NTNU
Norwegian University of
Science and Technology

Fusion between camera and lidar for autonomous surface vehicles

Vegard Kamsvåg

Master of Science in Cybernetics and Robotics

Submission date:	July 2018
Supervisor:	Edmund Førland Brekke, ITK
Co-supervisor:	Geir Hamre, DNV GL

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Dedicated to my parents, for their unconditional love and support.

Problem Description

Autonomous surface vehicles (ASVs) can use a variety of exteroceptive sensors for sense-and-avoid and navigation purposes. For the sake of redundancy and robustness the ASV should be equipped with both active (e.g., radar or lidar) and passive sensors (e.g., camera). While radar is required for larger vessels operating in open sea, lidar is a promising alternative for smaller ASVs operating in confined environments, both due to smaller size and higher resolution.

The purpose of this project is to develop a sensor fusion system for lidar and optical camera to be used for sense-and-avoid purposes onboard vehicles such as DNV GLs ReVolt scale-model or NTNUs autonomous ferry. The project builds on a 5th year specialization project written during the Autumn 2017, where data acquisition, registration and basic processing up towards measurement fusion were studied. The Masters thesis shall address the following tasks:

1. Installation of lidar and camera onboad test vehicle. DNV GLs ReVolt or Maritime Robotics Telemetron are possible candidates.

2. Calibration of camera.

3. ROS-based software architecture for data reception from the two sensors, synchronized with the inertial navigation system of the vehicle.

4. Formulation of suitable measurement models for data from the two exteroceptive sensors.

5. Exploration of a multi-target tracking method that performs measurement-level fusion using measurements from both lidar and camera. This may use existing implementations of JIPDA or MHT as its starting point.

6. Particular attention should be given to track initiation: examine to what extent two different sensors can give more reliable track initiation decisions than one sensor.

7. Analyze strengths and weaknesses of the two sensors with basis in tracking results.

Preface

This thesis marks the conclusion of the authors Masters degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). I would like to thank my supervisors Edmund F. Brekke at NTNU and Geir Hamre at DNV GL for their guidance, support and helpful and enlightening discussions during the work of this thesis. In addition to my supervisors, I would like to thank Tom Arne Pedersen and Rune Green, both DNV GL, for driving the target boats during the data gathering, and my good friend and fellow student Albert Havnegjerde for his assistance with both the data gathering as well as being a good discussion partner. I would also like to thank all the guys at the ITK workshop, for always being helpful and friendly, and lending me tools and equipment when needed.

The original plan for this thesis was to cooperate with the Trondheim-based company Maritime Robotics, and use their in-house developed 360 degree camera rig as a sensor. Due to a fire in their office spaces in early April where the camera rig was lost to the flames, the thesis was forced down a different path. While perhaps unfortunate, this just goes to show how uncertainty is something we have to live with, which is also a central theme in this thesis. In this regard, I would like to thank Marco Leonardi at NTNU for being so kind as to lending me his camera on short notice, such that this thesis could be completed.

This thesis is a continuation of the authors specialization project during the fall of 2017 [1]. The convolutional neural network used for detecting boats in images is a direct implementation of a network trained by Espen Tangstad during his masters thesis in the spring of 2017 (see section 4.5). The target tracking system is based on a unpublished MATLAB implementation of the JIPDA filter, provided by Edmund Brekke. The MATLAB implementation has been expanded upon by the author to include methods for track initiation and termination, and was implemented on real data gathered by the author. This thesis work ties together the thus far unrelated works of Tangstad, the tracking implementation, and the navigation system on the ReVolt model ship in a full target tracking pipeline. The derivations of the JPDA and JIPDA filters is partly based on unpublished lecture notes for a future course on sensor fusion at NTNU, and can be provided by Edmund Brekke upon request. This source is referenced as [2]. The sensor drivers and calibration software used are open-source ROS packages freely available online, and is not the authors original work. The author planned, organized and executed a series of experiments in a harbour environment, generating a data set which is not only used in this thesis, but hopefully will be valuable in future research as well.

Vegard Kamsvåg
Trondheim, July, 2018

Abstract

The development of autonomous surface vessels (ASVs) has seen great progress in the last few years, and are on the verge of becoming a reality. Sensing the environment in a reliable way is a key element in making a ship fully autonomous. The sensors needed to make a ship fully autonomous exist today, but the challenge remains to find the optimal way to combine them.

An ASV operating in a urban environment might need different exteroceptive sensors than a vessel operating at sea. Optical cameras and lidars (light detection and ranging) are suitable candidates for close-range sensing of the environment. Different sensors have different strengths and weaknesses, and in order to build a coherent world image on which e.g. sense-and-avoid decisions can be based on, information from the different sensors need to be fused and included into the state estimation of surrounding vessels. This is done using a target tracking system.

In this thesis, a target tracking framework based on the JIPDA filter is implemented and tested on real data gathered during a series of experiments in a harbour environment. Measurement models for both the lidar and the camera are formulated, and the sensors are geometrically calibrated and integrated with the navigation system onboard the ReVolt model ship. The data from the lidar are clustered using a slightly modified version of the DBSCAN algorithm. Sensor measurements from a number of different scenarios with two maneuvering targets are recorded, and the targets are tracked with the JIPDA filter using the lidar sensor as the primary sensor.

The results show that wakes behind the targets lead to many false tracks in close vicinity to the ReVolt model ship. The results also show that the detection probability of targets at range is reduced due to the spread of the laser beams. The presence of wakes did not lead to track loss for the true targets. At ranges where the targets were steadily detected, the targets were successfully tracked with few false tracks. It was also found that tracks can be lost due to occlusions, where one of the targets block the other target from being detected. The implemented Faster R-CNN detector showed limited range, where the detections at ranges over 20 meters are few and far between. At close ranges however, it shows the potential to be used in mitigating false tracks due to wakes or clutter, and could be used to aid in track formation and confirmation.

Sammendrag

Utviklingen av autonome overflatefartøy (ASV) har sett stor fremgang de siste årene, og er i ferd med bli en virkelighet. Sansing av omgivelsene på en pålitelig måte er et viktig element i å gjøre skip fullstendig autonome. Sensorene som skal til for å gjøre skip autonome fins på markedet i dag, utfordringen som gjenstår er å kombinere disse på en optimal måte.

Et ASV som opererer i urbane omgivelser kan ha behov for andre eksteroseptive sensorer enn et skip som opererer til havs. Optiske kameraer og lidarer (light detection and ranging, lys-deteksjon og avstandsmåling) er lovende kandidater for sansing av omgivelsene på nært hold. Forskjellige sensorer har forskjellige styrker og svakheter, og for å kunne generere et sammenhengende verdensbilde som sanse-og-unngå-avgjørelser kan baseres på må informasjonen fra de forskjellige sensorene fusjoneres og inkluderes i tilstandsestimeringen til fartøy i omgivelsene. Dette gjøres ved å bruke et målfølgingssystem.

I denne oppgaven er et målfølgingssystem basert på JIPDA-filteret implementert og testet på reelle data samlet inn under en rekke eksperimenter gjennomført i et havnemiljø. Målemodeller for både lidaren og kameraet er formulert, og sensorene er geometrisk kalibrert og integrert i det eksisterende navigasjonssystemet på ReVolt skalamodellen. Dataene fra lidaren klynges sammen ved hjelp av en lettere modifisert versjon av DBSCAN-algoritmen. Sensordata fra en rekke forskjellige scenarioer med to manøvrerende mål er tatt opp, og målene følges ved hjelp av JIPDA-filteret med lidaren som primær sensor.

Resultatene viser at bølger i kjølvannet bak målene fører til mange falske spor fra målfølgingssystemet i nærheten til ReVolt. Resultatene viser også at deteksjonssannsynligheten på større avstander reduseres på grunn av spredingen av laserstrålene til lidaren. Kjølvannsbølger fra målene førte ikke til at målfølgingssystemet mistet målene. På avstander hvor målene ble pålitelig detektert greide systemet å spore målene, med få falske spor. Det viste seg også at spor kan mistes på grunn av skygging, hvor det ene målet skygger for det andre og med det hindrer det fra å bli sett. Det implementerte Faster R-CNN nettverket for å detektere båter i bilder viste seg å ha begrenset rekkevidde, med få deteksjoner på avstander over 20 meter. På nært hold viser det derimot potensial til å kunne brukes til å motvirke falske spor på grunn av bølger fra kjølvann eller annen støy, og kan bidra i sporformasjon og sporbekreftelse.

Table of Contents

Abbreviations

ASV	=	Autonomous Surface Vessel
CCD	=	Charge-Coupled Device
CPU	=	Central Processing Unit
CNN	=	Convolutional Neural Network
VOC	=	Visual Object Challenge
EKF	=	Extended Kalman Filter
GPU	=	Graphical Processing Unit
RAM	=	Random Access Memory
GNSS	=	Global Navigation Satellite System
RTK	=	Real-Time Kinematic
INS	=	Inertial Navigation System
NED	=	North-East-Down
WGS	=	World Geodetic System
PDAF	=	Probabilistic Data Association Filter
JPDA	=	Joint Probabilistic Data Association
IPDA	=	Integrated Probabilistic Data Association
JIPDA	=	Joint Integrated Probabilistic Data Association
pdf	=	Probability Density Function
ROS	=	Robot Operating System
RPN	=	Region Proposal Network
RANSAC	=	RANdom SAmple Consensus
DBSCAN	=	Density Based Spatial Clustering of Applications with Noise
TCP	=	Transmission Control Protocol
IP	=	Internet Protocol
I/O	=	Input/Output
PoE	=	Power over Ethernet
UDP	=	User Datagram Protocol
SDK	=	Software Development Kit

Notation

Frequently Used Notation

$\mathcal{R}_a^b$	=	Rotation matrix from frame a to frame b
$\mathcal{T}_a^b$	=	Homogeneous transformation matrix from frame a to frame b
$\mathbf{t}_a^b$	=	Translation vector from frame a to frame b
$\mathbf{C}$	=	Bold capital letter represents a matrix
$\mathbf{x}$	=	Bold lowercase letter represents a vector
χ^2	=	Chi-square distribution
$\mathbf{K}$	=	Camera intrinsic calibration matrix

Estimation and Tracking Notation

$p(x)$	=	Probability density function (pdf) for the random variable x
$p(x \mid y)$	=	Pdf for x conditioned on y
$p(x, y)$	=	Joint pdf for x and y
$P\{\cdot\}$	=	Probability
$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	=	Normal distribution for random variable $\mathbf{x}$, with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$
$\bar{\mathbf{x}}_k$	=	Prior state estimate
$\hat{\mathbf{x}}_k$	=	Posterior state estimate
$\bar{\mathbf{z}}_k$	=	Measurement prediction
$\mathbf{x}_{0:k}$	=	State sequence from time step 0 to time step k
$\mathbf{z}_k$	=	Measurement
$\mathbf{z}_{k,i}$	=	Measurement i
$\mathbf{z}_{k,l}$	=	Lidar measurement
$\mathbf{z}_{k,c}$	=	Camera measurement
$\mathbf{Z}_k$	=	Measurement set at time step k
$\mathbf{Z}_{1:k}$	=	Cumulative measurement set up until time step k
$\bar{\mathbf{P}}_k$	=	Prior state covariance matrix
$\hat{\mathbf{P}}_k$	=	Posterior state covariance matrix
$\hat{\mathbf{P}}_k^c$	=	Covariance of the state updated with the correct measurement
$\tilde{\mathbf{P}}_k$	=	Spread of the innovations
$\mathbf{S}_k$	=	Innovation covariance matrix
$\mathbf{W}_k$	=	Kalman gain
$\boldsymbol{\nu}_k$	=	Combined innovation
$\boldsymbol{\nu}_{i,k}$	=	Innovation for measurement i
γ	=	Validation gate threshold
$\mathcal{V}(k, \gamma)$	=	Validation gate, with gate parameter γ

m_k	=	Number of validated measurements
$\mu_F(\cdot)$	=	Probability mass function
φ	=	Number of false measurements in validation region
λ	=	Poisson distribution intensity
$\beta_{i,k}$	=	Association probability for measurement i
$\beta_{i,k}^t$	=	Marginal association probability for measurement i to track t
$p_k^t(\mathbf{x}_k^t)$	=	Posterior Gaussian approximation for track t
V	=	Volume
P_D	=	Probability of detection
P_G	=	Gating probability
$\boldsymbol{\tau}$	=	Vector containing individual track-wise detections
$l^{t,a_k(t)}$	=	Likelihood ratio
$\hat{\mathbf{x}}_k^{t,a_k(t)}$	=	Posterior event-conditional state
$\hat{\mathbf{P}}_k^{t,a_k(t)}$	=	Posterior event-conditional covariance
$\bar{\varepsilon}_k^t$	=	Prior existence probability for target t
ε_k^t	=	Posterior existence probability for target t
$\bar{\eta}_k^t$	=	Prior visibility probability for target t
η_k^t	=	Posterior visibility probability for target t
p_{ij}^{ε}	=	Markov chain coefficient for target existence
p_{ij}^{η}	=	Markov chain coefficient for target visibility
$\mathbf{a}_k$	=	Association hypothesis
$\neg\mathbf{Z}_k$	=	Set of unassociated measurements
$\mathbf{w}_k$	=	Process noise
$\mathbf{v}_k$	=	Measurement noise
$\mathbf{v}_{k,l}$	=	Lidar measurement noise
$\mathbf{v}_{k,c}$	=	Camera measurement noise
$\mathbf{R}_k$	=	Measurement noise covariance matrix
$\mathbf{R}_l$	=	Lidar measurement noise covariance matrix
$\mathbf{R}_c$	=	Camera measurement noise covariance matrix
$\mathbf{Q}_k$	=	Process noise covariance matrix
$\mathbf{F}_k$	=	State transition matrix
$\mathbf{F}_{cv}$	=	Constant velocity model state transition matrix
$\mathbf{H}_k$	=	Measurement matrix
$\mathbf{H}_l$	=	Lidar measurement matrix
$\mathbf{h}(\mathbf{x}_k)$	=	Nonlinear measurement function
$\mathbf{h}_c(\mathbf{x}_k)$	=	Camera measurement function
σ_{cv}	=	Constant velocity model process noise strength
σ_l^2	=	Lidar measurement noise variance
v_{max}	=	Velocity threshold used in track formation
d_L	=	Distance threshold used in track merging
T	=	Sample time

Chapter 1

Introduction

1.1 Motivation

The development of autonomous vehicles has seen great progress in the last decade. With advances in technologies enabling perception of the surrounding environments, path planning and real-time vehicle control, in combination with sophisticated sensors and increasing data processing performance, full vehicular autonomy is within grasp in the immediate future [3]. The sensor technologies needed to make autonomous ships a reality exists today. The challenge remains to find the optimal way to combine them reliably and cost effectively.

Autonomous surface vehicles (ASVs) operating in urban environments, e.g. ferries, might need slightly different exteroceptive sensors than ASVs operating in the open sea. A lidar with a range of 100 meters may be more appropriate than a maritime radar with range of several kilometers. Furthermore, the complexity of the environment means that the rich information from optical cameras will be more important. In order to build a coherent world image, which, e.g., collision avoidance decisions can be based on, the data from these sensors must be fused, together with data from interoceptive sensor systems such as an inertial navigation system (INS). Koch [4] describes sensor fusion as the process of combining incomplete and imperfect pieces of information in such a way that a better understanding of a underlying real-world phenomenon is achieved.

1.1.1 The ReVolt Project

The ReVolt is a shipping concept developed by the classification society DNV GL. The ReVolt aim at solving the growing need for transport capacity, by moving freight from the increasingly stressed land based logistic networks, to the sea. The ReVolt vessel itself is envisioned as an 60 metres long autonomous, battery powered short-sea cargo freighter with a range of 100 nautical miles and a cargo capacity of 100 twenty-foot containers, operating at a speed of 6 knots. For the purpose of testing the autonomous capabilities of ReVolt, a 1:20 scaled model has been built. During previous student work, the scale model has been outfitted with dynamic positioning (DP) capabilities [5], and the aim of this thesis is to develop a sensor fusion system for lidar and optical camera to be used for sense-and-avoid purposes on board the ReVolt scale model. Parallel to the work described in this thesis, other students are working on a remote control station, guidance and path following, as well as development of a digital twin for the ReVolt in their thesis work. The ReVolt scale model is shown in figure 1.1.

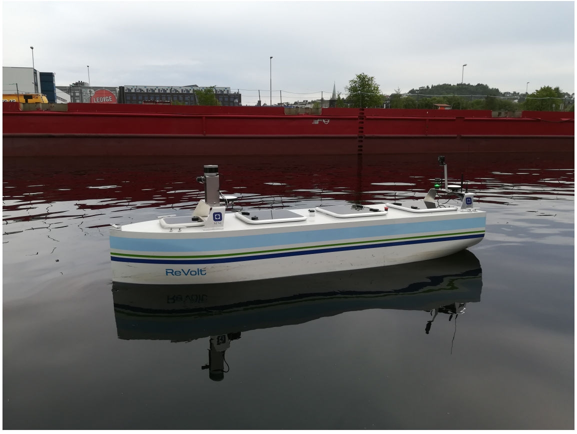


Figure 1.1: The ReVolt model ship.

1.2 Review of Previous Work

Substantial research has been conducted into the field of remote sensing and data fusion with applications for autonomous vehicles, with a particular drive from the auto industry. Some examples of different approaches are given below.

- Stiller et al. [6] in 2000 proposed a multisensor concept with a variety of different sensor technologies with widely overlapping fields of view for an autonomous, unsupervised vehicle. They used stereo vision, laser scanners, radar, and short range radar, combined by sensor fusion into a joint obstacle map. The research was conducted as a part of the German project *Autonomes Fahren*.

- Mälich et al. [7] used low-level fusion of multibeam lidar and vision sensor into a detection and tracking framework. They used a cascaded AdaBoost (adaptive boosting [8]) detector based on haar-wavelet like features [9] for the vision system. Lidar returns were used to generate regions of interest in the images.

- Aufrere et al. [10] at the NavLab group at Carnegie Mellon University proposed a high level fusion approach for object tracking using cameras and lidars for autonomous vehicles in cluttered urban environments. Their approach used a map-based fusion system, with a probability-based predictive model.

