estimate:
            trackObjects{idObj}.x_current_state = ...
                trackObjects{idObj}.x_next_state + K*v2';
            %Reset track score:
            trackObjects{idObj}.trackScore = kalmanParameters.MAXPROPAGATE;
            %Set event is used in track (use nearest event k):
            trackObjects{idObj}.eventsUsed = ...
                [trackObjects{idObj}.eventsUsed;fmEvents(k,1:2)];
            %Updated state covariance matrix
            trackObjects{idObj}.sCovMatrix = ...
                p(1)*trackObjects{idObj}.pCovMatrix + (1-p(1))*pc + ps;
            %Check if NaN and delete track
            if isnan(trackObjects{idObj}.sCovMatrix)
                trackObjects{idObj}=[];
                continue;
            end
        %If there is no events inside the gate, propagate track:
        elseif ¬insideGate && trackObjects{idObj}.trackScore > 0
            %Propagate predicted state:
            trackObjects{idObj}.x_current_state = ...
                trackObjects{idObj}.x_next_state;
            %Reduce trackScore
            trackObjects{idObj}.trackScore =  ...
                trackObjects{idObj}.trackScore - 1;
            %Log number of times the track has been propagated:
            trackObjects{idObj}.noTimesPropagated = ...
                trackObjects{idObj}.noTimesPropagated + 1;
        else
            %If no measurement are inside gate, delete track.
            trackObjects{idObj}=[];
            continue;
        end
        %Update trackData
        trackObjects{idObj}.fmTrack = [trackObjects{idObj}.fmTrack;...
            trackObjects{idObj}.x_current_state(1:2)'];
    end
    %Sort out deleted tracks
    trackObjects(cellfun(@isempty,trackObjects))=[];
```

After all tracks are updated the tracking algorithm checks if any tracks may originate from the same object. As seen in Figure 4.1 the diver in the sonar image causes three events that leads to three different tracks. Having three almost similar tracks for one object is unnecessary and for this reason a form of track fusion is applied. This is done by the function `fuseTrack`.

4.2.6 Track Fusion

Tracks that are similar are likely to originate from the same object. To avoid redundant tracks track fusion as applied. The function `fuseTrack` fuses tracks that are in close proximity and follow the same direction.

```
function [trackObjects] = fuseTrack(trackObjects, kalmanParameters)
%Function fuses tracks that share the last n events.
maxComEvents = kalmanParameters.MAXCOMMONEVENTS;
%Check if any tracks shares the same events:
if(length(trackObjects)>1)
    for k = 1:(length(trackObjects)-1)
        %Length of track has to be larger then number of common events.
        if (¬isempty(trackObjects{k})&& ...
                length(trackObjects{k}.eventsUsed)≥ maxComEvents)
            %Save last three events in matrix a:
            a = trackObjects{k}.eventsUsed(end-(maxComEvents-1):end,:);

            for l = k+1:length(trackObjects)
                if (¬isempty(trackObjects{l}) && ...
                     length(trackObjects{l}.eventsUsed)≥ maxComEvents)
                    %Save last three events in matrix b:
                    b = ...
                    trackObjects{l}.eventsUsed(end-(maxComEvents-1):end,:);
                    %Check if the last three events in track a shares the
                    %same cells as the last events in track b:
                    c = eq(round(a),round(b));
                    %If the two tracks share the three last events, fuse
                    %tracks:
                    if all(c(:)) == 1
                        trackObjects = fuseTwoTracks(trackObjects,k,l);
                    end
                end
            end
        end
    end
end
%Sort out deleted tracks
trackObjects(cellfun(@isempty,trackObjects))=[];
end
```

Input parameters: trackObjects contains all the updated tracks. The input kalmanParameters contains the limit for haw many common events two tracks can have before they are fused.
Output parameters: Returns trackObjects where redundant tracks are removed.

The function checks if any two tracks share the last n events. The number of common events before fusing is defined by the MAXCOMMONEVENTS variable which is found in kalmanParameters. If the two tracks share enough common events, the two tracks are fused. The definition of a common event is that the events from each track lies in the same cell. Tracks that meets these requirements are fused by the sub function fuseTwoTracks:

```
function [trackObjects] = fuseTwoTracks(trackObjects,i,j)
    %Fuse tracks i and j
    x1 = trackObjects{i}.x_current_state;
    x2 = trackObjects{j}.x_current_state;
    P1 = trackObjects{i}.sCovMatrix;
    P2 = trackObjects{j}.sCovMatrix;

    %Combined covariance matrix
    PC = P1 - P1*inv(P1+P2)*P1;
    %Commbined state vector
    xc = x1 + P1*inv(P1+P2)*(x2 - x1);
    %Update trackObjects
    trackObjects{i}.fmTrack = ...
        [trackObjects{i}.fmTrack(1:end-1,:);xc(1:2)'];
    trackObjects{i}.sCovMatrix = PC;