- Cho et al. [11] developed a multi-sensor system for moving object detection and tracking, building on the work of Aufrere et al. for the feature extraction for the lidar. Their approach fuses vision, lidar and radar. Detection in the vision module is represented as bounding boxes, and the data from the sensors were fused in an extended Kalman filter. The system detects and tracks pedestrians, bicyclists, and vehicles.

- Premebida et al. [12] demonstrated a perception system for pedestrian detection in urban scenarios using information from lidar and a single camera. Two sensor fusion architectures are described in their paper. A centralized architecture, where the fusion is done at the feature level, i.e. features from lidar and vision space combined in a single vector for posterior classification using a single classifier. The decentralized architecture employs two classifiers, one per sensor feature-space, fused by a trainable fusion method applied over the likelihoods provided by the component classifiers. They showed that the trainable fusion method lead to enhanced detection performance, and maintenance of false-alarms under tolerable values in comparison with single-based classifiers.

- Weigel et al. [13] demonstrated a vehicle tracking and lane detection multi-sensor system, using a lidar and a monocular camera. Detected vehicles are tracked and managed by a multi-object extended Kalman filter using the data from the lidar and the camera. The lidar was used to create appropriate regions of interest in the image plane, and subsequently the measurements in the image plane was incorporated into the Kalman filter.

There has also been done substantial research towards making autonomous vessels possible. Two examples which focus on the sensing and perception aspect of such systems are given below.

- Wolf et al. [14] describe the perception and planning systems of an autonomous surface vehicle with the goal to detect and track other vessels at medium to long ranges. They employ a NASA JPL developed tightly integrated system termed CARACaS (Control Architecture for Robotic Agent Command and Sensing) that blends the sensing, planning and behaviour autonomy necessary for such missions. In their paper, they presents an autonomy system that detects and tracks vessels of a defined class while patrolling near fixed assets. The sensor suite includes a wide-baseline stereo vision system for close-up perception and navigation, and a 360 degree camera head for longer range detection, identification, and tracking. The perception system termed SAVAnT (Surface Autonomous Visual Analysis and Tracking) receives sensory input from 6 cameras, stabilized by INS pose, detects objects of interest and calculates absolute bearings for each contact.

- Elkins et al. [15] published a paper on the Autonomous Maritime Navigation project, with the stated goal of creating a set of sensors, hardware, and software that enables autonomy on unmanned surface vehicle (USV) platforms. The sensor suite includes cameras, radar, lidar, compass and GPS, integrated into a sensor fusion engine. Fusion algorithms are used to compile and correlate these data into a common tactical picture for the USV. In their paper, they showed the benefits of using a lidar for close-range detections over more long-range sensors such as radars. They also argued that the wavelength of the lasers in the lidar is such that the radiated energy does not reflect well of the water surface, i.e. most points returned are not water, making it well suited in the detection of obstacles.

Previous research into fusion of camera and lidar show that the two sensors have complimentary characteristics, where the rich information in optical images make the camera suitable for object detection and classification, and the point cloud from the lidar gives a 3D situational overview of the surroundings. This makes the lidar and the camera good candidates to be used in a perception system for an autonomous vessel. The results presented by Elkins et al. show that a lidar is well suited for close-range sensing and detection for an ASV operating in urban environments.

1.3 Contributions

The main contributions of this thesis are listed below.

- Measurement models for the lidar as well as the camera are formulated, and the sensors are geometrically calibrated. The measurements are transformed to a common world frame by integration with the navigation system on board the ReVolt model ship.

- The installation and integration into the existing control system of a camera and a lidar on the ReVolt model ship.
- Implementation of a tracking system based on the JIPDA filter using real data gathered at sea. Although being given, the JIPDA implementation was expanded to include track formation, confirmation and termination capabilities, as well as computing association probabilities for image detections in addition to lidar measurements.
- A series of comprehensive experiments using real targets has been planned, organized and executed at sea using the ReVolt model ship as a sensor platform. The resulting data set has potential to be used in future research projects both for the ReVolt as well as in other projects at NTNU.

1.4 Thesis Outline

The structure of this thesis is as follows; In chapter two, a theoretical description of the two sensors used is given. The pinhole camera model is presented, and subsequently expanded upon to give the model of a CCD camera. A brief introduction to convolutional neural nets for image detection and classification is given, and the lidar sensor is described at the end of the chapter.

Chapter three deals with state estimation and target tracking. First, a brief primer on probability is given, and the workhorse of state estimation, the Kalman filter as well as the extended Kalman filter is presented. The target tracking theory first introduces the Probabilistic Data Association Filter (PDAF), before a brief description of the extension of the PDAF to multi-target tracking in the Joint Probabilistic Data Association (JPDA) filter, and finally the Joint Integrated Probabilistic Data Association (JIPDA) which is used in this thesis is presented.

Chapter four describes the physical implementation of the sensors on the ReVolt, as well as the calibration procedures performed. The different coordinate systems, and the transformations between them is derived. The practical implementation of the Faster R-CNN framework is explained as well as the segmentation and clustering of the lidar point cloud. At the end of chapter four the implemented target tracking framework is described, including the motion and sensor models, as well as some practical considerations with regard to the implementation of the JIPDA filter.

Chapter five presents and discusses the results of the target tracking framework from chapter four on a set of real-world data gathered by the author in the Dora harbour basin.

Chapter six concludes this thesis, and gives suggestions for future research areas to follow given the results presented.

Chapter 2

Sensors and Detection Fundamentals

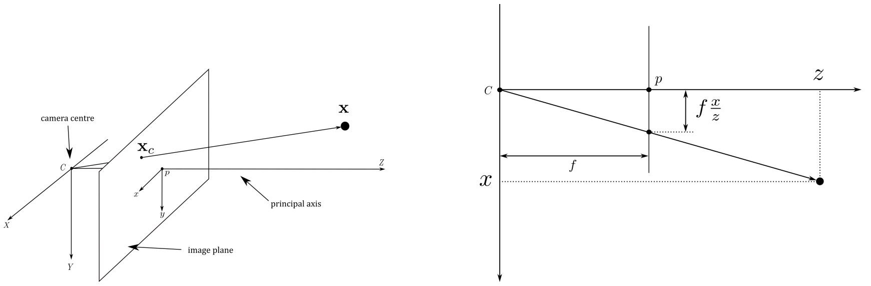
The aim of this chapter is to introduce the theory and principles behind the sensors used in this thesis, as well as to give an overview of the object detection process for the camera. This thesis employs a CCD camera and a 360° lidar as exteroceptive sensors to be fused within a object tracking system. Object tracking refers to the problem of determining the location, path and characteristics of objects of interest by using sensor measurements. This thesis uses a detection-based tracking approach, and thus the tracking process relies on detection results in order to be able to maintain an estimate of the state of a target. The detection problem can be summarized as the process of recognizing the presence of a target object within the sensor data, and object detection and classification for images will be discussed.

2.1 Camera Model

A camera can be viewed as a mapping between the 3D world and a 2D image. This mapping can be represented by a camera model, which is a matrix with certain properties. This section first introduces the basic pinhole camera, before generalizing this model to CCD (Charge-coupled device) sensors found in digital cameras. A model for the radial distortion typically found in cameras with real (non-ideal) lenses is presented, and a measurement model associating world points with its corresponding pixel coordinates is given at the end. The theory presented in this section is largely based on [16].

2.1.1 The Pinhole Camera Model

The basic pinhole camera model is a *central projection* of points in space onto a plane. Let the centre of projection be the origin of a Euclidean coordinate system, and let the *image plane* be the plane $z = f$. The line from the camera centre (also called the optical centre) perpendicular to the image plane is called the *principal axis* or *principal ray* of the camera. This ray meets the image plane in the *principal point*. A point in space with coordinates $\mathbf{x} = [x, y, z]^T$ is mapped to a point on the image plane where a line joining the centre of projection (the origin) and the point $\mathbf{x}$ meets the image plane. This is illustrated in figure 2.1a. As figure 2.1b illustrates by similar triangles, the point $\mathbf{x}$ in 3D space maps to the point $\mathbf{x}_c = \left[f\frac{x}{z}, f\frac{y}{z}, f\right]^T$ in the image plane. The final image coordinate is a constant, and ignoring this coordinate we see that



(a) The pinhole camera model geometry.

(b) The mapping of points in space to the image plane by similar triangles.

Figure 2.1: The pinhole camera model. The figures are adapted from [16].

$$\begin{bmatrix} x & y & z \end{bmatrix}^T \mapsto \begin{bmatrix} f\frac{x}{z} & f\frac{y}{z} \end{bmatrix}^T \tag{2.1}$$

describes the central projection mapping of the 3D world coordinates in $\mathbb{R}^3$ to the image plane coordinates in $\mathbb{R}^2$. Using homogeneous coordinates, this mapping can conveniently be represented by matrix product as

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} fx \\ fy \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \text{diag}(f, f, 1) \begin{bmatrix} \mathbf{I}^{3\times 3} & \mathbf{0}^{3\times 1} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.2}$$

where $\text{diag}(f, f, 1)$ is a diagonal matrix, $\mathbf{I}^{3\times 3}$ is the 3×3 identity matrix, and $\mathbf{0}^{3\times 1}$ is a 3×1 zero vector. We use the notation $\mathbf{x}$ for the world point given by the homogeneous coordinate vector $[x, y, z, 1]^T$, $\mathbf{x}_c$ for the corresponding point in the image plane represented by a homogeneous 3-vector, and $\mathbf{M}$ for the homogeneous 3×4 camera projection matrix, (2.2) can be written as

$$\mathbf{x}_c = \mathbf{M}\mathbf{x} \tag{2.3}$$

where

$$\mathbf{M} \triangleq \text{diag}(f, f, 1) \begin{bmatrix} \mathbf{I}^{3\times 3} & \mathbf{0}^{3\times 1} \end{bmatrix}. \tag{2.4}$$

CCD Cameras

The mapping given in (2.2) assumes that the origin of the image plane coordinates coincides with the principal point. In practice it may not, so the mapping between world and image plane coordinates is augmented as

$$\begin{bmatrix} x & y & z \end{bmatrix}^T \mapsto \begin{bmatrix} f\frac{x}{z} + p_x & f\frac{y}{z} + p_y \end{bmatrix}^T \tag{2.5}$$

where $[p_x, p_y]^T$ are the coordinates of the principal point. In homogeneous coordinates, the mapping may now be written as

$$\begin{bmatrix} fx + zp_x \\ fy + zp_y \\ z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} \mathbf{I}^{3\times3} & \mathbf{0}^{3\times1} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.6}$$

where

$$\mathbf{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.7}$$

is the *camera calibration matrix*. Moreover, in a camera where the sensor is a CCD array, the pixels may not be square. When the image coordinates are measured in pixels, the non-square pixels introduce unequal scale factors in both axial directions. Defining the number of pixels per unit distance in image coordinates as m_x and m_y in the x and y directions, the homogeneous mapping between world and image coordinates is obtained by premultiplying (2.7) with the factor diag$(m_x, m_y, 1)$, such that the general form of a calibration matrix for a CCD camera is [16]

$$\mathbf{K} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

where $\alpha_x = fm_x$ and $\alpha_y = fm_y$ represent the focal length of the camera in pixel units in the x and y dimensions, respectively, and $x_0 = m_x p_x$ and $y_0 = m_y p_y$ represent the coordinates of the principal point in terms of pixel dimensions. The parameters of the matrix $\mathbf{K}$ are known as the *intrinsic* camera parameters, and are typically found through a calibration procedure as described in [17].

Camera Coordinates and Rigid Transformations

Homogeneous coordinates are convenient for representing geometric transformations by a matrix product. Consider a point $\mathbf{x}$ in some right-handed coordinate frame. The nonhomogeneous coordinate vector $\mathbf{x}$ is the 3-vector $[x, y, z]^T$, while its homogeneous counterpart is the 4-vector $[x, y, z, 1]^T$. The change of coordinates between two (arbitrary) Euclidean coordinate systems a and b can be represented by a rotation matrix $\mathcal{R}_b^a$ and a translation vector $\mathbf{t}_b^a$ in $\mathbb{R}^3$ as

$$\mathbf{x}^a = \mathcal{R}_b^a \mathbf{x}^b + \mathbf{t}_b^a \tag{2.9}$$

Using homogeneous coordinates, the same transformation can be written as

$$\mathbf{x}^a = \mathcal{T}_b^a \mathbf{x}^b, \quad \text{where} \quad \mathcal{T}_b^a = \begin{bmatrix} \mathcal{R}_b^a & \mathbf{t}_b^a \\ \mathbf{0}^{1\times 3} & 1 \end{bmatrix} \tag{2.10}$$

where $\mathbf{x}^a$ and $\mathbf{x}^b$ are now 4-vectors. $\mathbf{0}^{1\times 3}$ denotes the 1×3 zero vector. Thus far, the world coordinates discussed in the previous section have been assumed given in the camera coordinate frame, where the origin is located in the optical centre and with the principal axis of the camera pointing down the positive z-axis. Generally points in space will be expressed in terms of a different Euclidean coordinate frame, the *world coordinate frame*. The camera coordinate frame and the world coordinate frame are related via a rotation matrix $\mathcal{R}$ and translation vector $\mathbf{t}$ such that

$$\mathbf{x}_{cam} = \mathcal{R}\mathbf{x}_{world} + \mathbf{t}. \tag{2.11}$$

Combining (2.11) and (2.6), the *camera matrix* relating a point represented in the world coordinates with pixel coordinates can be written as

$$\mathbf{M} = \mathbf{K}\begin{bmatrix} \mathcal{R} & \mathbf{t} \end{bmatrix}. \tag{2.12}$$

The parameters of $\mathcal{R}$ and $\mathbf{t}$ relating the camera orientation to the world coordinate frame are called the *extrinsic parameters*. The camera matrix in (2.12) has 10 degrees of freedom; 4 from the camera calibration matrix $\mathbf{K}$, 3 for $\mathcal{R}$ and 3 for the translation $\mathbf{t}$. There are several ways to parameterize the rotation matrix $\mathcal{R}$, such as by Euler angles or quaternions. For details on parameterizations of rotation matrices, the reader is referred to [18], [19] and [20].

Distortion Models

The assumption thus far has been that a linear model accurately models the imaging process. For cameras with real lenses, this assumption does not generally hold. The most important deviation is typically radial distortion, which is prevalent in lenses with short focal length and wide field of view [16]. Radial distortion occurs when light rays bend more near the edges of a lens than they do at its optical center. The remedy is to remove radial distortion as a preprocessing step, which in effect makes the camera a linear sensor again. Figure 2.2 illustrates this step. Let the image plane coordinates of a point in the camera frame be denoted by $\begin{bmatrix} \tilde{x} & \tilde{y} \end{bmatrix}^T$. For the ideal, non-distorted pinhole camera model, a world point $\mathbf{x} = \begin{bmatrix} x & y & z \end{bmatrix}^T$ in the camera coordinate system projected to the image plane in normalized coordinates is expressed as

$$\begin{bmatrix} \tilde{x} \\ \tilde{y} \\ 1 \end{bmatrix} = \frac{1}{z}\begin{bmatrix} x \\ y \\ z \end{bmatrix}. \tag{2.13}$$

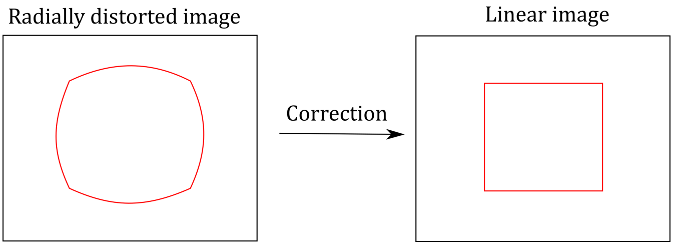
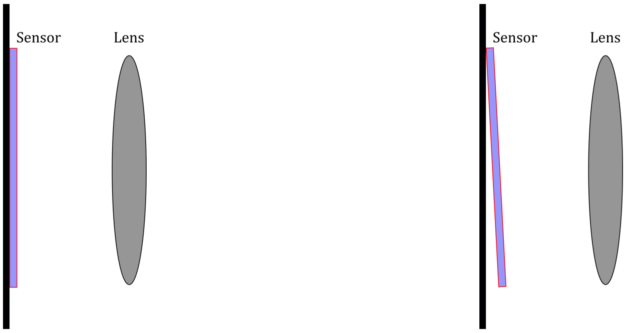


Figure 2.2: A radially distorted image, and the corrected image which would have been obtained with a linear lens.

The actual image coordinates are related to the ideal image coordinates by a radial displacement, which can be modelled as [16]

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x}(1 + k_1 r^2 + k_2 r^4 + \dots) \\ \tilde{y}(1 + k_1 r^2 + k_2 r^4 + \dots) \end{bmatrix} \tag{2.14}$$

where $\begin{bmatrix} x_d & y_d \end{bmatrix}^T$ are the distorted image coordinates, and $(1 + k_1 r^2 + k_2 r^4 + \dots)$ is a distortion factor, which is a function of the radius $r^2 = \tilde{x}^2 + \tilde{y}^2$. The parameters $k_1, k_2, \dots$ are the radial distortion parameters, which are normally calculated as a part of the camera calibration. Typically, one or two coefficients are enough to compensate for the radial distortion [21]. A lens could also introduce tangential distortion, which typically occurs if the lens and the sensor plane are not parallel. Figure 2.3b shows a case where the lens and sensor plane are not parallel, causing tangential distortion. The tangential distortion can be modeled by [21]



(a) No tangential distortion.

(b) Non-parallel lens and sensor leading to tangential distortion.

Figure 2.3: Illustration of tangential distortion.

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x} + 2p_1\tilde{x}\tilde{y} + p_2(r^2 + 2\tilde{x}^2) \\ \tilde{y} + p_1(r^2 + 2\tilde{y}^2) + 2p_2\tilde{x}\tilde{y} \end{bmatrix} \tag{2.15}$$

where p_1 and p_2 are the tangential distortion parameters. Combining the two distortion models, the distorted image coordinates in the normalized image plane can be expressed as

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x} + \tilde{x}(k_1 r^2 + k_2 r^4 + \dots) + 2p_1\tilde{x}\tilde{y} + p_2(r^2 + 2\tilde{x}^2) \\ \tilde{y} + \tilde{y}(k_1 r^2 + k_2 r^4 + \dots)) + p_1(r^2 + 2\tilde{y}^2) + 2p_2\tilde{x}\tilde{y} \end{bmatrix} \tag{2.16}$$

while the distorted pixel coordinates and the normalized distorted coordinates are related through the camera matrix by

$$\begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} = \mathbf{K} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} . \tag{2.17}$$

This model specifies the projection of 3D points to the normalized image plane, but it does not give a direct solution to the back-projection problem, where we want to recover the line of sight from image coordinates. If both radial and tangential distortion is considered, there is no analytic solution to the inverse mapping, so a nonlinear search is required to recover the undistorted image coordinates from the distorted ones [21]. Computer vision software can provide such methods, for example MATLAB [22] and OpenCV [23] readily provides functions for undistorting an image given the distortion parameters.

Complete Camera Sensor Model

Given that the measurements from the camera is a 2D coordinate in pixel values, and assuming that the radial and tangential distortions described in the previous section has been corrected, the camera model becomes

$$\begin{bmatrix} u \\ v \end{bmatrix} = \mathbf{HK} \begin{bmatrix} \mathcal{R} & \mathbf{t} \end{bmatrix} \frac{1}{z_w} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} \tag{2.18}$$

where

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.19}$$

is a projection matrix used to remove the bottom row from the homogeneous representation, and the factor $\frac{1}{z_w}$ scales the point $\mathbf{x}$ to normalized image coordinates (where the image plane is located at unit distance along the principal axis). The pixel coordinates are given by $[u, v]^T$ As is evident from the model there is a loss of information with a single camera, as we are going from three parameters in $\mathbf{x}$ to two parameters in $\begin{bmatrix} u & v \end{bmatrix}^T$. If the distortions described in the previous sections has been corrected in the image coordinates, this mapping can be inverted to give the normalized image plane projection of the image coordinates.

2.2 Computer Vision

Computer vision is a field of engineering and science concerned with extracting useful information from images. This has proved to be a challenging task, and it is still today an active field of research. Visual data is very complex, and the same object represented by two different images could be perceived very differently by a computer based on variations such as changes in illumination, partial occlusion of objects, changes in orientation, deformation and so on. Such variations are illustrated in figure 2.4. Despite these challenges, the recent advancements made within the field of deep learning, and deep convolutional neural nets in particular, has significantly improved the ability of computers to recognize objects in images [24]. The ImageNet Large Scale Visual Recognition Challenge has been run annually since 2010, and is a benchmark in object category classification and detection on hundreds of object categories and millions of images [25]. The results of the top-5 classification errors from the challenges up until 2016 is shown in figure 2.5. The top-5 error rate is the fraction of test images for which the correct label is not among the five labels considered most probable by the classifier.

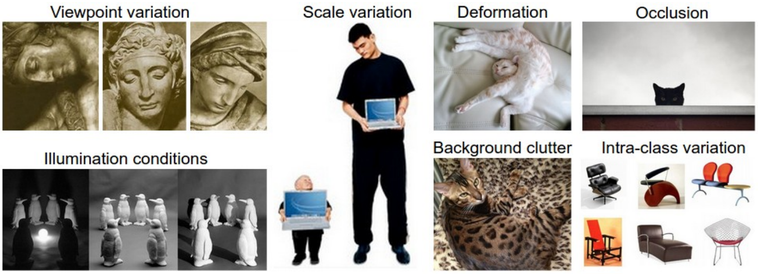


Figure 2.4: Challenges related to object classification in images.[1]

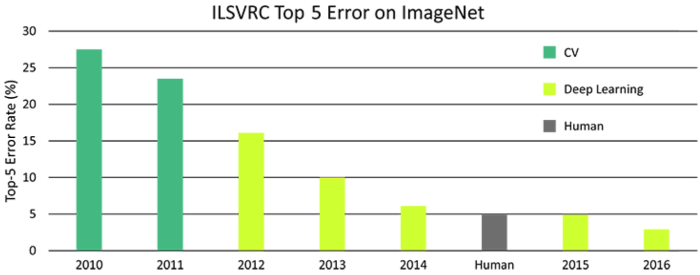


Figure 2.5: ImageNet top 5 errors over time.[2]

As seen in figure 2.5, deep learning algorithms dominate the field of object category classification and detection, even outperforming humans with state-of-the-art deep convolutional neural nets.

[1] Image courtesy: http://cs231n.github.io/classification/

[2] Image courtesy: https://www.dsiac.org/resources/journals/dsiac/winter-2017-volume-4-number-1/real-time-situ-intelligent-video-analytics

2.2.1 Convolutional Neural Nets for Computer Vision

A convolutional neural net, abbreviated CNN, is a neural network suited for object detection in computer vision applications. Since their introduction by LeCun et al. [26] in the late 1980's, convolutional neural nets have shown excellent performance at tasks such as hand-written digit classification and face detection. In 2012, Krizhevsky et al. [27] won the ImageNet 2012 classification benchmark with their deep convolutional neural network, dubbed AlexNet, achieving an top-5 test error rate of 15.3%, compared with the 2nd place result of 26.2%. Since then, the ImageNet classification benchmark has been dominated by deep convolutional nets. The dramatic improvement in performance can be attributed to several factors [28]:

1. The availability of very large datasets, with millions of labeled examples.
2. The adaptation of powerful GPU processing implementations, making training of very large models tractable.
3. General improvement of algorithms. Examples are better model regularization strategies, such as dropout, to prevent overfitting [29], batch normalization [30] and residual nets [31] to improve the training of deep networks.

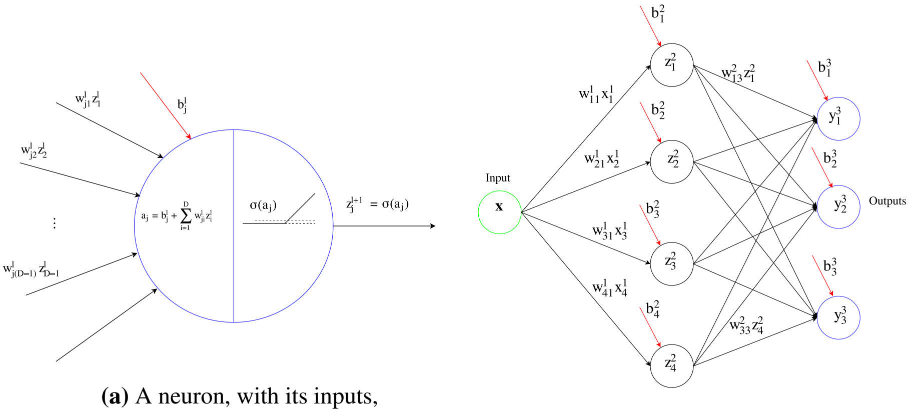
In a classifier using classical computer vision techniques, features are extracted from an image, for example using a histogram of oriented gradients, and the extracted features are subsequently used to train a classifier such as a support-vector machine. In a convolutional neural network on the other hand, the features are hidden, and feature extraction and classification is performed in a single pipeline. Features in a CNN are generated via one or several layers of filter convolutions, which generates a set of abstract sub-images from the input image. For the user, a CNN classifier appears as a ”black box” where the internal workings of the feature extraction and classification are largely hidden. In order to understand how a convolutional neural net works, some basic concepts need to be introduced. The following sections first introduce the neuron and the concept of a neural network, followed by a introduction to convolutional neural nets commonly employed in computer vision.

Neural Networks

In linear regression, a model can be considered as a linear combination of fixed nonlinear functions of the input variables, on the form

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{n-1} w_j \phi_j(\mathbf{x}) \tag{2.20}$$

where $\phi_j(\mathbf{x})$ are known as the nonlinear *basis functions*. A neural network utilized for regression and classification on the other hand, can be thought of as a combination of basis functions in parametric form, where the parameters of the basis functions are adapted during training. The term *neural network* has its origins in attempts to find mathematical representations of information processing in biological systems. The basic building block of a neural network is the *neuron*, or *perceptron*, illustrated in figure 2.6a.



(a) A neuron, with its inputs, bias and output.

(b) A simple neural network.

Figure 2.6: A neuron, alongside a simple neural network with a single hidden layer.

The basic neural network can be described as a series of functional transformations. First M linear combinations of the input variables $x_1, \ldots, x_D$ are formed, as

$$a_j = b^1_j + \sum_{i=1}^{D} w^1_{ji} x_i \tag{2.21}$$

where $j = 1, \ldots, M$, b^1_j is a bias, the parameters w^1_{ji} are the weights, and the superscript 1 indicates that the corresponding parameters are the parameters of the first layer in the network. The result of this weighted sum of the inputs, a_j is known as the activation. Each a_j is then transformed via a nonlinear, differentiable activation function, $z_j = \sigma(a_j)$, which is then input to the next layer in the network. Figure 2.6a illustrates this process for a single neuron. The quantities z_j are referred to as *hidden units*, and the corresponding layers are referred to as *hidden layers*. A popular choice for the nonlinear activation functions $\sigma(\cdot)$ is the *rectified linear unit* (ReLU),

$$\sigma(a) = \max\{0, a\}. \tag{2.22}$$

The outputs z_j are again linearly combined to give output unit activations as

$$a_k = b^2_k + \sum_{j=1}^{M} w^2_{kj} z_j \tag{2.23}$$

where $k = 1, \ldots, M$ and K is the total number of outputs. This corresponds to the second layer of the network. Lastly, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k. A simple network with a single hidden layer is illustrated in figure 2.6b. The choice for the output activation function is largely determined by the nature of the data and the assumed distribution of the target variables. For binary classification problems, the logistic sigmoid function is commonly used [8].

Convolutional Neural Nets

A convolutional neural net or ConvNet, abbreviated CNN, is similar to ordinary neural networks in that they are made up of neurons that have learnable biases and weights. What distincts it from ordinary neural nets is that it assumes that the input data has a grid-like topology, such as an image being a 3D grid of pixels, where the depth is represented by the color channels in the image. As the name implies, the CNN employs convolution in one or more of the layers in the network, and the layers have neurons arranged in 3 dimensions. A traditional neural network use matrix multiplication by a matrix of parameters, with a separate parameter describing the interactions between each input unit and each output unit, meaning that every input unit interacts with every output unit. CNNs however, have *sparse* interactions, accomplished by convolving the input with kernels (square matrices with weights as entries) which are smaller than the input. This drastically reduces the parameters in the network compared to the traditional network using matrix multiplication. If there are m inputs and n outputs, the matrix multiplication approach requires $m \times n$ parameters. For an input such as an image, which can have thousands or millions of pixels, the number of parameters needed become very large. In a CNN, the number of connections for each output is limited by the kernel size and number of kernels [32].

CNNs employ parameter sharing, where the same parameter is used for more than one output in the model. This is accomplised by the convolution operation, where each member of the kernel is used at every position of the input. This means that rather than learning a separate set of parameters for every location, only one set is learned, further reducing the number of parameters needed in the model. A simple representation of a layer in a convolutional neural net is shown in figure 2.7. The neurons (indicated by white circles) arranged in depth in the output activation layer are connected to the same inputs through different kernels, while all neurons at the same depth share kernel parameters. As indicated in the figure, a single layer may consist of several filter kernels.

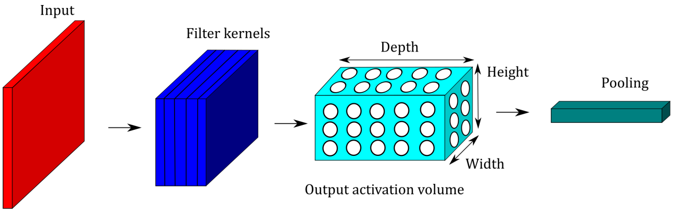


Figure 2.7: A CNN layer. The 3D input volume is transformed into a 3D volume of neuron activations.

A typical layer of a CNN consists of three stages; convolution, detection and pooling, illustrated in figure 2.8. Some terminology considers each stage in 2.8 as a separate layer in its own right, so the definition of what constitutes a convolutional layer may vary [32].

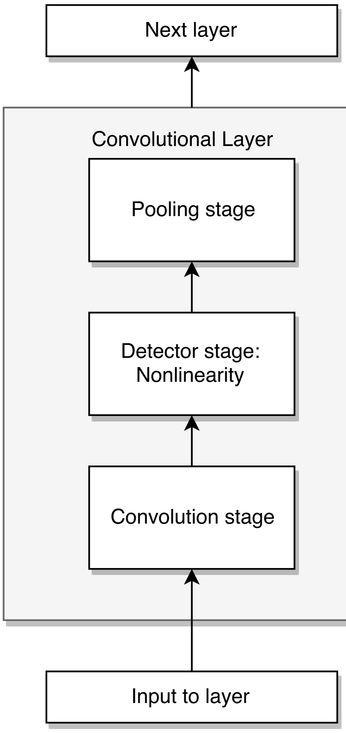


Figure 2.8: The three stages of a convolution layer.

The convolution stage performs several convolutions in parallel to produce a set of linear activations. At the next stage, the linear activations are run through a nonlinear activation function, in a similar fashion as for the traditional neural network. In the pooling stage, a *pooling function* is used to modify the output layer, replacing the output of the net by a summary statistic of the nearby outputs. As an example, the max pooling operation reports the maximum output within a rectangular region. Other examples are the average of a rectangular region, or a weighted average based on the distance from the central pixel.

A convolutional layer can be described by four *hyperparameters*. The term *hyperparameter* is used to distinguish them from the model parameters (weights and biases), and they are not subject to optimization during training. The four hyperparameters are

1. The number of kernels, K.

2. The spatial extent of the filter kernel, F.

3. The stride S, the number of pixel displacements for each calculation.
4. The amount of zero padding to the brim of the image, P.

For a given input image of size $\left[W_1 \times H_1 \times D_1\right]$ the spatial size of the output is

$$\begin{aligned} W_2 &= 1 + (W_1 - F + 2P)/S \\ H_2 &= 1 + (H_1 - F + 2P)/S \end{aligned} \tag{2.24}$$

Convolving the input with K kernels thus leads to the size of the output volume as $W_2 \times H_2 \times K$. Each filter kernel extends through the depth of the input, with dimensions $F \times F \times D_1$, making the total number of parameters for a single layer $K \cdot F \cdot F \cdot D_1$.

2.3 Spatial Data Acquisition using 3D Lidar

A lidar is an active electro-optical sensor that sends out a laser pulse, and subsequently measures the parameters of the return signal bounced off some object. The lidar is known under several names, some examples being ladar, lidar, LIDAR, LADAR or laser radar. The term lidar is the most common, and will be used throughout this thesis. In a lidar, a waveform generator generates a laser waveform. Depending on the type of lidar, the setup can include a single laser or a master oscillator with multiple lasers or laser amplifiers. There are many types of lidars, however this thesis uses a particular type of 3D lidar with a rotating detector array, which measures azimuth angle, elevation angle and range. The theory presented in this section is therefore not general to all lidars, but rather focuses on the particular type of lidar sensor used. The measurement is performed by the laser pulse being guided through transmit optics, traversing some medium, typically atmosphere, to a target. The laser pulse bounces off the target, and traverses the media again until receive optics captures the reflected pulse, guiding it to a detector or a detector array [33]. The laser and detector arrays rotate, taking multiple measurements as it scans the full 360 degree field of view. The range to a target can be determined based on the travel time of the laser pulse by

$$r = \frac{c}{2}(t_{rx} - t_{tx}), \tag{2.25}$$

where c is the speed of light in the intervening medium, and t_{tx} and t_{rx} is the transmission and reception time of the laser pulse, respectively.

2.3.1 The Velodyne VLP-16

The lidar used in this thesis is the Velodyne VLP-16 Puck, which is a small, real-time rotating lidar which continuously steams data over a TCP/IP connection when it is powered up. The Velodyne VLP-16 is shown in figure 2.9. The lidar has multiple return modes, where it can report either the strongest return signal, the last return signal, or both. The default return mode is to report the strongest return, and this mode is what is used in this thesis. The Velodyne VLP-16 features 16 laser/detector pairs mounted in a rotating housing, rapidly spinning to produce a 360° 3D point cloud. The specifications of the VLP-16 lidar are [34]

- Horizontal field of view of 360°.
- Horizontal angular resolution of 0.1°-0.4°, depending on rotational speed.
- Weight of 830 grams.
- Adjustable rotational speed between 5-20 Hz.
- Three laser return modes; strongest, last and dual.
- Vertical field of view of 30° (±15 °)
- Vertical angular resolution of 2°.
- Range of up to 100 meters (depending on application).
- Typical accuracy of ±3 centimeters.

Figure 2.9: The Velodyne VLP-16 Puck.