    %Remove other track
    trackObjects{j}=[];
end
```

Input parameters: trackObjects and the indexes i and j specifies the tracks to fuse.
Output parameters: trackObjects.

The fusing algorithm follows the equations defined in (3.33) and (3.34). Since the two track is expected to originate from the same object it is only necessary to keep track information from one of the two tracks. The combined state and covariance matrix are computed and added to the surviving track. The other track is then terminated.

This method ensures that there only exists one track for each potential object. Tracks that are initiated by random events are swiftly terminated by keeping a track score and removing tracks where the score drops to zero. The tracking algorithm

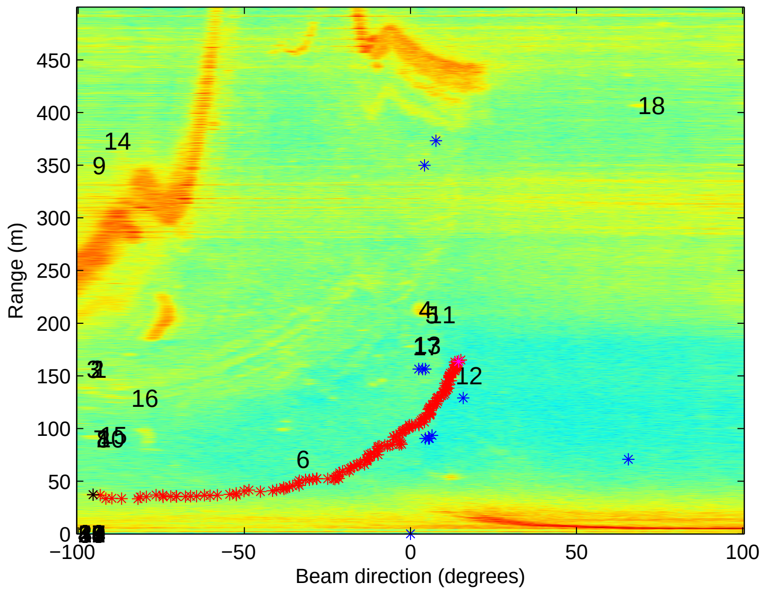

Figure 4.2: Track of a marine mammal over 161 pings. The red asterisks shows the track of the object. The blue asterisks indicates the events found in the last ping. The track started in the center of the image and the black asterisks indicates the last location of the track.

$$\begin{tikzcd}\boxed{\text{Extracted Chip}} \arrow[r] & \boxed{\text{Segmentation}} \arrow[r] & \boxed{\text{Feature Extraction}}\end{tikzcd}$$

Figure 4.3: Float diagram of the feature extraction procedure.

should now hold the potential to gather features over time from objects detected in the sonar images. Figure 4.2 shows a track of a possible marine mammal over 166 pings using the PDAF method.

4.3 Feature Extraction

During the tracking of an object the features are extracted for each ping. The feature extraction is done by extracting a chip of n cells surrounding the current state of the track. This chip is then segmented into background and foreground cells in order to extract the cells containing echo originated from the object. From these object cells information is extract to produce the features used for classification. Se Figure 4.3 for float diagram.

4.3.1 Chip Extraction

From each current state a chip is extracted containing the event or events that originated the state. In order to be sure that the hole object is extracted, the chip is approximately 20 meters times 10 degrees with the state estimate in center of the chip. This is done by the function `chipExtract`.

```
function [trackObjects] = chipExtract(trackObjects,currentPing,...
    pingPingOutput,scansPerMeter,kalParam)
%Extracts an image chip from the current ping for each track based on the
%state estimate for the track.

CHIPSIZE = [10 20]; %[meters degrees]
%Size of chip:
[M N] = size(currentPing);

%Run through the tracks and extract chip
for idObj = 1 : length(trackObjects)

    %Only extract chip if we have a valid event
    if (trackObjects{idObj}.trackScore == kalParam.MAXPROPAGATE)
        %Find center of chip. (last track position)
        trackPos = trackObjects{idObj}.fmTrack(end,:);
        centerChip =  [round(trackPos(2)*scansPerMeter),...
            round(trackPos(1))];
        %Set chip borders:
        top= centerChip(1)-round(CHIPSIZE(1)*scansPerMeter);
        buttom=centerChip(1)+round(CHIPSIZE(1)*scansPerMeter);
        left = centerChip(2)-CHIPSIZE(2);
        right = centerChip(2)+CHIPSIZE(2);
        %Correct if out of bounds
        if (top < 1)
            buttom = buttom + diff([topp 1]);
            top = 1;
        end
        if (buttom > M)
            top = top + diff([buttom M]);
            buttom = M;
        end
        if (left < 1)
            right = right + diff([left 1]);
            left = 1;
        end
        if (right > N)
            left = left + diff([right N]);
            right = N;
        end
        %Extract chip from current ping and normalized ping
        chip = currentPing(top:buttom,left:right);
        normalizedChip = pingPingOutput(top:buttom,left:right);

        %If chip elements exixist add to
        if (isfield(trackObjects{idObj},'Chips'))
            trackObjects{idObj}.Chips(:,:,end+1)= chip;
            trackObjects{idObj}.NormChips(:,:,end+1)= normalizedChip;
        else
            trackObjects{idObj}.Chips(:,:,1)= chip;
            trackObjects{idObj}.NormChips(:,:,1)= normalizedChip;
        end
    end

end
```

Input parameters: `trackObjects` contains the track data, `currentPing` is the sonar image of the last ping. The parameter `pingPingOutput` is the CFAR filtered image used to identify the events. `scansPerMeter` states how many cells that covers a meter in the range direction. `kalParam` is the structure array containing the Kalman parameters used for the tracking.
Output parameters: `trackObjects` where each track contains a chip of the state estimate in the last ping.

The function runs through every track stored in `trackObject` and extracts a chip from the sonar image of the last ping and a chip from the CFAR filtered sonar image `pingPingOutput`. Chips are only extracted if the track has a valid state estimate in the last ping. If the track was propagated, this will be indicated by a reduced track score for the track. Tracks being propagated suggest there where no events around the last estimate. Therefore, there is no reason to extract any chips.

4.3.2 Segmentation

The segmentation is done by region-based segmentation called region growing [8]. The chip is first low pass filtered using a adaptive noise removal filter. From each chip the highest amplitude is found. From the cell with the highest amplitude, the region grows appending all cells that are 8-connected[1] with a cell contained by the region and with an amplitude above a defined threshold. The region based segmentation is done by the function `region`. This function is called by the `featureExtract` which is described in the next section.

```
function [S] = region(chip)
% Segmetation of the input chip by using region growing segmentation
```

[1]Every cell that touches one of the edges or corners are considered neighbors.

```
% method. Returns segmented chip S containing one region.

%Finding dimension of chip
[noScans noBeams] = size(chip);
%Low pass filter the chip
fchip = wiener2(chip, [20 2]);
% %Finding threshold to extract the high precentile aplitudes in the image
[n xout] = hist(fchip,100);
threshold = xout(78);
%Eliminating the candidates that are not candidates by segmenting the image
BI =zeros(size(fchip));
for i = 1:numel(fchip)
    if (fchip(i)>threshold)
        BI(i)=1;
    end
end
%Seed matrix
S = zeros(size(chip));
%extracting center of chip to find seedpoint
centerChip = fchip(round(0.1*noScans):round(0.9*noScans),...
    round(0.1*noBeams):round(0.9*noBeams));
%Finding seed point. We know that it shold be around center of chip
[range beam] = find(fchip == max(centerChip(:)));

%Setting start point in seed matrix
S(range,beam)=1;