The range returned by the Velodyne VLP-16 lidar is measured along a beam at known azimuth angles α, and the 16 laser/detector pairs are mounted vertically at 2° intervals from -15° to 15° relative to the horizon in the lidar frame of reference. When the lidar is rotating at a frequency of 10 Hz the lasers fire at every $0.2°$, with a total of 1800 laser firings for a full rotation. The theoretical maximum number of range measurements for a full 360° scan is therefore $360/0.2 \times 16 = 28800$ measurements, where it is assumed that every single laser beam is reflected, and subsequently detected at the lidar. The lidar reports distances relative to itself in a spherical coordinate system (radius r, elevation ω, azimuth α). The relationship between the spherical and the cartesian coordinate frames is illustrated in figures 2.10 and 2.11. The lidar returns given in the sperical frame can be converted to a cartesian coordinate frame using basic trigonometry by

$$\begin{aligned} x &= P' \sin\alpha = r\cos\omega\sin\alpha \\ y &= P' \cos\alpha = r\cos\omega\cos\alpha \\ z &= r\sin\beta. \end{aligned} \tag{2.26}$$

A single reflection from the lidar can be represented in a cartesian coordinate frame centered in the lidar, shown in figure 2.11, using the transformation given in (2.26) as:

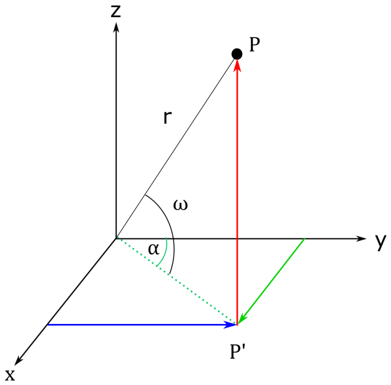
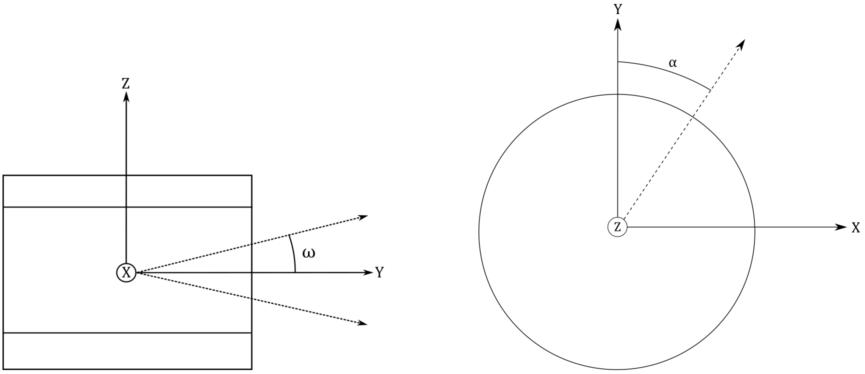


Figure 2.10: Lidar spherical coordinate system, with equivalent cartesian coordinates.



(a) Side view.

(b) Top view.

Figure 2.11: The lidar cartesian coordinate frame, with added elevation and azimuth angles.

$$\mathbf{P}^l_{ij} = \begin{bmatrix} r_{ij}\cos\omega_i\sin\alpha_j \\ r_{ij}\cos\omega_i\cos\alpha_j \\ r_{ij}\sin\omega_i \end{bmatrix} \tag{2.27}$$

Here, α_j is the j^{th} azimuth angle in the scan, and ω_i is the angle of the i^{th} vertical laser measured from the horizon. The superscript l denotes the cartesian frame centered in the lidar. r_{ij} is the range measured at the given azimuth and elevation angle. The Velodyne VLP-16 comes pre-calibrated from the factory, and Glennie et al. [35] did a calibration and stability analysis of the VLP-16 lidar, where they found that the factory supplied calibration is within the stated ±3 centimeter ranging accuracy.

Chapter 3

State Estimation and Object Tracking

An autonomous vessel operating in a dynamic and unpredictable environment such as an urban harbour environment will have to rely on its sensors in order to generate a coherent world view. A sensor will have limits to what it can percieve. The range and resolution of a sensor is limited, and moreover the sensors are subject to noise which perturbs the measurements in unpredictable ways. These limitations gives rise to *uncertainty* about the environment surrounding the vessel, and the process of inferring the value of a quantity of interest from indirect, inaccurate and uncertain observations is called estimation. This chapter first introduces some basic probability-theoretic concepts important in later derivations, and carries on in introducing probabilistic state-estimation techniques, as well as target tracking in systems where the origin of measurements is uncertain. For the state estimation, the Kalman filter and the Extended Kalman filter is introduced, before moving on to the target tracking algorithms in section 3.3.

3.1 Review of Basic Probability

This section serves as a review of basic concepts from probability and statistics which are important for later chapters. The theory presented in this section is largely based on chapter 2 in [36].

3.1.1 Probability Density Functions

A *probability density function* (pdf) $p(x)$ describes how a random variable, x, is distributed. Let $p(x)$ be the PDF for the random variable x over the interval $[a, b]$. The pdf is a non-negative function that satisfies the *axiom of total probability*, which can be stated as

$$\int_a^b p(x)dx = 1. \tag{3.1}$$

Probability is given by the area under the density function. The probability that x lies between c and d is given by

$$\Pr(c \leq x \leq d) = \int_c^d p(x)dx. \tag{3.2}$$

A conditioning variable can be introduced by letting $p(x|y)$ be a PDF over $x \in [a, b]$ conditioned on $y \in [c, d]$ such that

$$\forall y \qquad \int_a^b p(x)dx = 1. \tag{3.3}$$

This can be extended to the N-dimensional case as $p(\mathbf{x})$, where $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ with $x_i \in [a_i, b_i]$. The *joint density* of $\mathbf{x}$ and $\mathbf{y}$ can be written as $p(\mathbf{x}, \mathbf{y})$. For the N-dimensional case, the axiom of total probability requires that

$$\int_{\mathbf{a}}^{\mathbf{b}} p(\mathbf{x})d\mathbf{x} = \int_{a_N}^{b_N} \cdots \int_{a_2}^{b_2} \int_{a_1}^{b_1} p(x_1, x_2, \ldots x_N)dx_1 dx_2 \ldots dx_N = 1 \tag{3.4}$$

where $\mathbf{a} = (a_1, a_2, \ldots, a_N)$ and $\mathbf{b} = (b_1, b_2, \ldots, b_N)$

An important and much encountered probability density function is the *Normal* distribution, also known as a *Gaussian* distribution. In the multivariate case, this distribution is defined as

$$\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = |2\pi\boldsymbol{\Sigma}|^{-\frac{1}{2}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{3.5}$$

where $|\cdot|$ denotes the determinant, $\boldsymbol{\mu}$ is the mean vector, and $\boldsymbol{\Sigma}$ is the covariance matrix. For a normally distributed process, normality is preserved through linear transformations. The distribution is fully described by the two parameters, the mean value and the covariance.

3.1.2 Bayes' Rule

The joint probability density $p(\mathbf{x}, \mathbf{y})$ can be factored into a conditional and non-conditional factor as

$$p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x}|\mathbf{y})p(\mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x}). \tag{3.6}$$

In the case where $\mathbf{x}$ and $\mathbf{y}$ are statistically independent, the joint probability can be factored as $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{x})p(\mathbf{y})$. Rearranging (3.6) gives rise to *Bayes' rule*:

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{p(\mathbf{y})}. \tag{3.7}$$

Letting $\mathbf{x}$ represent the state of a system, and $\mathbf{y}$ be the data (e.g. a measurement), Bayes' rule can be used to infer the likelihood of the state given the data, also known as the *posterior*, $p(\mathbf{x}|\mathbf{y})$. This requires that we have a *prior* PDF over the state, $p(\mathbf{x})$, and a sensor model, $p(\mathbf{y}|\mathbf{x})$. This is done by expanding the denominator of (3.7) by marginalization such that

$$p(\mathbf{x}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{x})p(\mathbf{x})}{\int p(\mathbf{y}|\mathbf{x})p(\mathbf{x})d\mathbf{x}}. \tag{3.8}$$

where the limits of the denominator integral is over the entire allowable domain of the integration variable $\mathbf{x}$.

3.2 Bayesian State Estimation

Estimation is the process of inferring the value of a quantity of interest from imperfect, inaccurate and uncertain observations, and *tracking* is the estimation of the state of a moving object. State estimation uses statistical knowledge of the various uncertainties and prior information, the measurements given from sensors influenced by noise, and the system dynamics to estimate the quantities of interest. A general outline of the components within a tracking system is displayed in figure 3.1. This section first introduces the theory of Bayesian state estimation, including the Kalman filter and its extensions to non-linear systems (the Extended Kalman filter), before delving into the data association problem central to object tracking. The Probabilistic Data Association filter (PDAF) is introduced, and its extension to the multi-target case with the Joint Probabilistic Data Association (JPDA) filter is presented. The Joint Integrated Probabilistic Data Association (JIPDA) filter, which is an extension of the JPDA to include target existence and visibility probabilities is presented at the end. This is also the target tracking filter used in this thesis.

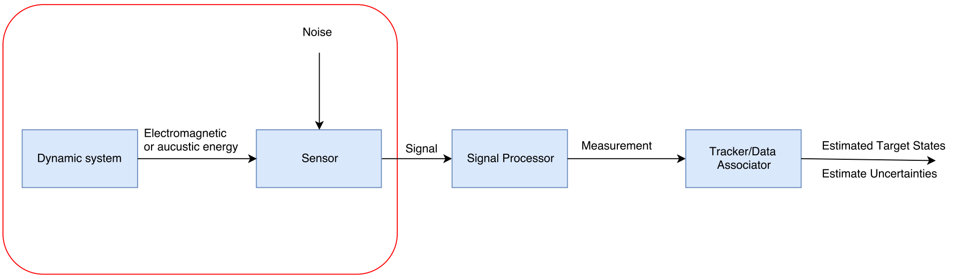


Figure 3.1: Overview of the components of a tracking system. The internal states of the components within the red box is generally not available to the system designer. The figure is adapted from [37].

3.2.1 Optimal Bayesian Filter

Bayesian inference in dynamic systems is concerned with the calculation of the probability density function $p(\mathbf{x}_{0:k}|\mathbf{Z}_{1:k})$ of a hidden state sequence $\mathbf{x}_{0:k} = \{\mathbf{x}_i\}_{i=0}^{k}$, given an observable sequence of measurements $\mathbf{Z}_{1:k} = \{\mathbf{Z}_i\}_{i=1}^{k}$. If a prior distribution of the states, $p(\mathbf{x}_{0:k})$, is known, and the relationship between the states and the measurements in the form of a sensor model or measurement equation is available, then Bayes' rule given in (3.8) can be used to calculate the posterior pdf as

$$p(\mathbf{x}_{0:k}|\mathbf{Z}_{1:k}) = \frac{p(\mathbf{Z}_{1:k}|\mathbf{x}_{0:k})p(\mathbf{x}_{0:k})}{p(\mathbf{Z}_{1:k})}. \tag{3.9}$$

In an object tracking application, measurements are received one by one over time, thus a recursive formulation of (3.9) which can be applied each time a new measurement arrives is needed. For this purpose, the elements of (3.9) can be factored as follows:

$$p(\mathbf{Z}_{1:k}|\mathbf{x}_{0:k}) = p(\mathbf{z}_k|\mathbf{Z}_{1:k-1}, \mathbf{x}_{0:k})p(\mathbf{Z}_{1:k-1}|\mathbf{x}_{0:k-1}) \tag{3.10a}$$

$$p(\mathbf{x}_{0:k}) = p(\mathbf{x}_k|\mathbf{x}_{0:k-1})p(\mathbf{x}_{0:k-1}) \tag{3.10b}$$

$$p(\mathbf{Z}_{1:k}) = p(\mathbf{z}_k|\mathbf{Z}_{1:k-1})p(\mathbf{Z}_{1:k-1}) \tag{3.10c}$$

where (3.10a) made use of the fact that measurements at time $k-1$ only depend on the states up to time $k-1$. Substituting into (3.9), rearranging and using a second application of Bayes' rule gives the general recursive form [38]

$$p(\mathbf{x}_{0:k}|\mathbf{Z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{Z}_{1:k-1}, \mathbf{x}_{0:k})p(\mathbf{x}_k|\mathbf{x}_{0:k-1})}{p(\mathbf{z}_k|\mathbf{Z}_{1:k-1})}p(\mathbf{x}_{0:k-1}|\mathbf{Z}_{1:k-1}). \tag{3.11}$$

The Markov Assumption

The general recursive form of the Bayes filter given in (3.11) requires that the complete state and measurement history up until time $k-1$ is available, and is used in the computations. In order to make the filter computationally feasible, the *Markov* assumption is commonly employed. The Markov assumption, or the complete state assumption, postulates that past and future data are independent if one knows the current state $\mathbf{x}_k$. Moreover, the measurements are also assumed to be conditionally independent of previous measurements and state histories, and depend only on the current state. With these assumptions, the distribution $p(\mathbf{z}_k|\mathbf{Z}_{1:k-1}, \mathbf{x}_{0:k})$ simplifies to $p(\mathbf{z}_k|\mathbf{x}_k)$, and $p(\mathbf{x}_k|\mathbf{x}_{0:k-1})$ becomes $p(\mathbf{x}_k|\mathbf{x}_{k-1})$. Such a temporal generative model is known as a *Hidden Markov model* (HMM) [39], illustrated in figure 3.2. Using the stated assumptions, (3.11) can be restated as

$$p(\mathbf{x}_{0:k}|\mathbf{Z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k)}{p(\mathbf{z}_k|\mathbf{Z}_{1:k-1})}p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{0:k-1}|\mathbf{Z}_{1:k-1}). \tag{3.12}$$

In Bayesian filtering, the *filtering distribution* $p(\mathbf{x}_k|\mathbf{Z}_{1:k})$ is of interest, in stead of the full joint posterior density $p(\mathbf{x}_{0:k}|\mathbf{Z}_{1:k})$. The filtering distribution can be derived from (3.12)

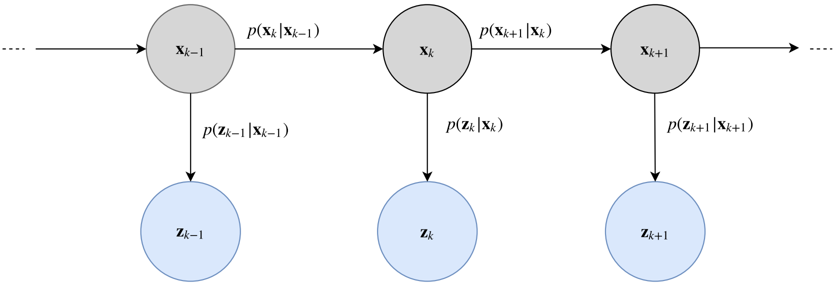

$p(\mathbf{x}_k|\mathbf{x}_{k-1})$
$p(\mathbf{x}_{k+1}|\mathbf{x}_k)$
$\mathbf{x}_{k-1}$
$\mathbf{x}_k$
$\mathbf{x}_{k+1}$
$p(\mathbf{z}_{k-1}|\mathbf{x}_{k-1})$
$p(\mathbf{z}_k|\mathbf{x}_k)$
$p(\mathbf{z}_{k+1}|\mathbf{x}_{k+1})$
$\mathbf{z}_{k-1}$
$\mathbf{z}_k$
$\mathbf{z}_{k+1}$

Figure 3.2: Hidden Markov model. Observable nodes are shaded in blue.

by marginalization over all previous states at all previous time steps [40],

$$\begin{aligned} p(\mathbf{x}_k|\mathbf{Z}_{1:k}) &= \int_{\mathbf{x}_{k-1}} \cdots \int_{\mathbf{x}_0} p(\mathbf{x}_{0:k}|\mathbf{Z}_{1:k})\mathrm{d}\mathbf{x}_0 \dots \mathrm{d}\mathbf{x}_{k-1} \\ &= \frac{p(\mathbf{z}_k|\mathbf{x}_k)}{p(\mathbf{z}_k|\mathbf{Z}_{1:k-1})} \int_{\mathbf{x}_{k-1}} \cdots \int_{\mathbf{x}_0} p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{0:k-1}|\mathbf{Z}_{1:k-1})\mathrm{d}\mathbf{x}_0 \dots \mathrm{d}\mathbf{x}_{k-1} \\ &= \frac{p(\mathbf{z}_k|\mathbf{x}_k)}{p(\mathbf{z}_k|\mathbf{Z}_{1:k-1})} \\ & \int_{\mathbf{x}_{k-1}} \left(p(\mathbf{x}_k|\mathbf{x}_{k-1}) \underbrace{\int_{\mathbf{x}_{k-2}} \cdots \int_{\mathbf{x}_0} p(\mathbf{x}_{0:k-1}|\mathbf{Z}_{1:k-1})\mathrm{d}\mathbf{x}_0 \dots \mathrm{d}\mathbf{x}_{k-2}}_{p(\mathbf{x}_{k-1}|\mathbf{Z}_{1:k-1})} \right) \mathrm{d}\mathbf{x}_{k-1} \end{aligned} \tag{3.13}$$

where the inner integrals over $\mathbf{x}_0$ through $\mathbf{x}_{k-2}$ simplify to $p(\mathbf{x}_{k-1}|\mathbf{Z}_{1:k-1})$ through marginalization. The remaining integral is known as the *Chapman-Kolmogorov equation* [40]

$$p(\mathbf{x}_k|\mathbf{Z}_{1:k-1}) = \int_{\mathbf{x}_{k-1}} p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{Z}_{1:k-1})\mathrm{d}\mathbf{x}_{k-1}. \tag{3.14}$$

The posterior pdf can finally be stated as [39], [40]

$$p(\mathbf{x}_k|\mathbf{Z}_{1:k}) = \underbrace{\frac{\overbrace{p(\mathbf{z}_k|\mathbf{x}_k)}^{\text{Filtering}}}{p(\mathbf{z}_k|\mathbf{Z}_{1:k-1})}}_{\text{Normalization}} \overbrace{\underbrace{\int_{\mathbf{x}_{k-1}} p(\mathbf{x}_k|\mathbf{x}_{k-1}) p(\mathbf{x}_{k-1}|\mathbf{Z}_{1:k-1})\mathrm{d}\mathbf{x}_{k-1}}_{p(\mathbf{x}_k|\mathbf{z}_{k-1})}}^{\text{Prediction}}. \tag{3.15}$$

This is the recursive formula that all Bayesian filtering and tracking algorithms tries to compute. The *prediction step* in (3.15) takes the conditional density at the previous stage $p(\mathbf{x}_{k-1}|\mathbf{Z}_{1:k-1})$ through the transition density $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ to form the predicted density $p(\mathbf{x}_k|\mathbf{Z}_{1:k-1})$. The filtering step uses the new data $\mathbf{z}_k$ through the likelihood function $p(\mathbf{z}_k|\mathbf{x}_k)$ to form the filtering distribution $p(\mathbf{x}_k|\mathbf{Z}_{1:k})$. The normalization factor $p(\mathbf{z}|\mathbf{Z}_{1:k-1})$ in (3.15) ensures that the filtering distribution is a valid pdf [40].

3.2.2 State Space Models

Almost all manouvering target tracking methods are model based, in that they assume that the target motions and observations can be mathematically modelled with sufficient accuracy [41]. State-space models are most commonly used, with additive noise, such as the discrete-time model

$$\mathbf{x}_{k+1} = \mathbf{f}(k, \mathbf{x}_k, \mathbf{u}_k) + \mathbf{w}_k, \qquad p(\mathbf{w}_k) = \mathcal{N}(\mathbf{w}_k; 0, \mathbf{Q}_k) \tag{3.16}$$

$$\mathbf{z}_k = \mathbf{h}(k, \mathbf{x}_k) + \mathbf{v}_k, \qquad p(\mathbf{v}_k) = \mathcal{N}(\mathbf{v}_k; 0, \mathbf{R}_k) \tag{3.17}$$

where $\mathbf{x}_k$, $\mathbf{z}_k$ and $\mathbf{u}_k$ are the state vector, measurement vector and control input vector, respectively, at the discrete time step k. The process noise $\mathbf{w}_k$ and the measurement noise $\mathbf{v}_k$ are white noise sequences. In a general tracking problem, the control input $\mathbf{u}_k$ is usually not known. $\mathbf{f}_k$ is a vector-valued function moving the current state and control input to the next state. The vector-valued function $\mathbf{h}_k$ maps the current state to the current measurement. These functions may also be time-varying. The covariance of the process noise is given by $\mathbf{Q}_k$, and the covariance of the measurement noise is given by $\mathbf{R}_k$.

The linear counterpart to (3.16) and (3.17), where the input $\mathbf{u}_k$ is omitted, are the equations

$$\mathbf{x}_{k+1} = \mathbf{F}_k \mathbf{x}_k + \mathbf{w}_k, \qquad p(\mathbf{w}_k) = \mathcal{N}(\mathbf{w}_k; 0, \mathbf{Q}_k) \tag{3.18}$$

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k, \qquad p(\mathbf{v}_k) = \mathcal{N}(\mathbf{v}_k; 0, \mathbf{R}_k) \tag{3.19}$$

where $\mathbf{F}_k$ and $\mathbf{H}_k$ are now (possibly time-varying) matrices.

The sensor model is given by (3.17) (in the linear case by (3.19)), and depends on the characteristics of the sensor and its noise characteristics.

3.2.3 Kalman Filter

The Kalman filter [42] is a linear state estimator, first introduced by Rudolf Kalman in 1960. It was quickly adopted by the engineering community, especially within the field of navigation [43]. The Kalman filter is derived based on some fundamental assumptions:

- The system's state evolves according to a known linear plant equation driven by a known input and an additive zero-mean white process noise with known covariance $\mathbf{Q}_k$, as in (3.18).
- The measurements are given by a known linear function of the state with an additive zero-mean white measurement noise with known covariance $\mathbf{R}_k$, as in (3.19).
- The initial state is unknown, assumed to be a random variable with a known mean and covariance.
- The initial error and noises are assumed mutually uncorrelated.

If these assumptions are satisfied, and if the initial state error and all the noises entering into the system are Gaussian distributed, the Kalman filter is the *minimum mean square error* (MMSE) estimator [37], which provides an exact, closed-form solution to (3.15). A derivation of the Kalman filter equations from (3.15) is provided in [44].

First, we will introduce some notation used in the Kalman filter algorithm, presented in table 3.1.

$\bar{\mathbf{x}}_k$	The prediction (a priori estimate) of the true state $\mathbf{x}_k$
$\hat{\mathbf{x}}_k$	The a posteriori estimate of the true state $\mathbf{x}_k$
$\bar{\mathbf{P}}_k$	The a priori estimated error covariance matrix, $E[(\mathbf{x}_k - \bar{\mathbf{x}}_k)(\mathbf{x}_k - \bar{\mathbf{x}}_k)^T]$
$\hat{\mathbf{P}}_k$	The a posteriori estimate of the error covariance matrix, $E[(\mathbf{x}_k - \hat{\mathbf{x}}_k)(\mathbf{x}_k - \hat{\mathbf{x}}_k)^T]$
$\mathbf{S}_k$	The innovation covariance matrix
$\mathbf{W}_k$	The Kalman gain matrix

Table 3.1: Notation used in the Kalman filter algorithm.

The Kalman filter is an iterative updating scheme with two main steps, the *prediction* and the *update* (or filtering) steps. The Kalman filter algorithm can be summarized as follows:

Input

- The process and measurement noise covariance matrices, $\mathbf{Q}_k$ and $\mathbf{R}_k$ provided by the system designer.

- The initial conditions, $\bar{\mathbf{x}}_0 = \mathbf{x}_0$ and $\bar{\mathbf{P}}_0 = E[(\mathbf{x}_0 - \hat{\mathbf{x}}_0)(\mathbf{x}_0 - \hat{\mathbf{x}}_0)^T]$.

Prediction

The first step of the Kalman filter algorithm is predicting the state and covariance for the next time step given the state estimate of the previous time step, using the system motion model and noise covariance, given by

$$\bar{\mathbf{x}}_{k+1} = \mathbf{F}_k \hat{\mathbf{x}}_k \tag{3.20}$$

$$\bar{\mathbf{P}}_{k+1} = \mathbf{F}_k \hat{\mathbf{P}}_k \mathbf{F}_k + \mathbf{Q}_k. \tag{3.21}$$

Update

In the update step the state vector is updated with the new measurement. In this simple example, it is assumed that there is only one measurement, which is known to originate from the target, i.e. the data-to-object associations are unambiguous. The *residual vector* $\boldsymbol{\nu}_{k+1}$ is the difference between the measurement and the predicted measurement, defined as

$$\boldsymbol{\nu}_{k+1} = \mathbf{z}_{k+1} - \bar{\mathbf{z}}_{k+1} \tag{3.22}$$

where

$$\bar{\mathbf{z}}_{k+1} = \mathbf{H}_{k+1}\bar{\mathbf{x}}_{k+1} \tag{3.23}$$

is the predicted measurement. The Kalman filter does not fully rely on either the measurement or the predicted state, but in stead applies a covariate scaling to the residual, the *Kalman gain* $\mathbf{W}_{k+1}$, which can be viewed as a comparison between confidence in the measurements versus confidence in the predicted state. The confidence of the measurement is described by the innovation covariance matrix resulting from the covariance of the prediction $\bar{\mathbf{P}}_{k+1}$ and the sensor model,

$$\mathbf{S}_{k+1} = \mathbf{H}_{k+1}\bar{\mathbf{P}}_{k+1}\mathbf{H}_{k+1}^T + \mathbf{R}_{k+1}. \tag{3.24}$$

The Kalman gain is given by

$$\mathbf{W}_{k+1} = \bar{\mathbf{P}}_{k+1}\mathbf{H}_{k+1}^T\mathbf{S}_{k+1}^{-1}. \tag{3.25}$$

The Kalman gain is applied to the residual, giving the posterior state estimate as

$$\hat{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_{k+1} + \mathbf{W}_{k+1}\boldsymbol{\nu}_{k+1}. \tag{3.26}$$

The state covariance matrix is also updated, taking into account the blending of the predicted state and the measurements as

$$\hat{\mathbf{P}}_{k+1} = (\mathbf{I} - \mathbf{H}_{k+1}\mathbf{W}_{k+1})\bar{\mathbf{P}}_{k+1}. \tag{3.27}$$

The discrete-time variable is incremented, and when a new measurement is ready the algorithm moves to the prediction step. The Kalman filter calculations for one cycle is summarized in figure 3.3.

3.2.4 Extended Kalman Filter

In the case where the system plant equation and/or measurement equation are given by nonlinear equations, the standard Kalman filter cannot be used. As the measurement model for the camera described in chapter 4 is a nonlinear one, the methods of the extended Kalman filter is used in the calculation of the innovation covariance matrix $\mathbf{S}$. The *Extended Kalman Filter* (EKF) linearizes the nonlinear dynamics and/or measurement equations about a trajectory that is continually updated with the state estimates resulting

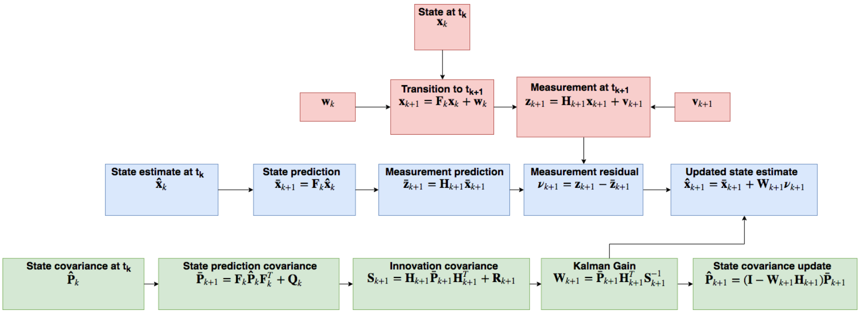

State at t_k
$\mathbf{x}_k$
$\mathbf{w}_k$
Transition to t_{k+1}
$\mathbf{x}_{k+1} = \mathbf{F}_k\mathbf{x}_k + \mathbf{w}_k$
Measurement at t_{k+1}
$\mathbf{z}_{k+1} = \mathbf{H}_{k+1}\mathbf{x}_{k+1} + \mathbf{v}_{k+1}$
$\mathbf{v}_{k+1}$
State estimate at t_k
$\hat{\mathbf{x}}_k$
State prediction
$\bar{\mathbf{x}}_{k+1} = \mathbf{F}_k\hat{\mathbf{x}}_k$
Measurement prediction
$\bar{\mathbf{z}}_{k+1} = \mathbf{H}_{k+1}\bar{\mathbf{x}}_{k+1}$
Measurement residual
$\boldsymbol{\nu}_{k+1} = \mathbf{z}_{k+1} - \bar{\mathbf{z}}_{k+1}$
Updated state estimate
$\hat{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_{k+1} + \mathbf{W}_{k+1}\boldsymbol{\nu}_{k+1}$
State covariance at t_k
$\hat{\mathbf{P}}_k$
State prediction covariance
$\bar{\mathbf{P}}_{k+1} = \mathbf{F}_k\hat{\mathbf{P}}_k\mathbf{F}_k^T + \mathbf{Q}_k$
Innovation covariance
$\mathbf{S}_{k+1} = \mathbf{H}_{k+1}\bar{\mathbf{P}}_{k+1}\mathbf{H}_{k+1}^T + \mathbf{R}_{k+1}$
Kalman Gain
$\mathbf{W}_{k+1} = \bar{\mathbf{P}}_{k+1}\mathbf{H}_{k+1}^T\mathbf{S}_{k+1}^{-1}$
State covariance update
$\hat{\mathbf{P}}_{k+1} = (\mathbf{I} - \mathbf{W}_{k+1}\mathbf{H}_{k+1})\bar{\mathbf{P}}_{k+1}$

Figure 3.3: One cycle of the Kalman filter. The true evolution of the state is shaded in red, the state estimation is shaded in blue, and the covariance computation is shaded in green. The figure is adapted from [37].

from the measurements, and is a nonoptimal state estimation algorithm for nonlinear systems [37]. The linearization is based on the first order Taylor expansion of the nonlinear functions. The noise is assumed to enter additively, as for the standard Kalman filter.

The main difference between the standard Kalman filter and the EKF is that the Jacobians of the plant- and measurement equations are used in calculating the state prediction covariance, the innovation covariance, and the filter gain. The Jacobians for the plant- and measurement equations are given by

$$\mathbf{F}_k = \left.\frac{\partial \mathbf{f}(k,\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_k} = \left.\begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}\right|_{\mathbf{x}=\hat{\mathbf{x}}_k} \tag{3.28}$$

$$\mathbf{H}_{k+1} = \left.\frac{\partial \mathbf{h}(k+1,\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\bar{\mathbf{x}}_{k+1}} = \left.\begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \frac{\partial h_1}{\partial x_2} & \cdots \\ \frac{\partial h_2}{\partial x_1} & \frac{\partial h_2}{\partial x_2} & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}\right|_{\mathbf{x}=\bar{\mathbf{x}}_{k+1}} . \tag{3.29}$$

The EKF loop is displayed in figure 3.4. Since the EKF uses a series expansion of the nonlinear equations in the Kalman filter equations, the extended Kalman filter can diverge if the reference about which the linearization takes place is poor. This situation commonly occurs at the initial starting point of the recursive estimation process, the EKF is very sensitive to the accuracy of the initial conditions. Inaccurate prior information about the true state causes a large error in $\bar{\mathbf{x}}_0$, which forces $\bar{\mathbf{P}}_0$ to be large. This can lead to two problems in getting the filter started [43]:

1. A large initial prior error covariance $\bar{\mathbf{P}}_0$ combined with a low-noise measurement at the first iteration of the EKF algorithm causes the error covariance to jump from a very large value to a small value in one step. This can lead to numerical problems, a non-positive-definite error covariance matrix $\bar{\mathbf{P}}_k$ at any point in the recursive process usually leads to divergence. The filter designer must take precautions in preserving

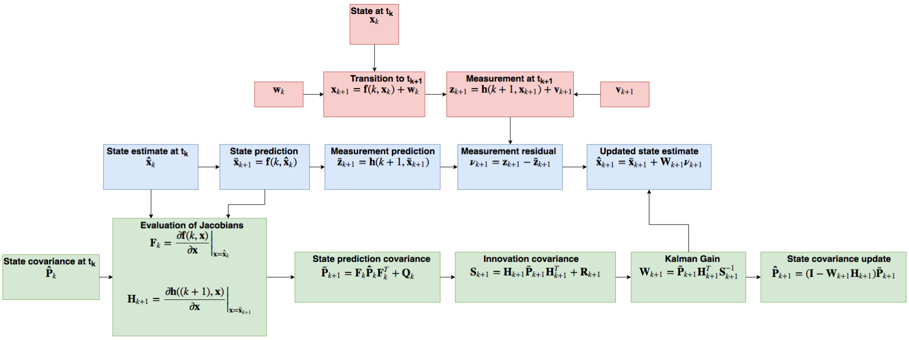

State at t_k
$\mathbf{x}_k$
$\mathbf{w}_k$
Transition to t_{k+1}
$\mathbf{x}_{k+1} = \mathbf{f}(k, \mathbf{x}_k) + \mathbf{w}_k$
Measurement at t_{k+1}
$\mathbf{z}_{k+1} = \mathbf{h}(k+1, \mathbf{x}_{k+1}) + \mathbf{v}_{k+1}$
$\mathbf{v}_{k+1}$
State estimate at t_k
$\hat{\mathbf{x}}_k$
State prediction
$\bar{\mathbf{x}}_{k+1} = \mathbf{f}(k, \hat{\mathbf{x}}_k)$
Measurement prediction
$\bar{\mathbf{z}}_{k+1} = \mathbf{h}(k+1, \bar{\mathbf{x}}_{k+1})$
Measurement residual
$\boldsymbol{\nu}_{k+1} = \mathbf{z}_{k+1} - \bar{\mathbf{z}}_{k+1}$
Updated state estimate
$\hat{\mathbf{x}}_{k+1} = \bar{\mathbf{x}}_{k+1} + \mathbf{W}_{k+1}\boldsymbol{\nu}_{k+1}$
Evaluation of Jacobians
$\mathbf{F}_k = \frac{\partial \mathbf{f}(k, \mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\hat{\mathbf{x}}_k}$
$\mathbf{H}_{k+1} = \frac{\partial \mathbf{h}((k+1), \mathbf{x})}{\partial \mathbf{x}}\Big|_{\mathbf{x}=\bar{\mathbf{x}}_{k+1}}$
State covariance at t_k
$\hat{\mathbf{P}}_k$
State prediction covariance
$\bar{\mathbf{P}}_{k+1} = \mathbf{F}_k\hat{\mathbf{P}}_k\mathbf{F}_k^T + \mathbf{Q}_k$
Innovation covariance
$\mathbf{S}_{k+1} = \mathbf{H}_{k+1}\bar{\mathbf{P}}_{k+1}\mathbf{H}_{k+1}^T + \mathbf{R}_{k+1}$
Kalman Gain
$\mathbf{W}_{k+1} = \bar{\mathbf{P}}_{k+1}\mathbf{H}_{k+1}^T\mathbf{S}_{k+1}^{-1}$
State covariance update
$\hat{\mathbf{P}}_{k+1} = (\mathbf{I} - \mathbf{W}_{k+1}\mathbf{H}_{k+1})\bar{\mathbf{P}}_{k+1}$

Figure 3.4: One cycle of the Extended Kalman filter. The true evolution of the state is shaded in red, the state estimation is shaded in blue, and the covariance computation (including the evaluation of the Jacobians) is shaded in green. The figure is adapted from [37].

the symmetry and positive definiteness of the error covariance matrix in the first step of the recursive filter loop.

2. The initial state $\bar{\mathbf{x}}_0$ is the best estimate of the state at the time of initialization, and is used as the reference for linearization in the first step of the algorithm. If this initial state is far from the true state (i.e. the error is large), the first-order approximation used in the linearization will be poor. This may cause the filter to diverge.

3.3 Object Tracking

This section presents object tracking methods which are based on the point-target assumption, meaning that any potential targets will give rise to a single point measurement, if any measurements at all. The object tracking methods considers the possibility that a measurement could originate from clutter or noise as well as from the actual target itself, and the basic idea is to calculate the *association probabilities* between a set of *validated measurements* and the predicted state of a target before using the measurements in the state update. This section starts with the Probabilistic Data Association Filter (PDAF), before progressively generalizing it to the multi-target case with the Joint Integrated Probabilistic Data Association filter (JPDA), and its extension to include calculation of target existence and visibility probabilities in the Joint Integrated Probabilistic Data Association filter (JIPDA).

3.3.1 The Probabilistic Data Association Filter

As described in section 3.2.3, the basic Kalman filter assumes that the measurements received are unambiguously originating from the actual target, i. e. the tracking is performed with an ideal sensor. In a realistic tracking situation, this is not necessarily true. A realistic sensor could give returns that originate from clutter or noise rather than the actual target of interest. In other words, the origin of the measurements are uncertain. This leads to the *data association* problem, which is the problem of determining which measurements are believed to originate from the target, and which measurements are clutter or noise, and can be discarded. The naive solution is to select the nearest gated measurement (or measurements) as the target-originated one, whereas the probabilistic data association filter (PDAF) uses the probabilistic distances to all gated measurements, scaling each one with its probabilistic distance. The derivations presented in this section is largely based on chapter 3 in [45]. The basic assumptions of the PDAF are listed below.