%Structuring element 8 connectivety
SE = strel('square',3);
%Loop for growing region. dialate from seedpoint and check if any
%connecting neighbours are within the thresholded difference.
% Setting start value for the while loop
 growing=1;
 while(growing)
    %Reset growing
     growing=0;
     %Save last S for comparison
     prevS = S;
     %Dialate the region in S by structureing element SE (8 connected)
     diaS= imdilate(S,SE);
     %Check if we have valid bins in the dialated region
     S = diaS.*BI;
     %If there has been a change in the region the loop will continue, if
     %the region is the same as last iteriation break the loop.
     if ¬isequal(prevS,S)
        growing =1;
     end
 end
```

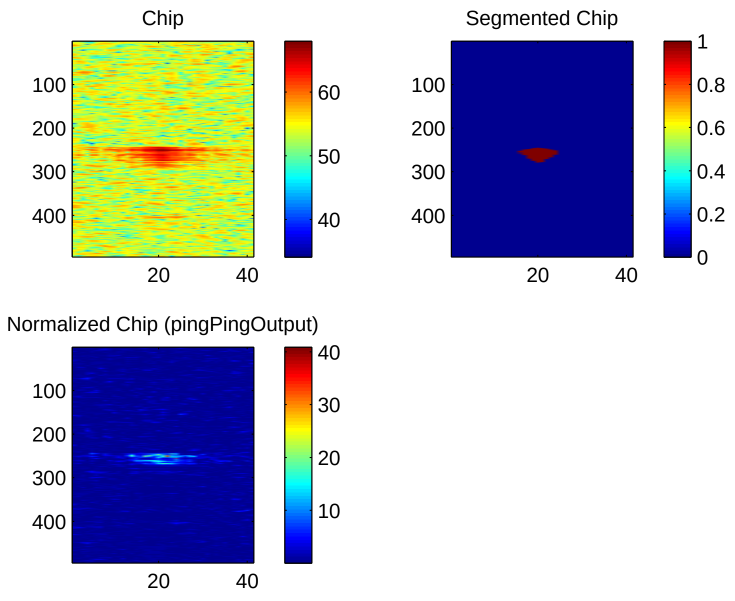

Figure 4.4: First plot is the chip extracted from the match filtered sonar image. Next is the chip extracted from the CFAR filtered image. The chip to the right is the segmented chip

Input parameters: `chip` is the extracted chip from the sonar image containing the tracked object.
Output parameters: `S` is a binary segmented version of the input chip containing one region defining the object.

The image is segmented by thresholding original image with a threshold set to a high percentile. Cells that have a amplitude that differs maximum 22% of the seed point are set to one, and cells whith amplitude below the threshold are set to zero. This percentage has shown good results on the data sets used in this thesis. For other data sets a different criteria may be more suited. The seed point is set by finding the cell with the highest amplitude. Since the low pass filter may have introduced some noise around the border of the image, the border cells are exluded as seed point candidates.

From the seed cell, the function checks if any neighboring cells meets the predicate (8-connected to region and amplitude above threshold). Any cells that meets these requirements are added to the region. The region continues to grow until there are no more candidate cells. The output image consist of one region defining the object. See Figure 4.4 for details.

4.3.3 Features

In each track, a chip containing the tracked object exists for every ping. It is from these chips the object features are extracted. This is done by the function `featureExtract`:

```
function [trackObjects] = featureExtract(trackObjects)
%Function for extraction static and temporal features for each track in
%input trackObjects.
```

Input parameters: `trackObjects` contains one or multiple tracks, each track having a sequence of extracted chips.
Output parameters: `trackObjects` is the input structure array with the static and temporal feature statistics added.

The function runs through every track in the input `trackObjects`. For every track there is a sequence of chips extracted from every ping in the track. Each chip is segmented to a binary version containing only the tracked object. This is done by the `region` function described in the last section. Using both the extracted chips and their segmented version, the static features are extracted.

4.3.3.1 Static Ping Features

The static features extracted from each chip is described in Section 3.3.2.1. As a reminder, the features can be seen in Table 4.3. The features *area, perimeter, eccentricity, major- and minor axis* are extracted using the MATLAB function `regionprops` which is included in the Image Processing Toolbox™ for MATLAB;

```
function [BWobject Stats] = extractRegionProps(bwImage)
%Label each region in binary image:
L = bwlabel(bwImage);
%Extract fetures with regionprops:
stats = regionprops(L,'Area','Perimeter','MajorAxisLength',...
    'MinorAxisLength','Eccentricity');
%find largest area
idx = find([stats.Area]== max([stats.Area]));
%if we have multiple objects of same size, chose one:
if(size(idx,2)>1)
    idx = idx(1);
end
%create image only containing largest object:
BWobject = ismember(L,idx);
%Save features for single object:
```

```
Stats = stats(idx);
end
```

Input parameters: `bwImage` is the binary segmented chip.
Output parameters: `BWobject` is the segmented chip containing one region. `stats` is a structure containing features extracted from the region.

The function labels each region in the input chip `bwImage`. The MATLAB function `regionprops` returns the features selected for each region. Usually the input image only contains one region, but if it by accident contains more than one region, `regionprops` returns statistics from every region. To guarantee that only the features from the object region is returned, the largest region in the image is chosen.

The features *mean*, *variance*, *compactness*, *first-* and *second order invariant moments*, have their own functions. They are all calculated as described in section 3.3.2.1 and are only described briefly;

```
function objectMean = extractMean(chip,BWobject)
```

Input parameters: `chip` is the chip extracted from the sonar image containing the object. The `BWobject` is the segmented chip. It defines which cells in `chip` to include for the mean amplitude calculation.
Output parameters: `objectMean` is a scalar with the mean amplitude of the object cells. It is calculated as defined in (3.38).

```
function objectVariance = extractVariance(chip,BWobject,objectMean)
```

Input parameters: `chip` is the extracted chip with `BWobject` as the segmented version. `objectMean` is the output from `extractMean` and is used for calculating the variance of the object cells
Output parameters: `objectVariance` is a scalar with the variance of the object cells. It is defined in (3.39).

```
function compact = extractCompactness(perimeter,area)
```

Input parameters: `perimeter` and `area` are features extracted by the function `extractRegionProps`.
Output parameters: `compact` is a scalar describing the compactness of the object. See (3.37).

Static features	
Area	The total number of cells in the segmented object
Perimeter	Every object cell that are a 4-connected with a background cell is considered a boundary cell. The perimeter is the length of all boundary cells.
Major axis	Length of the major axis of the smallest enclosing ellipse of the segmented object
Minor axis	Length of the minor axis of the smallest enclosing ellipse of the segmented object
Eccentricity	Eccentricity is the ratio of the distance between the foci of an ellipse and its major axis length. The eccentricity is calculated from an ellipse that has the same second order moment as the region.
Mean amplitude	This is the mean amplitude of the cells included in the region
Variance amplitude	This is the variance of the amplitude to the cells in the region
Compactness	The ratio between area and perimeter
The first and second invariant moment	Derived from the second order normalized central moments of the region

Table 4.3: Static features

```
function [invM1 invM2] = invMoments(chip)
```

Input parameters: `chip`
Output parameters: `invM1` and `invM2` are found using the equations (3.40) to (3.46). All functions are called from `featureExtract` and the output is stored in the structure array `trackObjects`. The static features are used to derive the temporal features.

4.3.3.2 Temporal Ping Features

The temporal feautures used are *mean*, *variance*, *mean rate of change* and *variance rate of change*. These temporal features are used on all the static features giving four temporal features for each static feature. Each static feature consists of a vector with feature measurements from every ping. Five feature measurements are used to produce a temporal feature measurement.