- There is only one target of interest, modeled by (3.18) and (3.19).
- The track has been initialized.
- The past information about the target is summarized approximately by

  $$p(\mathbf{x}_k|\mathbf{Z}_{1:k-1}) = \mathcal{N}[\mathbf{x}_k; \hat{\mathbf{x}}_k, \hat{\mathbf{P}}_k]$$

  where, as before, $\mathbf{Z}_{1:k-1} = \{\mathbf{z}_i\}_{i=1}^{k-1}$ is the sequence of measurements up to and including time $k-1$.
- At each time a validation region as in (3.31) is set up.
- At most one of the possibly many validated measurements can be target originated.
- The remaining measurements are assumed due to false alarm or clutter and are modeled as independent identically distributed with uniform spatial distribution.
- The target detections occur independently over time with known probability P_D.

The *latest set of validated measurements*, i.e. measurements at time k that falls within the validation region, is denoted

$$\mathbf{Z}_k = \{\mathbf{z}_{i,k}\}_{i=1}^{m_k} \tag{3.30}$$

where $\mathbf{z}_{i,k}$ is the i-th validated measurement, and m_k is the number of measurements within the validation region at time k.

The Validation Region

A *validated* measurement is a measurement falling within a range gate set up for detecting the signal from the target. This is done to avoid searching the entire measurement space for the signal from the target of interest. A measurement that falls within the gate, while not guaranteed to have originated from the target of interest, is a *valid association candidate*, thus the name *validation region*. Consider a target that has a valid track, meaning its filter is initialized. Then, according to the assumptions described in the previous section, one has the predicted mean value of the measurement $\bar{\mathbf{z}}_k = \mathbf{H}_k \bar{\mathbf{x}}_k$ and the associated covariance $\mathbf{S}_k$. With the assumption that the true measurement conditioned on the past is normally distributed, the true measurement will be within the region described by

$$\mathcal{V}(k, \gamma) = \{\mathbf{z} : [\mathbf{z} - \bar{\mathbf{z}}_k]^T \mathbf{S}_k^{-1} [\mathbf{z} - \bar{\mathbf{z}}_k] \leq \gamma\} \tag{3.31}$$

where γ is the gate threshold determined by the chosen *gate probability* P_G [45]. This threshold is obtained from tables of the χ^2 distribution, since the quadratic form in (3.31) that defines the validation region is χ^2 distributed [37]. The volume of the validation region is

$$V(k) = c_{n_z} |\gamma \mathbf{S}_k|^{\frac{1}{2}} = c_{n_z} \gamma^{\frac{n_z}{2}} |\mathbf{S}_k|^{\frac{1}{2}} \tag{3.32}$$

where c_{n_z} is the volume of the n_z-dimensional hypersphere, and n_z is the dimension of the measurement. An example of a gating process is shown in figure 3.5. In this example, measurements $\mathbf{z}_{1,k}$ and $\mathbf{z}_{2,k}$ are within the gating region, and are considered valid association candidates. The remaining measurements, indicated by red dots, are outside the gating region and are discarded.

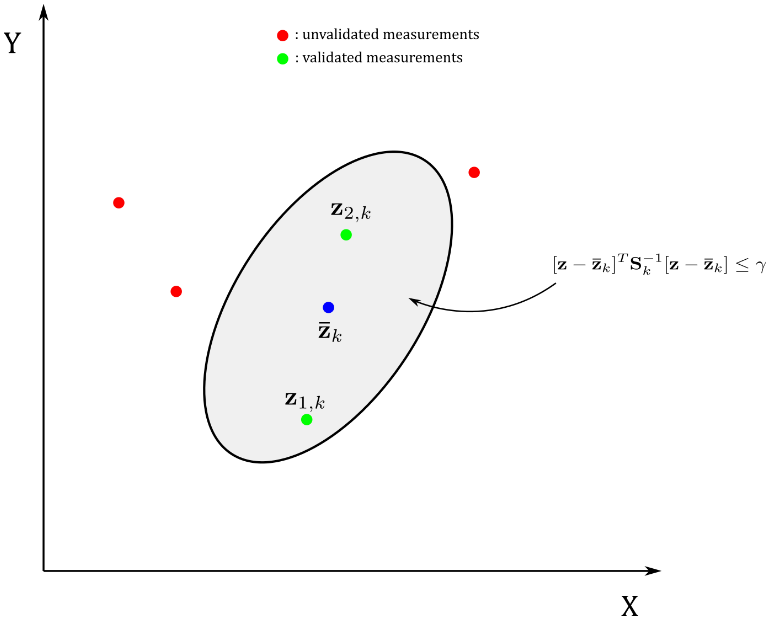

Y
● : unvalidated measurements
● : validated measurements
$\mathbf{z}_{2,k}$
$[\mathbf{z} - \bar{\mathbf{z}}_k]^T \mathbf{S}_k^{-1} [\mathbf{z} - \bar{\mathbf{z}}_k] \leq \gamma$
$\bar{\mathbf{z}}_k$
$\mathbf{z}_{1,k}$
X

Figure 3.5: A validation gate with two validated measurements.

State Estimation in the PDAF

The prediction of the state $\bar{\mathbf{x}}_k$, measurement $\bar{\mathbf{z}}_k$ and the covariance $\bar{\mathbf{P}}_k$ from $k-1$ to k is done in the same way as for the Kalman filter, given by (3.20) and (3.21). With the previously stated assumptions, the association events

$$\theta_{i,k} = \begin{cases} \{\mathbf{z}_{i,k} \text{ is the target originated measurement}\} & i = 1, \ldots, m_k \\ \{\text{none of the measurements is target originated}\} & i = 0 \end{cases} \tag{3.33}$$

are mutually exclusive and exhaustive for $m_k \geq 1$. Defining the association probability as

$$\beta_{i,k} \triangleq P(\theta_{i,k}|\mathbf{Z}_k) \tag{3.34}$$

the conditional mean of the state can be written as [45]

$$\hat{\mathbf{x}}_k = \sum_{i=0}^{m_k} \hat{\mathbf{x}}_{i,k}\beta_{i,k} \tag{3.35}$$

where $\hat{\mathbf{x}}_{i,k}$ is the estimated state conditioned on the event that the i-th measurement is correct given as

$$\hat{\mathbf{x}}_{i,k} = \bar{\mathbf{x}}_k + \mathbf{W}_k\boldsymbol{\nu}_{i,k} \quad i = 1, \ldots, m_k \tag{3.36}$$

and the corresponding innovation is

$$\boldsymbol{\nu}_{i,k} = \mathbf{z}_{i,k} - \bar{\mathbf{z}}_k. \tag{3.37}$$

The gain matrix $\mathbf{W}_k$ is the same as in the Kalman filter, given in (3.25). Combining (3.35) and (3.36), the state update can be written as

$$\hat{\mathbf{x}}_k = \bar{\mathbf{x}}_k + \mathbf{W}_k\boldsymbol{\nu}_k \tag{3.38}$$

where $\boldsymbol{\nu}_k$ is the *combined innovation*

$$\boldsymbol{\nu}_k = \sum_{i=1}^{m_k} \beta_{i,k}\boldsymbol{\nu}_{i,k} \tag{3.39}$$

In the case where there is no validated measurements, or none of the validated measurements are correct, the filter propagates the previous state estimate as the updated state.

The Covariance Update

The covariance for the updated state in (3.38) is [45]

$$\hat{\mathbf{P}}_k = \beta_{0,k}\bar{\mathbf{P}}_k + [1 - \beta_{0,k}]\hat{\mathbf{P}}_k^c + \tilde{\mathbf{P}}_k \tag{3.40}$$

where $\hat{\mathbf{P}}_k^c$ is the covariance of the state updated with the correct measurement, similarly as in the Kalman filter:

$$\hat{\mathbf{P}}_k^c = \bar{\mathbf{P}}_k - \mathbf{W}_k\mathbf{S}_k\mathbf{W}_k^T. \tag{3.41}$$

The term $\tilde{\mathbf{P}}_k$ represents the *spread of the innovations*, defined as

$$\tilde{\mathbf{P}}_k \triangleq \mathbf{W}_k\Big[\sum_{i=1}^{m_k} \beta_{i,k}\boldsymbol{\nu}_{i,k}\boldsymbol{\nu}_{i,k}^T - \boldsymbol{\nu}_k\boldsymbol{\nu}_k^T\Big]\mathbf{W}_k^T. \tag{3.42}$$

The two first terms of (3.40) is a weighing between the prediction covariance and the covariance updated with the correct measurement, weighted by the probability of no correct measurements ($\beta_{0,k}$) and the probability that the correct measurement is available ($1-\beta_{0,k}$), respectively. Since one does not know which of the m_k gated measurements is the correct one, the positive semidefinite term given by (3.42) increases the covariance to account for the measurement origin uncertainty [45].

The Association Probabilities

The form of the association probabilities depend on the *probability mass function* $\mu_F(\varphi)$ of the number of false measurements in the validation region. Two models can be used for $\mu_F(\varphi)$ in the volume of interest V, a parametric Poisson model with spatial density given by λ,

$$\mu_F(\varphi) = e^{-\lambda V}\frac{(\lambda V)^{\varphi}}{\varphi!} \tag{3.43}$$

and a nonparametric *diffuse prior model*,

$$\mu_F(\varphi) = \mu_F(\varphi - 1) = \delta. \tag{3.44}$$

The two models yield slightly different expressions for the assiciation probabilities $\beta_i(k)$, which will simply be stated below. For a full derivation of the expressions, the reader is reffered to [45], section 3.4.3. Using the Poisson clutter model, the association probabilities are

$$\beta_{i,k} = \begin{cases} \frac{e_i}{b+\sum_{j=1}^{m_k} e_j} & i = 1, \ldots, m(k) \\ \frac{b}{b+\sum_{j=1}^{m_k} e_j} & i = 0 \end{cases} \tag{3.45}$$

where

$$e_i \triangleq e^{-\frac{1}{2}\boldsymbol{\nu}_{i_k}^T \mathbf{S}_k^{-1} \boldsymbol{\nu}_{i,k}} \tag{3.46}$$

and

$$b \triangleq \frac{2\pi}{\gamma}^{\frac{n_z}{2}} \lambda V(k) c_{n_z}^{-1} \frac{1-P_D P_G}{P_D}. \tag{3.47}$$

The nonparametric diffuse prior model is the same as the above expressions except for replacing (3.47) with

$$b \triangleq \frac{2\pi}{\gamma}^{\frac{n_z}{2}} m_k c_{n_z}^{-1} \frac{1-P_D P_G}{P_D}. \tag{3.48}$$

3.3.2 JPDA

The Joint Probabilistic Data Association filter (JPDA) is a multi-target extension of the PDAF presented in section 3.3.1. In the PDAF, all incorrect measurements are modeled as random interference, with uniform spatial distribution, while the JPDA allows the measurements from a target to fall within the validation region of a neighboring target. This can happen over several sampling intervals, acting as a persistent interference. Such a situation is depicted in figure 3.6, where two targets have overlapping validation regions, and measurement $\mathbf{z}_{1,k}$ falls within the validation region of both targets. The performance of the PDAF degrades significantly when a neighboring target acts as a persistent interference [37]. In addition to allowing measurements from one target to fall within the validation region of a neighboring target, the JPDA builds on the assumptions that:

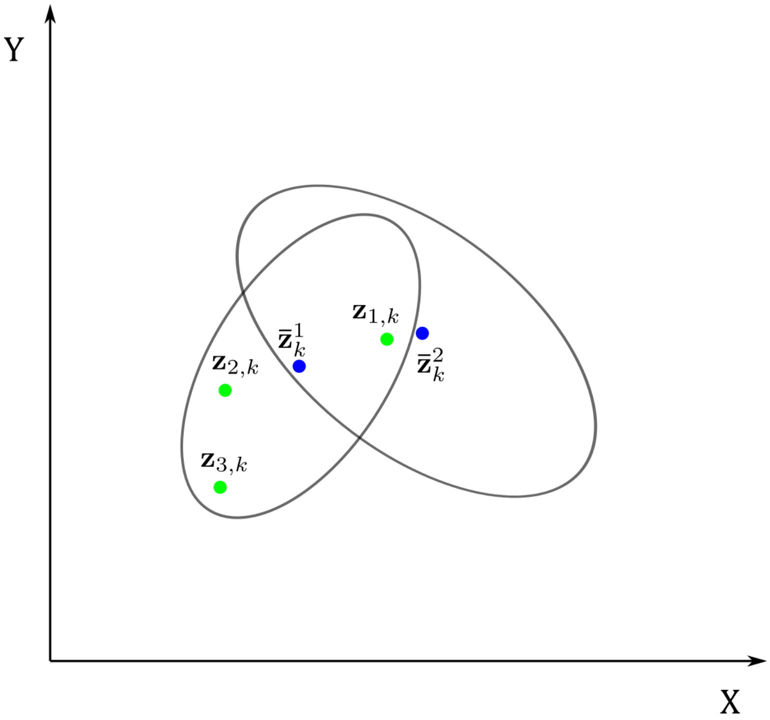


Figure 3.6: Two overlapping validation gates, sharing the measurement $\mathbf{z}_{1,k}$. The JPDA filter accounts for the fact that this measurement is more likely to originate from target 2 than target 1 in this situation, while the PDAF would make no such discrimination.

- There is a known number of established targets in clutter.
- The past is summarized by an approximate sufficient statistic - state estimates (approximate conditional mean) and covariances for each target.
- The states are assumed Gaussian distributed with the aforementioned mean and covariances.
- Each target has a motion and measurement model as in (3.18) and (3.19). Different targets may have different models.

- The number of false measurements is assumed Poisson distributed with intensity λ.

Moreover, it is assumed that a target may generate at most one measurement (*point target assumption*), and that any measurement comes from at most one target (*no merged measurement assumption*) meaning that measurements from two or several targets may not merge into one measurement. The approach of the JPDA is to compute the measurement-to-target association probabilities jointly across the targets. Its key feature is the evaluation of the conditional probabilities of the *association hypotheses*. Following along the lines of [2], an association hypothesis $\mathbf{a}_k$ can be defined as a vector $\mathbf{a}_k = [a_k(1), a_k(2), \dots, a_k(n)]^T$ such that

$$\mathbf{a}_k(t) = \begin{cases} j & \text{if measurement } j \text{ is claimed by target } t \\ 0 & \text{if no measurement is claimed by target } t \end{cases} \tag{3.49}$$

where $j = 1, \dots, m_k$ is the number of measurements, and $t = 1, \dots, n$ is the number of known targets. Due to the point target assumption and the no merged measurements assumption, the association hypotheses are mutually exclusive and exhaustive. The theorem of total probability therefore yields

$$p(\mathbf{x}_{1,k}, \mathbf{x}_{2,k}, \dots, \mathbf{x}_{n,k} | \mathbf{Z}_{1:k}) = \sum_{\mathbf{a}_k} p(\mathbf{x}_{1,k}, \mathbf{x}_{2,k}, \dots, \mathbf{x}_{n,k} | \mathbf{a}_k, \mathbf{Z}_{1:k}) P\{\mathbf{a}_k | \mathbf{Z}_{1:k}\} \tag{3.50}$$

where the sum goes over all feasible association hypotheses. The event-conditional densities $p(\mathbf{x}_k^1, \mathbf{x}_k^2, \dots, \mathbf{x}_k^n | \mathbf{a}_k, \mathbf{Z}_{1:k})$ can be easily obtained using a Kalman filter whenever a measurement $\mathbf{z}_{a_k(t),k}$ is assigned to a target t, and use the prediction otherwise. The equations for obtaining the association hypotheses probabilities $P\{\mathbf{a}_k | \mathbf{Z}_{1:k}\}$ will be given in section 3.3.3, and thus will not be stated here. Once the association hypotheses probabilities are found, the goal is to merge together different hypotheses, such that one is left with a single Gaussian for each track. The marginal association probabilities $\beta_{j,k}^t$ for each target t is found by summing over all joint events in which the marginal event of interest occurs,

$$\beta_{j,k}^t = P\{\mathbf{z}_{j,k} \text{ is assigned to target } t | \mathbf{Z}_{1:k}\} = \sum_{a_k \text{ s.t. } a_k(t)=j} P\{\mathbf{a}_k | \mathbf{Z}_{1:k}\}. \tag{3.51}$$

Once the marginal association probabilities are found, the state update for each target is done as for the PDAF, with (3.38) and (3.40).

3.3.3 JIPDA

The Joint Integrated Probabilistic Data Association filter (JIPDA) [46] is a multi-target tracking filter along the lines of the JPDA presented in section 3.3.2, where the main difference is the computation of track existence probabilities, as for the IPDA filter [47]. For a single target, the JIPDA filter reduces to the IPDA filter. The track existence probabilities allow for automatic track initiation, maintenance and termination. The derivations in the following sections is based on work from [37], [46] and [2]. The notation used is the same as in [2].

Assumptions

The JIPDA filter builds on the same assumptions as the JPDA, repeated here for convenience:

- There is a known number of established tracks in clutter.
- The past is summarized by an approximate sufficient statistic - state estimates (approximate conditional mean) and covariances for each target.
- The states are assumed Gaussian distributed with the aforementioned mean and covariances.
- Each target has a motion and measurement model as in (3.18) and (3.19). Different targets may have different models.
- The number of false measurements is assumed Poisson distributed with intensity λ.