```
function [interFeatureMatrix] = extractInterScanFeatures(featureSet)
    %Splitt the featureSet into block of size N and compute interframe
    %measurements for each block
    N = 5;
    %Setting number of blocks with N samples. Overshooting samples
    %will be discarded.
    noOfBlocks = floor(size(featureSet,2)/N);
    startIndex = 1;
    %initializing arrays
    interMean   = zeros(1,noOfBlocks);
    interVar    = zeros(1,noOfBlocks);
    meanROC     = zeros(1,noOfBlocks);
    varROC      = zeros(1,noOfBlocks);

    for i = 1:noOfBlocks
        %Extract block of N samples from featureset
        blockSet = featureSet(startIndex:(startIndex + (N-1)));
        %inter scan mean and variance
        interMean(i) = mean(blockSet);
        interVar(i) = var(blockSet);

        %Mean Rate Of Change
        for j = 2:N
            meanROC(i) = meanROC(i) + (blockSet(j)-blockSet(j-1));
        end
        meanROC(i) = meanROC(i)/(N-1);
        %Mean Variance Of Change
        for j = 2:N
            varROC(i) = ...
                varROC(i) + ((blockSet(j)-blockSet(j-1))-meanROC(i))^2;
        end
        varROC(i) = varROC(i)/(N-1);
        %Set startindex for next block
        startIndex = startIndex + N;
    end
    interFeatureMatrix = [interMean',interVar',meanROC',varROC'];
end
```

Input parameters: `featureSet` is a vector containing all feature measurements from one static feature.
Output parameters: `interFeatureMatrix` is a matrix consisting of four vectors, one for each temporal feature.

4.4 Classification

The MATLAB toolbox *prTools*[6] is used for the classification. This toolbox has build in all the functions needed to test and evaluate the features described in this chapter.

4.4.1 Preparing the data set

In order to train the classifier the feature sets for the static and temporal features must be prepared to make a data set. All feature statistics are stored in the structure array `trackObjects`. The features are extracted to form a matrix on the form,

$$\overbrace{\begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,n} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ f_{m,1} & f_{m,2} & \cdots & f_{m,n} \end{bmatrix}}^{\text{features}}, \tag{4.2}$$

where each row represents a specific feature (e.g. area, perimeter). Four matrices are made:

1. Static features for diver objects
2. Temporal features for diver objects
3. Static features for marine life objects
4. Temporal features for marine life objects

These matrices are then combined to form two data sets, one for the static features containing samples from both classes and one for the temporal features. The two datasets are used to both train and test the classifier. Each of the data sets are split into two subsets; a training set and a test set. The training set consists of 1/3 of the samples and the remaining 2/3 are used for the test set.

4.4.2 Choice of classifier

The classification is done as described in Section 3.4. The linear discriminant function based on normal density distributions is used. The *prTools* toolbox uses a Gaussian linear discriminant function (`ldc`). The functions makes a quadratic problem linear as described in Section 3.4.2.2. The linear model is chosen because it has shown better results then a quadratic discriminant function in similar tests [4]. The simplicity of a linear model can also show better performance on new data, than a more sophisticated classifier specially designed to classify a specific data set.

4.4.3 Choice of features

The static feature set consists of 10 different features. Using the four temporal features on the static feature set gives a full temporal feature set of 40 features. Many of these features may not be able to distinguish the two classes or may be redundant when combined with other features. Features with poor separation ability may also act as noise to the classifier, making the classification results worse.

To reduce the complexity of the classifier and ensure optimal performance its desirable to have as few features as possible. Selecting only the best features that are able to distinguish between the classes and eliminating the redundant ones will ensure this.

A forward feature selection method i chosen. This method starts with a empty feature set and adds the feature with the best individual performance. All remaining features are tested in combination whit the first one and the pair with the best classification performance is chosen. The third feature is chosen in the same manner and this procedure is done until all features are selected.

4.5 Summary

In this chapter the implementation of the theory in Chapter 3 is presented. The main blocks in this chapter covers the detection of events in the sonar image, algorithm for tracking an object over time, a model for extracting features from the object and how to use these features to identify an object to a specific class.