Track Existence and Visibility Models

In the original paper, Musicki and Evans [46] model target existence using two distinct Markov chains, which they named Markov Chain One and Markov Chain Two. Markov Chain One recognizes two possibilities; the target either does not exist, or it exists and is visible with a probability of detection. Markov Chain Two recognizes three possibilities; in addition to the two possibilities of Markov Chain One, the target may also exist, but not be visible. In this thesis, a similar approach to Markov Chain Two is used. We assume that track existence and track visibility are separated and decoupled. Let the predicted existence probability at time k be denoted $\bar{\varepsilon}_k^t$, and the predicted visibility probability be denoted $\bar{\eta}_k^t$. The prediction equations for track existence and visibility are given by

$$\begin{bmatrix} \bar{\varepsilon}_k^t \\ 1 - \bar{\varepsilon}_k^t \end{bmatrix} = \begin{bmatrix} p_{11}^{\varepsilon} & p_{12}^{\varepsilon} \\ p_{21}^{\varepsilon} & p_{22}^{\varepsilon} \end{bmatrix} \begin{bmatrix} \varepsilon_{k-1}^t \\ 1 - \varepsilon_{k-1}^t \end{bmatrix} \tag{3.52}$$

$$\begin{bmatrix} \bar{\eta}_k^t \\ 1 - \bar{\eta}_k^t \end{bmatrix} = \begin{bmatrix} p_{11}^{\eta} & p_{12}^{\eta} \\ p_{21}^{\eta} & p_{22}^{\eta} \end{bmatrix} \begin{bmatrix} \eta_{k-1}^t \\ 1 - \eta_{k-1}^t \end{bmatrix}. \tag{3.53}$$

The Markov chain coefficients must satisfy

$$\begin{aligned} p_{11}^{\varepsilon} + p_{12}^{\varepsilon} &= p_{21}^{\varepsilon} + p_{22}^{\varepsilon} \\ &= p_{11}^{\eta} + p_{12}^{\eta} \\ &= p_{21}^{\eta} + p_{22}^{\eta} + = 1. \end{aligned} \tag{3.54}$$

By assuming there is a known number of established targets, no target births can occur, and the Markov model is given by the survival of existing targets.

Association Events and Posterior Conditional Densities

As described in section 3.3.2, the association hypotheses probabilities $P\{\mathbf{a}_k|\mathbf{Z}_{1:k}\}$ must be found in order to perform the state update for each target. Using Bayes rule, the association hypotheses probabilities can be written as

$$\begin{aligned} P\{\mathbf{a}_k|\mathbf{Z}_{1:k}\} &= P\{\mathbf{a}_k|\mathbf{Z}_k, m_k, \mathbf{Z}_{1:k-1}\} \\ &\propto p\big(\mathbf{Z}_k|m_k, \mathbf{a}_k, \mathbf{Z}_{1:k-1}\big) P\{\mathbf{a}_k|m_k, \mathbf{Z}_{1:k-1}\}. \end{aligned} \tag{3.55}$$

Before we find concrete expressions for the terms in (3.55), let's first express the posterior event-conditional expected states and covariances as

$$\hat{\mathbf{x}}_k^{t,a_k(t)} = \begin{cases} \bar{\mathbf{x}}_k^t & \text{if } a_k(t) = 0 \\ \bar{\mathbf{x}}_k^t + \mathbf{W}_k^t\big(\mathbf{z}_k^{a_k(t)} - \mathbf{H}\bar{\mathbf{x}}_k\big) & \text{if } a_k(t) > 0 \end{cases} \tag{3.56}$$

$$\mathbf{P}_k^{t,a_k(t)} = \begin{cases} \bar{\mathbf{P}}_k^t & \text{if } a_k(t) = 0 \\ \big(\mathbf{I} - \mathbf{W}_k^t\mathbf{H}\big)\bar{\mathbf{P}}_k^t & \text{if } a_k(t) > 0. \end{cases} \tag{3.57}$$

In words, this means that a KF update is used if measurements are associated to the target, and the prediction is used if not. The joint event-conditional posterior can then be expressed as

$$p(\mathbf{x}_k^1, \mathbf{x}_k^2, \ldots, \mathbf{x}_k^n|\mathbf{a}_k, \mathbf{Z}_{1:k}) = \prod_{t=1}^{n} \mathcal{N}\big(\mathbf{x}_k^t; \mathbf{x}_k^{t,a_k(t)}, \mathbf{P}_k^{t,a_k(t)}\big). \tag{3.58}$$

Association Events and Posterior Probabilities

Next, define the likelihood ratio

$$l^{t,a_k(t)} = \frac{1}{\lambda}\mathcal{N}\big(\mathbf{z}_k^{a_k(t)}; \mathbf{H}\hat{\mathbf{x}}_k^{t,a_k(t)}, \mathbf{H}\mathbf{P}_k^{t,a_k(t)}\mathbf{H}^T + \mathbf{R}\big). \tag{3.59}$$

Let the number of clutter measurements hypothesized by the current association hypothesis be φ. The first term on the right-hand side of (3.55) can then be expressed as [2]

$$p\big(\mathbf{Z}_k|m_k, \mathbf{a}_k, \mathbf{Z}_{1:k-1}\big) = \frac{1}{V^\varphi} \prod_{t:a_k(t)>0} \big[\lambda l^{t,a_k(t)}\big] \tag{3.60}$$

where V is the volume of the surveillance region. To find the second term of (3.55), first define the individual track-wise detection events $\boldsymbol{\tau}(t)$ as $\boldsymbol{\tau}(t) = 1$ if target t is detected, $\boldsymbol{\tau}(t) = 0$ otherwise. Further define the track-oriented configuration as a vector containing the individual track-wise detections, $\boldsymbol{\tau} = [\boldsymbol{\tau}(1), \ldots, \boldsymbol{\tau}(n)]^T$. The vector $\boldsymbol{\tau}$ contains information about which targets are detected, but contains no information about measurement-to-target associations. The unconditional probability of the event $\boldsymbol{\tau}$ is given entirely by the individual detection events,

$$P\{\boldsymbol{\tau}\} = \prod_{t:a_k(t)=0} \left(1 - \bar{\varepsilon}_k^t P_D^t \bar{\eta}_k^t\right) \prod_{t:a_k(t)>0} \bar{\varepsilon}_k^t P_D^t \bar{\eta}_k^t. \tag{3.61}$$

Conditioned on $\boldsymbol{\tau}$, we have the following probabilities:

$$P\{m_k|\boldsymbol{\tau}\} = e^{\lambda V} \frac{(\lambda V)^{\varphi}}{\varphi!} \tag{3.62}$$

$$P\{\mathbf{a}_k|\boldsymbol{\tau}, m_k\} = \frac{\varphi!}{m_k!}. \tag{3.63}$$

The first probability follows from the fact that the number of measurements not accounted for by $\boldsymbol{\tau}$, the number of clutter measurements φ is Poisson distributed with rate λV, and the number of measurements m_k are uniquely determined by φ and $\boldsymbol{\tau}$. The second probability follows because given $\boldsymbol{\tau}$, we have $m_k!/\varphi!$ equally likely permutations of the detected measurements, each constituting a separate association hypothesis. The second term of (3.55) can be rewritten as

$$\begin{aligned} P\{\mathbf{a}_k|m_k\} &= P\{\mathbf{a}_k, \boldsymbol{\tau}|m_k\} \\ &\propto P\{\mathbf{a}_k|\boldsymbol{\tau}, m_k\}P\{m_k|\boldsymbol{\tau}\}P\{\boldsymbol{\tau}\} \end{aligned} \tag{3.64}$$

The association hypotheses probabilities $P\{\mathbf{a}_k|\mathbf{Z}_{1:k}\}$ can now be found by combining (3.61)-(3.63) and (3.64) into one equation, where after cancelling the V-terms and moving the terms involving λ into the proportionality constant, we are left with

$$P\{\mathbf{a}_k|\mathbf{Z}_{1:k}\} \propto \prod_{t:a_k(t)=0} \left(1 - \bar{\varepsilon}_k^t P_D^t \bar{\eta}_k^t\right) \prod_{t:a_k(t)>0} \bar{\varepsilon}_k^t P_D^t \bar{\eta}_k^t l^{t,a_k(t)}. \tag{3.65}$$

The total power of λ is always m_k, since one λ is contributed by every target in the likelihood and every false alarm in the prior probability contribute one λ, explaining the rationale behind moving it into the proportionality constant.

Marginal Association Probabilities

The key principle of the JIPDA is to merge together different hypotheses, such that in the end one is left with a single Gaussian for each track. This requires the marginal probabilities

$$\beta_{j,k}^t \propto \sum_{a_k \text{ s.t. } a_k(t)=j} P\{\mathbf{a}_k|\mathbf{Z}_{1:k}\}. \tag{3.66}$$

Once the marginal probabilities are calculated, the posterior Gaussian approximation for each track can be found according to

$$p_k^t(\mathbf{x}_k^t) \propto \sum_{j=0}^{m_k} \beta_{j,k}^t \mathcal{N}\left(\mathbf{x}_k^t; \hat{\mathbf{x}}_k^{t,j}, \mathbf{P}_k^{t,j}\right) \approx \mathcal{N}\left(\mathbf{x}_k^t; \hat{\mathbf{x}}_k^t, \mathbf{P}_k^t\right). \tag{3.67}$$

The state and covariance update equations are subsequently done in the same way as for the PDAF, with (3.38) and (3.40).

Posterior Existence and Visibility Probabilities

The final step in the JIPDA tracking algorithm is to update the existence and visibility probabilities. First, calculate the quantities

$$\varepsilon_{0,k}^t = \frac{\left(1 - P_D \bar{\eta}_k^t\right)}{1 - \bar{\varepsilon}_k^t P_D \bar{\eta}_k^t} \bar{\varepsilon}_k^t \tag{3.68}$$

$$\eta_{0,k}^t = \frac{\left(1 - P_D\right)}{1 - P_D \bar{\eta}_k^t} \bar{\eta}_k^t \tag{3.69}$$

representing the posterior existence and visibility probabilities conditioned on no measurement originating from target t. The marginal existence and visibility probabilities can subsequently be found by

$$\varepsilon_k^t = \beta_{0,k}^t \varepsilon_{0,k}^t + \sum_{j=1}^{m_k} \beta_{j,k}^t \tag{3.70}$$

$$\eta_k^t = \frac{1}{\varepsilon_k^t} \left(\beta_{0,k}^t \varepsilon_{0,k}^t \eta_{0,k}^t + \sum_{j=1}^{m_k} \beta_{j,k}^t \right). \tag{3.71}$$

Chapter 4

Implementation

In order to implement the target tracking system on the ReVolt model ship, some practical issues must be given some attention. This chapter describes the integration of the sensors in the ReVolt model ship, the intrinsic calibration of the camera, as well as the extrinsic calibration between the camera and the lidar. Furthermore, the coordinate systems used as well as the transformation of measurements to a common world frame is described. The convolutional neural network used to detect ships in images is briefly explained, as well as some details concerning the implementation and use of the detector in MATLAB. The clustering method used on the lidar data is described, and at the end the complete target tracking framework used in this thesis is presented, including the motion model for the targets as well as the sensor models used.

4.1 Sensors, Hardware and Processing Pipeline

The camera used on ReVolt is a single Point Grey Blackfly GigE color camera, with a 1/3" Sony ICX445 CCD sensor and a resolution of 1288×964 pixels. The camera is equipped with a fixed focal length low-distortion wide-angle lens, with a 125° field of view when paired with a 1/3" sensor. The camera is powered by Power over Ethernet (PoE), and exposes 6 general purpose I/O pins which can be used to control the operation of the camera. The key specifications for the camera is listed in table 4.1. The specifications are pulled from the technical reference manual for the camera [48]. The lidar is connected to the system via its interface box, providing connections for power and Ethernet. The specifications for the Velodyne VLP-16 were listed in section 2.3.1, and will not be repeated here. The existing control system on ReVolt runs on the Robot Operating System (ROS) [49], this makes ROS a natural choice for the sensor drivers and data acquisition in this thesis

Model	BLFY-PGE-13S2C-CS
Resolution	1288 × 964 pixels (1.3 Megapixels)
Framerate	30 frames per second
Sensor	SONY ICX445, CCD, 1/3”
Readout Method	Global shutter
Interface	GigE PoE
Lens Mount	CS-mount

Table 4.1: Point Grey Blackfly specifications.

as well.

4.1.1 Sensor Integration in ReVolt

The two sensors used in this thesis was integrated into the existing system on board the ReVolt model ship. The ReVolt is equipped with an on-board computer running the existing control system, as well as sensors used in the navigation system. The ships position is provided by a Hemisphere VS330 GNSS Compass with real-time kinematic (RTK) differential GNSS, giving a typical position accuracy of < 0.1 meters [50]. The ship is also equipped with a Xsens MTi-G-710 GNSS/INS inertial navigation system, used to provide the orientation of the ship relative to a local North East Down (NED) reference frame following the ship.

The ReVolt model has two targa hoops mounted on the deck, one close to the bow and the other close to the stern of the ship. Originally these targa hoops were used as mounts for the two antennas of the Hemisphere GNSS, but they also provide a convenient mounting placement for the sensors. The mechanical workshop at the Department of Engineering Cybernetics (ITK) at NTNU designed and built a mounting bracket for the lidar and the camera, and performed some modifications to the fore targa hoop to accommodate the sensor bracket. The sensor bracket was mounted as shown in figure 4.1. The lidar was placed on top, to give it a clear, unobstructed view of the surroundings. The camera was mounted directly below, looking straight ahead. Both the camera and the lidar send their data over a network connection. The camera sends its data as UDP packets, while the lidar sends its data as TCP packets. The camera is also powered via the Ethernet cable through PoE. For this reason a network switch with PoE capabilities was used in connecting the sensors to the existing system. The interconnections between the sensors and the existing control system is displayed in figure 4.2. Due to the high bandwidth of the data streams from both the lidar and the camera, and the limited processing power available on the on-board computer, the software drivers for the camera and lidar was run on a separate laptop which was installed in ReVolt. The specifications of the laptop is listed in table 4.2. Both the GNSS receiver and the Xsens INS send their data to the on-board computer via a

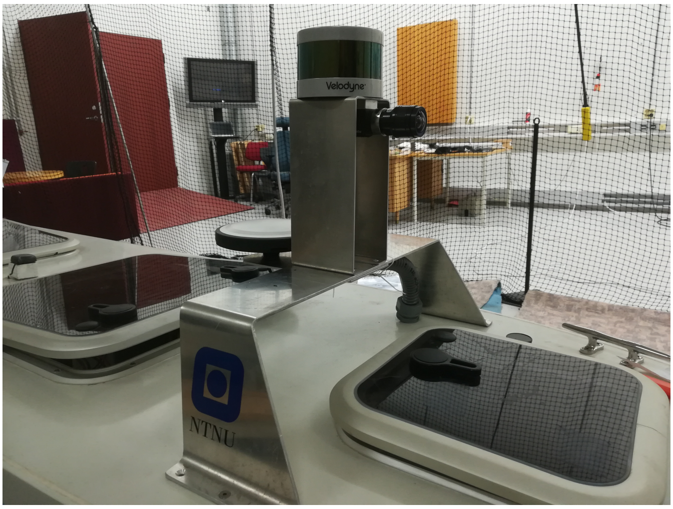


Figure 4.1: The sensor mounting bracket on the fore targa hoop.

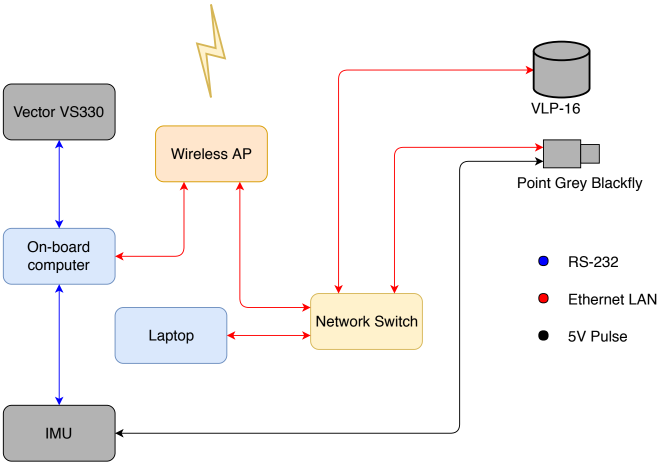


Figure 4.2: Physical interconnections.

serial RS-232 connection. The on-board computer is connected to a wireless access point, providing a connection to the system from the outside. The new PoE network switch was connected to this access point, providing a connection between the laptop processing the lidar and camera data and the on-board computer. The Xsens INS also has a syncout pin providing a means to send trigger signals to other devices at a set interval. This trigger signal was connected to the trigger input of the camera for synchronization, and set up to trigger at a frequency of 10 Hz.

Model	HP zbook 15
CPU	Intel Core i7-4800MQ 8×2.7 GHz
Memory	8 GB RAM
Graphics	Quadro K2100M/PCIe/SSE2
Storage	250 GB Solid State HDD
Operating System	Linux Ubuntu 16.04 LTS (Xenial)

Table 4.2: Laptop computer specifications.

4.1.2 Software Drivers and Synchronization

The software drivers used for the sensors are based on open-source ROS packages developed by the ROS community. The driver for the camera is the pointgrey_camera_driver package [51], which depend on the FlyCapture2 SDK [52] for interfacing with the camera. If the FlyCapture2 SDK is not installed on the target computer, the ROS package will download and install it automatically during the build process. The camera driver allows the user to configure parameters such as the triggering mode, shutter duration, frame rate and so on through a launch file. Each time a new image is received, the driver publish the image onto the ROS network as a `sensor_msgs/Image` topic. The driver for the lidar is the velodyne_driver package [53], which continuously receive and decode the TCP packets from the lidar. The driver converts the measurements received from polar to cartesian coordinates in the lidar frame of reference, as in (2.27) in section 2.3.1. All the points received corresponding to a full 360° scan are subsequently published to the ROS network as a `sensor_msgs/PointCloud2` topic. Since the sensors in the navigation system as well as the lidar sends their data continuously, outside the users control, the time-stamps upon reception in ROS are used to temporally align the data. The camera can be triggered externally by the user, and this was set up to trigger upon the reception of the first TCP packet in the first scan received from the lidar, triggering at the same frequency as the lidar. Both the lidar and the camera have a sampling frequency of 10 Hz.