The event detection method uses a temporal CFAR filter to detect events. The filter estimates the square difference between the variance of the cell and the temporal mean of the cell variance. Any cell with a difference above a specific threshold is considered a valid event.

A function for tracking events originated from an object is described. The tracking function can use either the Nearest Neighbor Kalman Filter or the Probabilistic Data Association Filter as a model for tracking. This choice is specified by the user. A detailed description of every part in the tracking algorithm is presented.

In order to extract the features needed for classification, a small image called a chip from every validated state in the track is extracted from the sonar image. This chip contains the a image of the reflected echo of the object being tracked. The chip is segmented into object and background cells in order to extract feature statistics from the object cells.

A set of static features are extracted from the chip. Taking statistics of each feature over multiple pings produces a second feature set of temporal features.

A linear discriminant function is presented as classifier. The features from both classes are combined to produce a data set for the static features and one for the temporal. These data sets are split into a training set and a test set. The training set is used for training the classifier and the test set is used to measure the performance.

To select an optimal feature set a forward feature search method is described. Each feature individual performance is measured and the one with the best performance is chosen for the first feature. Other features are tested in combination with this first and the pair with the best classification results is selected as the feature set. This is done until all features are chosen.

Chapter 5

Results and Discussion

In this section the results of the problem is presented. Classification results of both static and temporal feature sets will be shown and the different performance between the two feature set will be discussed. The individual performance of each feature will give rise to forming an optimal subset of features to give the least classification error.

5.1 Analysis of Variance

Analysis of Variance is a statistical procedure for comparing sample means. The One-Way ANOVA test is used to test mean between different groups are equal. By comparing the means of the different features between the classes, the ANOVA test gives a probability if the means comes from the same class or if the classes are in fact different. An ANOVA test of the features will give an indication of how well the different features are able to separate the two classes.

In Table 5.1 and Table 5.2 the F-number indicates "how different" the two classes are. A small number would indicate that the classes are the same and a large number would indicate a significant difference between the classes. The p-value shows the probability of observing the given F-number if the classes were truly equal.

The results of the ANOVA test indicates that the static features selected for classification shows a good ability to distinguish between the two classes. The difference between the temporal features seems to be a little smaller and therefor might not be as suitable for classification purposes.

Feature	F-number	Prob > F
Area	285.9	0
Perimeter	1493.6	0
Major Axis Length	2287.7	0
Minor Axis Length	322.3	0
Eccentricity	491.3	0
Mean	316.1	0
Variance	5.2	0.0229
Compactness	1216.2	0
First Invariant Moment	2425.1	0
Second Invariant Moment	766.1	0

Table 5.1: One-Way ANOVA test of the static features

5.2 Classification

The classification was done by using the *prTools* toolbox for MATLAB[6]. The classifier was trained by the function `ldc` which is a linear discriminant function. This type of classifier has been used in similar experiments and shown good results[3, 4].

5.2.1 Performance of classifier

The feature set containing both classes are divided into a training set and a test set. The training set is used to train the classifier and the test set is later used to test the classifier. One drawback of this method is that the both sets are very similar since they are taken randomly from the same feature set. This may lead to an over optimistic success rate, however the method still gives a good indication of how well the different features work.

5.2.1.1 Confusion matrix

A confusion matrix is a convenient way of displaying the performance of a classifier. It plots the the true class labels versus the labels classified by the classifier. A confusion matrix can in some cases give a better picture of the performance of the classifier than to only look at the error rate. For instance if the number of feature samples from each class are unbalanced a error rate will give little or no information on how effective the classifier is.

Table 5.3 shows the classification of 341 samples using the static features described in section 4.3.3.1. The classifier classifies 6 objects wrong. One diver event is classified as marine life and five marine life events are classified as divers.

Feature		F-number	Prob > F
Area	Temporal mean	78.2621	0.0000
	Temporal variance	0.0569	0.8120
	Mean ROC	1.2175	0.2726
	Variance ROC	0.2382	0.6266
Perimeter	Temporal mean	523.5723	0
	Temporal variance	0.1793	0.6729
	Mean ROC	1.3310	0.2515
	Variance ROC	0.4014	0.5279
Major Axis Length	Temporal mean	778.4026	0
	Temporal variance	4.4755	0.0370
	Mean ROC	2.0529	0.1552
	Variance ROC	4.2223	0.0426
Minor Axis Length	Temporal mean	129.0564	0
	Temporal variance	41.9407	0.0000
	Mean ROC	1.0711	0.3033
	Variance ROC	36.8563	0.0000
Eccentricity	Temporal mean	268.1971	0
	Temporal variance	47.7795	0.0000
	Mean ROC	0.0126	0.9110
	Variance ROC	36.2639	0.0000
Mean	Temporal mean	74.0701	0.0000
	Temporal variance	55.1243	0.0000
	Mean ROC	1.0669	0.3042
	Variance ROC	35.3636	0.0000
Variance	Temporal mean	4.4814	0.0369
	Temporal variance	20.7161	0.0000
	Mean ROC	0.3174	0.5745
	Variance ROC	16.5120	0.0001
Compactness	Temporal mean	564.1035	0
	Temporal variance	99.1252	0.0000
	Mean ROC	0.4876	0.4867
	Variance ROC	72.1760	0.0000
First Invariant Moment	Temporal mean	785.0002	0
	Temporal variance	12.9733	0.0005
	Mean ROC	1.8371	0.1785
	Variance ROC	11.8479	0.0009
Second Invariant Moment	Temporal mean	246.0793	0
	Temporal variance	10.6272	0.0015
	Mean ROC	2.0031	0.1602
	Variance ROC	9.5283	0.0026

Table 5.2: One-Way ANOVA test of the temporal features

	Estimated labels		
True labels	Diver	Marine life	Total
Diver	174	1	175
Marine life	5	161	166
Total	179	162	341

Table 5.3: Confusion matrix for classification using static features

	Estimated labels		
True labels	Diver	Marine life	Total
Diver	24	0	24
Marine life	0	24	24
Total	24	24	48

Table 5.4: Confusion matrix for classification using temporal features.

A classification using the temporal features from section 4.3.3.2 classifies all the objects correctly. See table 5.4.

5.2.2 Feature Evaluation

The evaluation of the features will look at the individual features ability to separate the two classes and which set of features that will give the highest classification rate. This is done for both the static and temporal feature sets.

5.2.2.1 Ranking of static features

The ranking of the static features are done by the MATLAB function `featrank` which is found the the prTools toolbox. This function ranks the features by their individual ability to correctly classify the samples. The ranking of the features can vary slightly depending on how the training and test are divided. To get a good estimate of the correct ranking, the average of 100 feature rankings have been done. Table 5.5 shows the static features sorted by their ability to separate the two classes. The temporal features are shown in table 5.6.

5.2.3 Number of Features

The more features used to train a classifier does not necessarily give a better results. Some features may just make the classification more complex. To find the optimum number of features, and which features to use can reduce the complexity of the classifier. There are several methods to find the best feature set. In this thesis a forward feature selection based on the individual feature ranking.

Rank	Static feature
1	Perimeter
2	Minor axis length
3	Major axis length
4	Second invariant moment
5	Area
6	First invariant moment
7	Compactness
8	Eccentricity
9	Mean
10	Variance

Table 5.5: Static feature ranking after 100 runs

Rank	Static feature
1	Variance Minor axis length
2	Variance rate of change Mean
3	Mean Minor axis length
4	Mean rate of change Perimeter
5	Mean rate of change Mean
6	Variance rate of change Compactness
7	Variance Mean
8	Variance Perimeter
9	Mean rate of change Eccentricity
10	Variance Eccentricity

Table 5.6: Temporal feature ranking

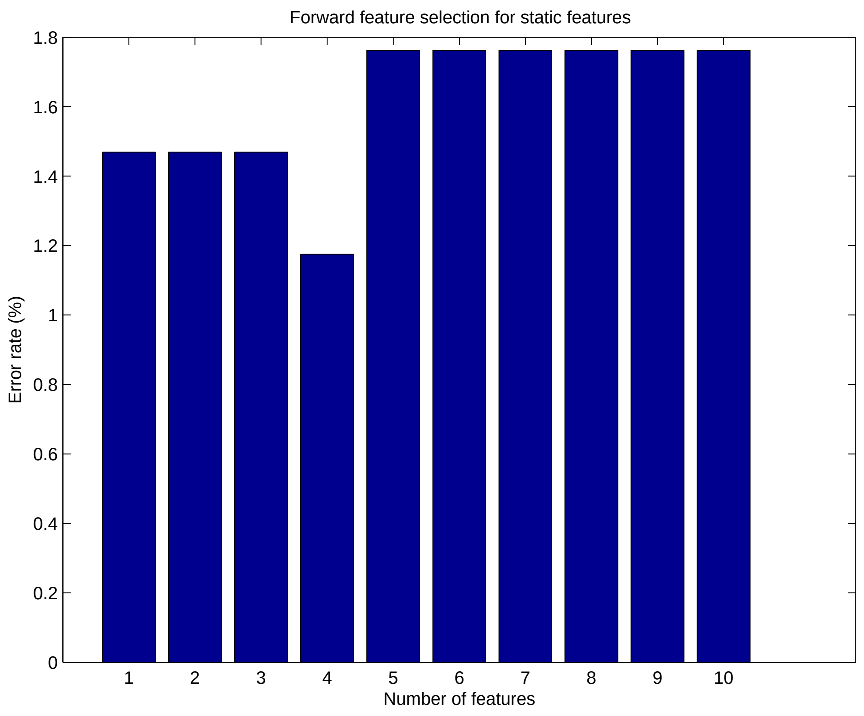

Figure 5.1: Classification error against number of features for static feature set

Starting with a empty feature set, features are incrementally added one at a time and the error rate is calculated. This method performs well when the optimal subset of features are small. A disadvantage of this method is that it does not discard any features added. This means that any features in the subset that becomes redundant when new features are added, will be kept. Because of the timespan of the thesis other feature selection methods has not been tested. The forward selection method was chosen because if its simplicity and that there was a small number of features to be tested. Other selection methods should be tested in the case of further work on this topic. Figure 5.1 shows the classification error rate for the static features using the forward selection routine. The plot shows that using only one feature gives a classification error of approximately 1.5%. Expanding the feature set to include four features reduces the error rate to about 1.18%. At a featureset of five and six features, the errror rate rises giving more misclassified samples. After five fetaures are added the error rate keeps stable at around 1.8%.

For the temporal features Figure 5.2 shows that for a feature set up to 19 features the classifier manages to correctly classify every object. Adding more features after this will only act as noise and lead to mis classifications.

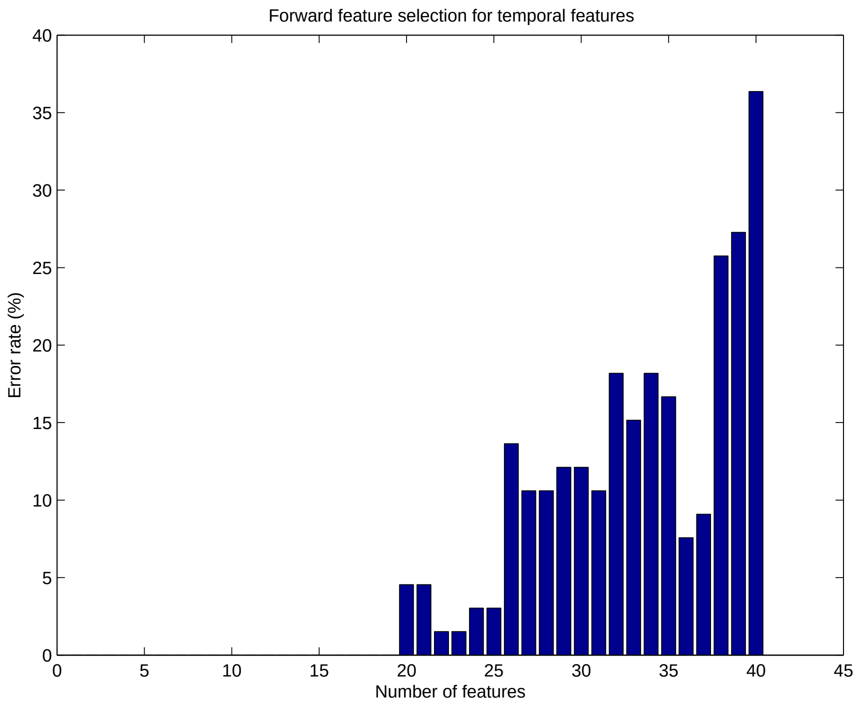

Figure 5.2: Classification error against number of features for temporal feature set

5.2.4 Error rate of diver and marine life

Looking at the error rate gives no indication on which samples are misclassified. Figures 5.3 and 5.4 shows the error rate specified by each class. As the plot shows, there are far more marine life objects that are misclassified than diver objects. For this evaluation subsets containing less then five features, no diver objects are misclassified. This does not mean that the results will be the same for another test set.

In a harbor protection system it will be more desirable to have marine life objects misclassified than diver objects. Classifying marine life objects as divers would result in *false alarms*, but if a diver object gets misclassified, a potential threat has been neglected. To ensure that all diver objects are correctly classified, a cost function [5] should be implemented.

Looking at Figures 5.3 and 5.4, the error rate for the marine life are much higher that that of the diver object. This distribution is much more preferable than a high numbered of misclassified diver objects.

5.2.5 Static versus Temporal Features

The results shows that the both the static and the temporal features manages to separate the two classes. The static features classifies with an error rate approximately between 1.2 - 1.8%. The temporal features shows an error rate of 0%. Because the temporal feature set is sparse than the static feature set, it hard to compare the two feature sets. A different test set and training set may show differen results.

As shown in Figure 5.2 the error rate is zero for the first 19 features. After 20 features the error rate increase dramatically, reaching a error rate of approximately 37% when the full feature set is used.

The classification using the static feature set with up to five features shows that only marine life objects are misclassified. When the feature set consists of more that four features both diver and marine life objects are misclassified. For the temporal features, features sets up to 20 features gives no misclassification for any of the two objects. As the feature set expands the complexity of the feature set gets higher resulting in an increase of misclassification. Using the full set of temporal features gives a error rate of almost 60% for marine life and approximately 18% for diver objects. For the static fetures, using the full feature set gives a error rate of around 3% for marine life objects and 0.5% for diver objects.

Using temporal features reduces the variance of the feature giving higher densities and a larger class separation. This can be seen in Figures 5.5 and 5.6. This shows scatter plot of the static features *perimeter* and *minor axis length* and the temporal mean of the same features. The straight line between the two classes are the linear discriminant function for a classifier using the for mentioned features.

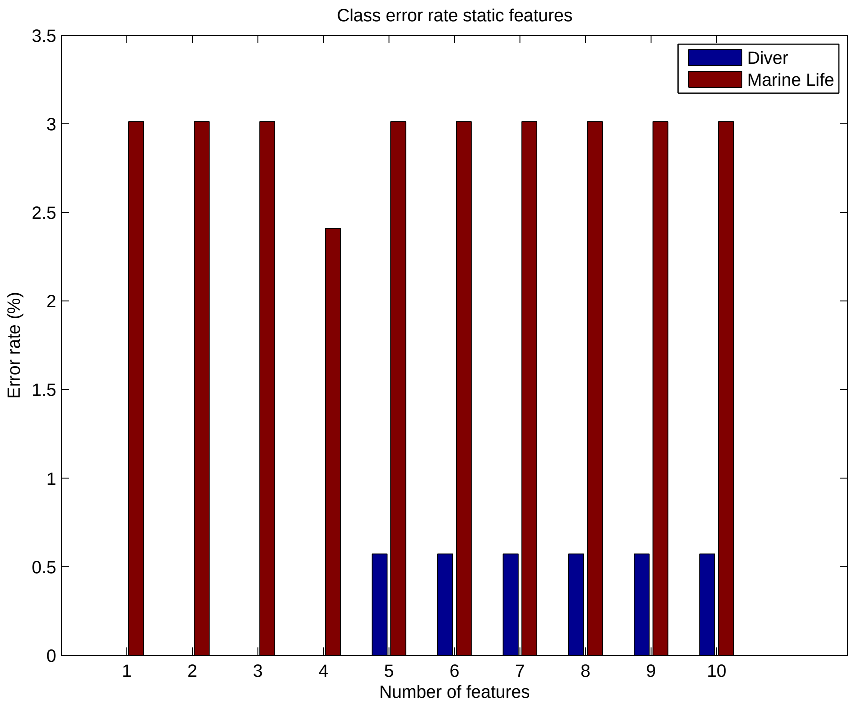

Figure 5.3: Classification error for each class for static features

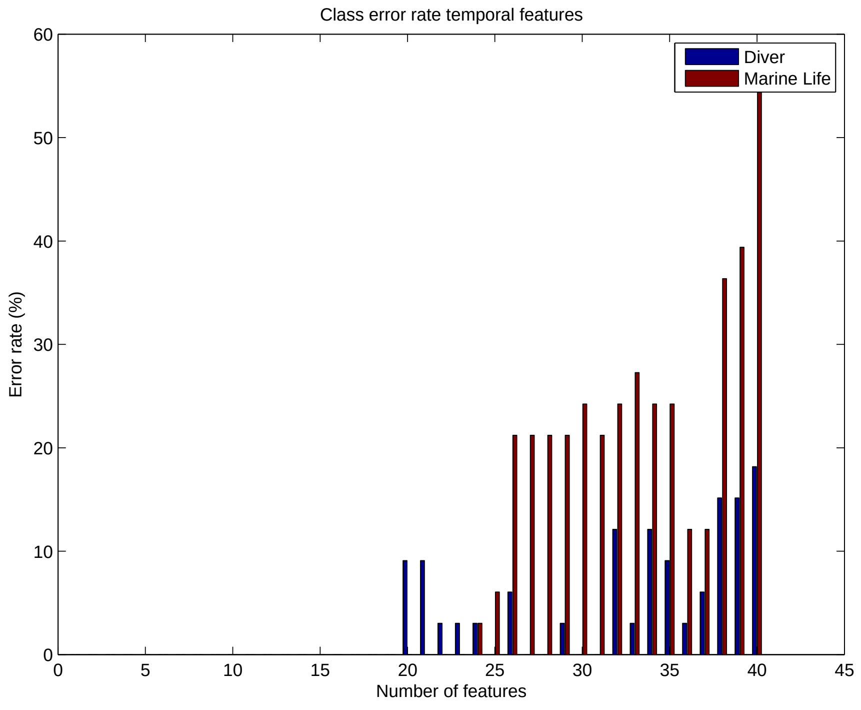

Figure 5.4: Classification error for each class for temporal features

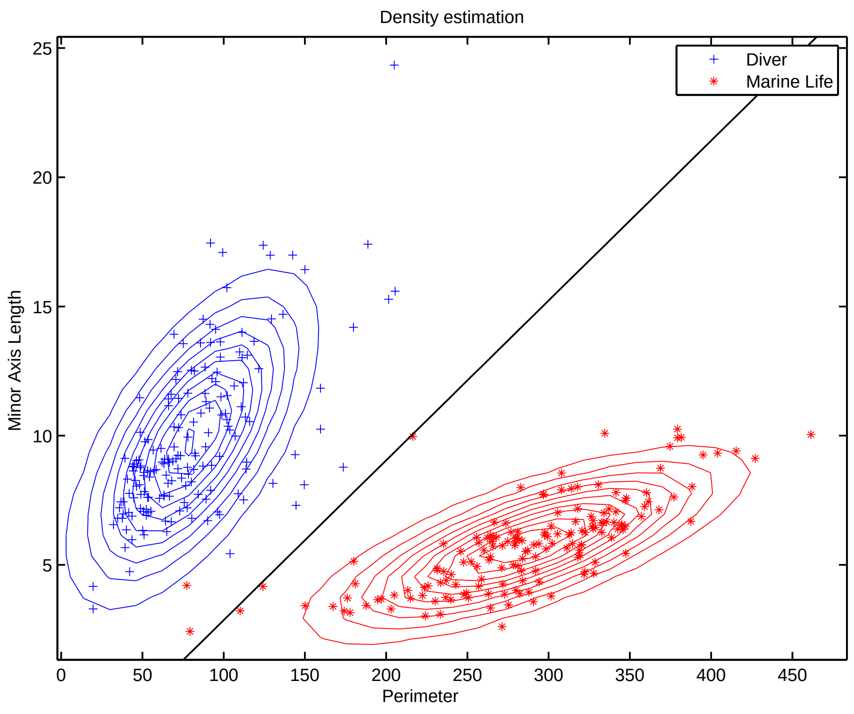

Figure 5.5: Scatter plot with density estimation

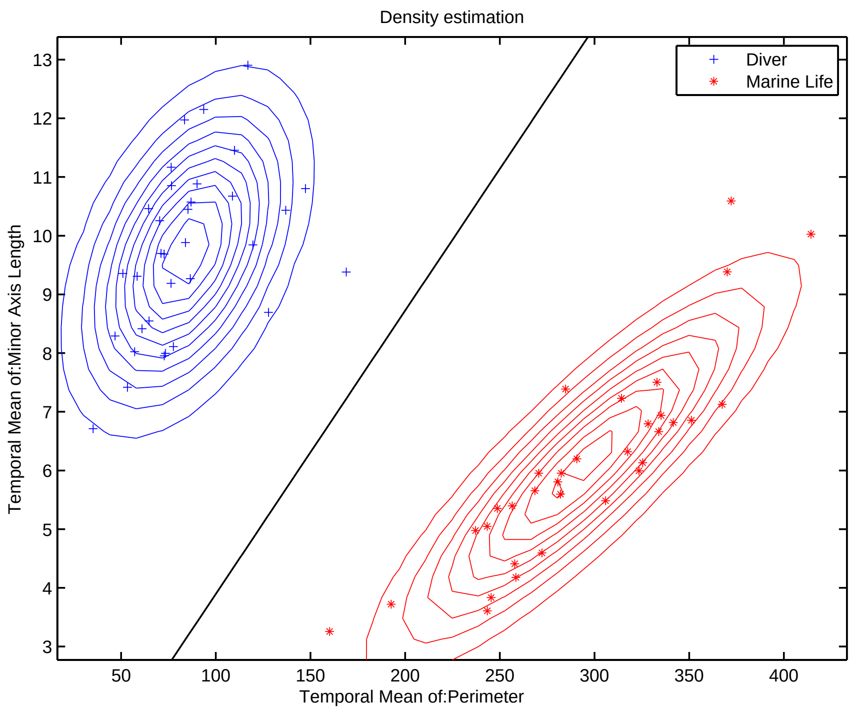

Figure 5.6: Scatter plot with density estimation

Chapter 6

Conclusion and Further Work

6.1 Conclusion

The detection and classification of underwater threats are essential in order to protect harbor infrastructure. The ability to efficient detect and correctly classify objects will reduce the workload of a sonar operator monitoring the sonar and increasing the reaction time to a potential threat.

By finding distinct features that can distinguish between different objects a simple classifier can be used to correctly classify objects of interest.

Features tested are static and temporal features.

The features tested in this thesis shows the capability to separate and correctly classify objects such as divers and marine life. The temporal features shows a greater ability than the static feature set, however the number of samples for the temporal features are limited so it is unknown how representative this data is. Using temporal features to classify gives a longer time before classification can be made because of the need of multiple pings to get the features. The static features may produce a correct classification faster but with a slightly increased chance of error.

A linear discriminant function seems to be a sufficient classifier for this task. Even though some objects will be misclassified a classifier more adapted to this set of samples may not perform any better or even worse than the linear discriminant function.

The tracking is used to collect the features samples. At this state it is not robust enough to follow an object for more than approximately 150 pings. The robustness is strong enough to collect the features when sections with known objects are present.

Thesis shows that the features tested are capable to classify the two classes. The ranking of the features may differ when tested on new data as well as the number of features needed to give good results may also vary depending on data set used.

6.2 Further work

The tracking algorithm can be optimized so it can hold the track longer than the current version can. System and measurement noise should be estimated and introduced to the algorithm.

Since the tracker has been used for feature gathering on ping sections with known objects, no new tracks are initiated after the first track initiation. This method works for feature gathering on a known data set, but for target detection on a live system, the tracker has to be continuously updated with new tracks, To use this tracker for target detection, every new event that is not associated with any existing tracks must trigger a new track.

The region growing function can sometimes spread the region over the entire chip if the chip contains a lot of noise. Some improvements to prevent this will be preferable.

Due to time restraints on this thesis, other feature selection methods have not been tested. Different types of feature selection methods may produce more optimal feature sets.

The classifier uses a linear discriminant function which is easy and robust. The classification may be improved with other types of classifications and should be tested.

List of Figures

List of Tables

Bibliography

[1] Kristina Alexander. Whales and sonar: Environmental exemptions for the navy's mid-frequency active sonar training. Technical report, Fort Belvoir, VA, 2008.

[2] Y. Bar-Shalom, F. Daum, and J. Huang. The probabilistic data association filter. *Control Systems Magazine, IEEE*, 29(6):82 –100, dec. 2009.

[3] M.J. Chantler and J.P. Stoner. Robust classification of sector-scan sonar image sequences. In *OCEANS '94. 'Oceans Engineering for Today's Technology and Tomorrow's Preservation.' Proceedings*, volume 2, pages II/591 –II/596 vol.2, 13-16 1994.

[4] M.J. Chantler and J.P. Stoner. Automatic interpretation of sonar image sequences using temporal feature measures. *Oceanic Engineering, IEEE Journal of*, 22(1):47 –56, jan 1997.

[5] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000.

[6] R.P.W Duin, P Juzczak, P. Paclik, E. Pekalska, D. de Riddler, D.M.J. Tax, and S. Verzakov. Prtools4.1, a matlab toolbox for pattern recognition. Delft University of Technology, 2007.

[7] P.P. Gandhi and S.A. Kassam. Optimality of the cell averaging cfar detector. *Information Theory, IEEE Transactions on*, 40(4):1226 –1228, jul 1994.

[8] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[9] B. Magaz and M.L. Bencheikh. Dsp implementation of a range azimuth cfar processor. pages 1 –4, may. 2008.

[10] VPS Naidu, G Girija, JR Raol, and Raj R Appavu. Data association and fusion algorithms for tracking in presence of measurement loss. In *Symposium*

on Modern Trends in Radar Technology, RADSYM-2002, volume 86, pages 17–28. IE(I), 2002.

[11] A. Rodningsby and Y. Bar-Shalom. Tracking of divers using a probabilistic data association filter with a bubble model. *Aerospace and Electronic Systems, IEEE Transactions on*, 45(3):1181 –1193, july 2009.

[12] I. Tena Ruiz, D.M. Lane, and M.J. Chantler. A comparison of inter-frame feature measures for robust object classification in sector scan sonar image sequences. *Oceanic Engineering, IEEE Journal of*, 24(4):458 –469, oct 1999.