4.2 Camera Calibration

In order to associate world points with pixel values in an image, the intrinsic calibration parameters and the radial distortion parameters described in section 2.1.1 is needed. ROS has a built-in camera calibration software package, which allows easy calibration of a camera using a checkerboard calibration target. The camera calibration package uses OpenCV camera calibration [23], a description of the software and its functions can be found in [54]. The intrinsic camera model is inspired by Heikkila and Silven [21], while the main calibration procedure is inspired by Zhang [17]. Using the calibration software is straightforward; the user must provide a checkerboard pattern with squares of known dimensions, and while the camera publishes images as a ROS topic of type `sensor_msgs/Image`, the camera calibration can be started by running

```
$ rosrun camera_calibration cameracalibrator.py
        --size 10x7 --square 0.100
        image:=/my_camera/image camera:=/my_camera
```

The size parameter tells the calibration software how many inner corner points there are in the checkerboard. In this example, a 11×8 checkerboard was used, where there are 10×7 inner corners. The square parameter describes the side length of the checkerboard squares in millimeters. The image parameter refers to the topic name where the camera images are being published, while the camera parameter refers to the namespace of the camera node in ROS. The camera matrix was found as

$$\mathbf{K} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 344.033691 & 0 & 624.013722 \\ 0 & 343.379790 & 482.967792 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.1}$$

and the distortion parameters were

$$\begin{bmatrix} k_1 & k_2 & p_1 & p_2 \end{bmatrix} = \begin{bmatrix} 0.007694 & -0.004260 & 0.000716 & -0.000181 \end{bmatrix}, \tag{4.2}$$

where k_1, k_2 are the radial distortion parameters, and p_1, p_2 are the tangential distortion parameters.

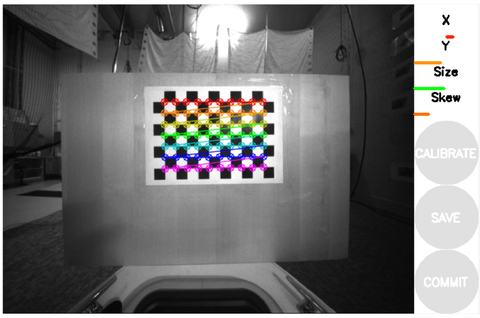


Figure 4.3: A screenshot of the camera calibration procedure in ROS.

4.3 Lidar-Camera Calibration

In order to associate the coordinates of measurements from the lidar with pixels in images captured with the camera, a transformation between the two coordinate systems must be found. The camera coordinate system is described in section 2.1.1 and illustrated in figure

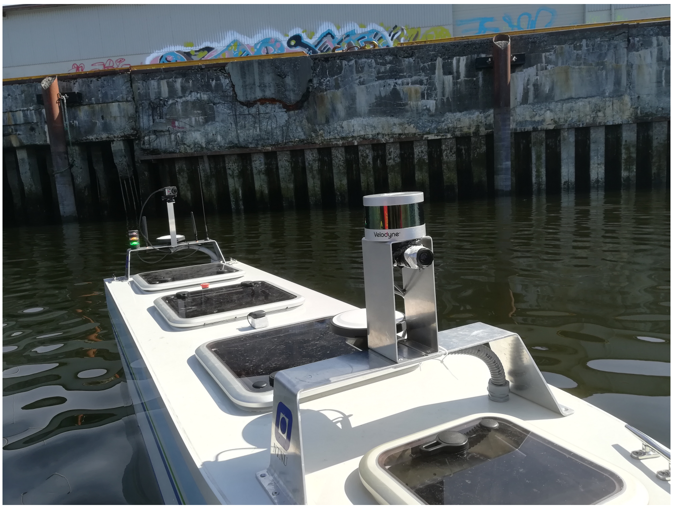


Figure 4.4: The camera and lidar mounted together on the ReVolt scale model ship.

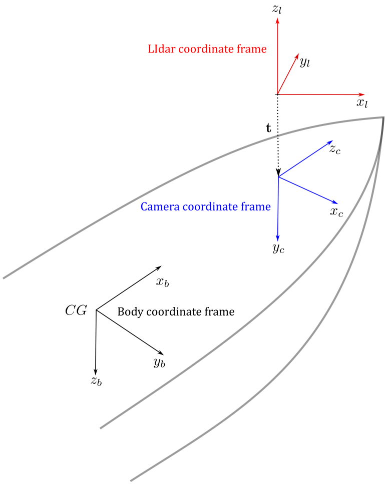


Figure 4.5: The lidar and camera coordinate systems mounted on ReVolt.

2.1, while the lidar coordinate system is described in section 2.3.1 and illustrated in figure 2.11. The sensors are mounted together as illustrated in figure 4.4. The rigid-body transformation between the lidar and the camera frame of reference consists of a rotation and a translation, for a total of 6 extrinsic calibration parameters. Dhall et al. [55] proposed an accurate and repeatable method to estimate the extrinsic calibration parameters using 3D-3D correspondences, which is used in this thesis. The method uses augmented reality markers to generate 3D points in the camera frame of reference. The proposed method uses ArUco markers [56] attached to rectangular planar surfaces, which given a calibrated camera provides the rotation and translation between the camera and the center of the marker.

The ArUco markers are special encoded binary patterns that facilitate error detection and correction in the marker patterns themselves. Two examples of ArUco markers are shown in figure 4.6. Since the marker size is known, once a marker has been detected it is possible to estimate its pose (with respect to the camera) by iteratively minimizing the reprojection error of the corners of the marker. The authors of [56] provides this functionality in a free open-source software library [57]. The authors of [55] provides their calibration software as a ROS package. A detailed description on how to use the package can be found in [58].

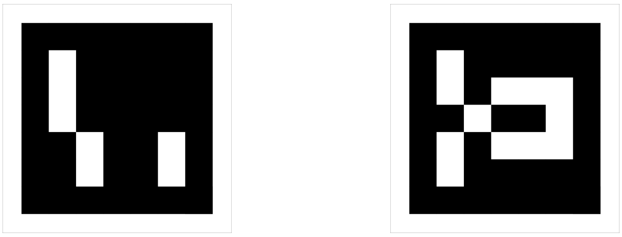


Figure 4.6: An example of two ArUco markers. The left marker has marker ID 10, the right marker has marker ID 100.

4.3.1 Calibration Setup

In the calibration, two ArUco tags were printed and attached to two cardboard planes of known dimensions. To ensure that enough points from the lidar was hitting the cardboard plane such that a line could be fitted, the cardboard planes were hung at a distance of approximately 1.5 meters, and rotated approximately 45° relative to the floor. At this distance, the cardboard planes covered almost the entire vertical spread of the laser beams from the lidar. The physical calibration setup is shown in figure 4.7. All distance measures given in this section have units of meters.

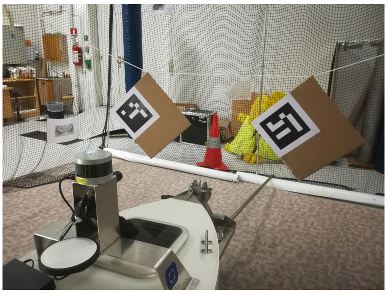


Figure 4.7: The lidar-camera calibration setup.

3D Points in the Camera Frame

Given that the 3D locations of the center of the ArUco markers are known, as well as the dimensions of the marker plane, the location of the corners of each plane can be calculated. The ArUco tags give the rotation and translation, $[\mathcal{R}|\mathbf{t}]$, between the camera and the center of the ArUco marker. The location of the corners in the markers frame of reference (where the marker plane is the plane $z = 0$, and the origin is in the center of the marker) is known, and the corner points can be transformed to the cameras frame of reference using the transformation found through the ArUco marker.

3D Points in the Lidar Frame

For the lidar, the corner points of the marker plane is found by first detecting the edges of the cardboard. For each edge of the marker plane, the user draws a polygon around the points corresponding to a single edge, for a total of 8 edges when using two ArUco markers. Random Sample Consensus [59] (RANSAC) is then used to fit lines to each edge, using the points marked by the user. When the line equations for the four edges are found, the corners of the marker planes can be found as the intersection of the edge lines. These lines may not intersect exactly, in this case the midpoint of the shortest line segment between the two lines is used as the corner point.

Since the point correspondences are known, a closed-form solution to the transformation between the coordinate frames can be found. The calibration procedure uses the Kabsch algorithm [60] to find the rotation between the two point clouds, and the translation can be found once the rotation is known. The initial translation between the vector sets describing the respective points is removed by translating the centroid of each vector set to the origin. Let P_i denote the vector of the i^{th} point in the lidar frame, and Q_i denote the vector of the i^{th} point in the camera frame. Let the centroid of each vector set be denoted by $\bar{P}$ and $\bar{Q}$. The rotation from the lidar frame to the camera frame can then be found by solving

$$\mathcal{R} = \underset{\mathcal{R} \in \mathcal{SO}(3)}{\text{arg min}} \sum_{i=1}^{n} \left|\left|(\mathcal{R}(P_i - \bar{P}) - (Q_i - \bar{Q}))\right|\right|^2 \tag{4.3}$$

where $\mathcal{SO}(3)$ denotes the 3D rotation group, ensuring that the solution is a valid rotation matrix. Now let P and Q be two matrices, where the i^{th} column is the vector $P_i - \bar{P}$ and $Q_i - \bar{Q}$, respectively. Compute the cross-covariance matrix H as

$$H = P^T Q. \tag{4.4}$$

Next, calculate the singular value decomposition (SVD) of H,

$$H = USV^T. \tag{4.5}$$

To decide whether or not we need to correct our rotation matrix to ensure a right-handed coordinate system, compute the determinant

$$d = \det VU^T. \tag{4.6}$$

The optimal rotation matrix can then be calculated as

$$\mathcal{R} = V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & d \end{bmatrix} U^T. \tag{4.7}$$

Once the rotation between the two coordinate frames is found, the translation between the frames can be found as the distance between matched points in the aligned frames.

Calibrated Values

The rotation matrix between the lidar frame and the camera frame was found using the calibration routine as

$$\mathcal{R}_l^c = \begin{bmatrix} 0.764258 & -0.644885 & 0.00575709 \\ 0.0236893 & 0.0191512 & -0.999536 \\ 0.644476 & 0.764039 & 0.0299133 \end{bmatrix} \tag{4.8}$$

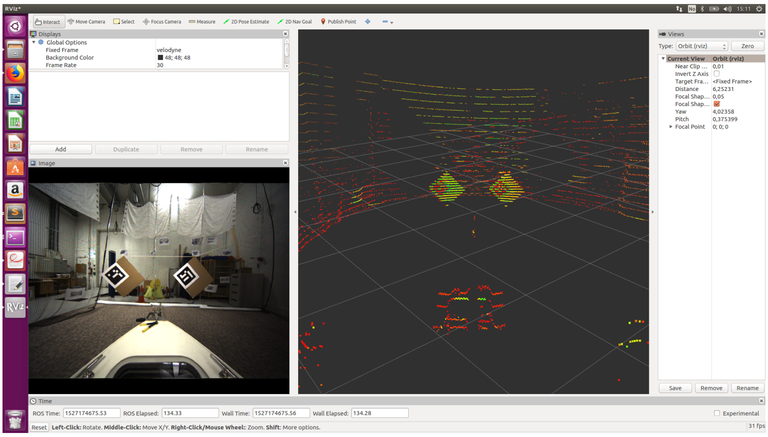
and the translation from the camera frame to the lidar frame (in the camera frame of reference) was

$$\mathbf{t}_l^c = \begin{bmatrix} -0.00117 \\ -0.10957 \\ -0.10682 \end{bmatrix} \tag{4.9}$$

Using homogeneous coordinates, this can be combined into a homogeneous transformation matrix

$$\mathcal{T}_l^c = \begin{bmatrix} \mathcal{R}_l^c & t_l^c \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.10}$$

A screenshot from ROS showing the camera view as well as the point cloud during the calibration is shown in figure 4.8.



Global Options
Fixed Frame: velodyne
Background Color: 48; 48; 48
Frame Rate: 30

Type: Orbit (rviz)
Current View: Orbit (rviz)
Near Clip ...: 0,01
Invert Z Axis: □
Target Fra...: <Fixed Frame>
Distance: 6,25231
Focal Shap...: 0,05
Focal Shap...: ☑
Yaw: 4,02358
Pitch: 0,375399
Focal Point: 0; 0; 0

ROS Time: 1527174675.53 ROS Elapsed: 134.33 Wall Time: 1527174675.56 Wall Elapsed: 134.28 □ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

Figure 4.8: The camera view of the ArUco markers along with the 3D point cloud from the lidar, displayed in RViz (ROS).

4.4 Transformation to a Common World Frame

Section 4.3 described the method used to find the transformation between the lidar and the camera frame of reference. The measurements received by the camera and the lidar are transformed to a common North East Down (NED) world frame, defined relative to the Earth's reference ellipsoid. The z-axis points downward normal to the tangent plane of the ellipsoid, and the x-axis points towards true north. The y-axis points eastward, completing the right-handed coordinate system.

The ReVolt model ship has a defined BODY coordinate system, which moves and rotates with the ship. As described in the previous section, the transformation between the lidar and the camera was found with a calibration procedure. What remains is to find the transformation between the camera frame and the BODY frame, as well as the transformation between the BODY and the NED frame. As illustrated in figure 4.5, the BODY x-axis is aligned with the camera z-axis, and the body y-axis is aligned with the camera x-axis. The rotation from the camera frame to the BODY frame is therefore

$$\mathcal{R}_c^b = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{4.11}$$

The translation between the origin of the camera frame and the origin of the BODY frame was hand measured to be

$$\mathbf{t}_c^b = \begin{bmatrix} 0.87 \\ 0 \\ -0.36 \end{bmatrix} \tag{4.12}$$

in the BODY frame of reference. The transformation matrix between the camera and the

BODY frame is thus

$$\mathcal{T}_c^b = \begin{bmatrix} \mathcal{R}_c^b & \mathbf{t}_c^b \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}. \tag{4.13}$$

The units of translation are in meters. The navigation system onboard ReVolt measures the roll (ϕ), pitch (θ) and yaw (ψ) angles of the BODY frame relative to the NED frame. The roll, pitch and yaw angles are illustrated in figure 4.9. The rotation matrix from the BODY frame to the NED frame is given as [61]

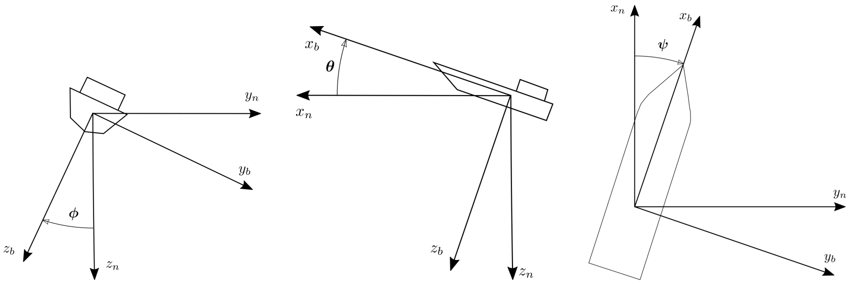


Figure 4.9: Roll, pitch and yaw rotations of the BODY frame.

$$\mathcal{R}_b^n = \begin{bmatrix} c_\psi c_\theta & -s_\psi c_\phi + c_\psi s_\theta s_\phi & s_\psi s_\phi + c_\psi s_\theta c_\phi \\ s_\psi c_\theta & c_\psi c_\phi + s_\psi s_\theta s_\phi & -c_\psi s_\phi + s_\psi s_\theta c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{4.14}$$

where c_ϕ is $\cos(\phi)$ and s_ϕ is $\sin(\phi)$. The position of the origin of the BODY frame is given by the navigation system in geodetic coordinates on the World Geodetic System (WGS) 84 reference ellipsoid as latitude, longitude and altitude. This must then be transformed to a position in the local NED frame. Let μ_0, l_0 and h_0 be the latitude, longitude and height of the origin of the local NED frame in geodetic coordinates, respectively. Further, let μ_b, l_b and h_b be the latitude, longitude and height of the origin of the BODY frame in geodetic coordinates. The transformation to NED coordinates starts by calculating the Earth-centered Earth-fixed (ECEF) cartesian coordinates of both the origin of the local NED frame as well as the BODY frame by [61]

$$\begin{bmatrix} x_e \\ y_e \\ z_e \end{bmatrix} = \begin{bmatrix} (N+h)\cos(\mu)\cos(l) \\ (N+h)\cos(\mu)\sin(l) \\ \left(\frac{r_p^2}{r_e^2}N + h\right)\sin(\mu) \end{bmatrix}, \tag{4.15}$$

where N is the prime vertical radius of curvature,

$$N = \frac{r_e^2}{\sqrt{r_e^2\cos^2(\mu) + r_p^2\sin^2(\mu)}}, \tag{4.16}$$

where $r_e = 6378137\text{m}$ and $r_p = 6356752\text{m}$ are the equatorial and polar Earth radii, which are WGS-84 parameters. The position of a point in a local NED frame can now be found by the rotation

$$\begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = \mathcal{R}_e^n \begin{bmatrix} x_{e,b} - x_{e,0} \\ y_{e,b} - y_{e,0} \\ z_{e,b} - z_{e,0} \end{bmatrix} \tag{4.17}$$

where

$$\mathcal{R}_e^n = \begin{bmatrix} -\sin(\mu_0)\cos(l_0) & -\sin(l_0) & -\cos(\mu_0)\cos(l_0) \\ -\sin(\mu_0)\sin(l_0) & \cos(l_0) & -\cos(\mu_0)\sin(l_0) \\ \cos(\mu_0) & 0 & -\sin(\mu_0) \end{bmatrix} \tag{4.18}$$

and $x_{e,0}$, $y_{e,0}$ and $z_{e,0}$ are the ECEF coordinates of the origin of the NED frame, and $x_{e,b}$, $y_{e,b}$ and $z_{e,b}$ are the ECEF coordinates of the origin of the BODY frame. Now that the position of the origin of the BODY frame in the NED frame is found, the transformation of points in the BODY frame to the NED frame can be expressed as

$$\mathcal{T}_b^n = \begin{bmatrix} \mathcal{R}_b^n & \mathbf{t}_b^n \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.19}$$

where $\mathbf{t}_b^n$ is the position of the BODY frame origin in the NED frame, found by equations (4.15)-(4.17). To summarize, the transformation from the lidar frame of reference to the local NED frame is found by the compound transformation

$$\mathcal{T}_l^n = \mathcal{T}_b^n \mathcal{T}_c^b \mathcal{T}_l^c \tag{4.20}$$

and the transformation from the camera frame of reference to the local NED frame is found by the transformation

$$\mathcal{T}_c^n = \mathcal{T}_b^n \mathcal{T}_c^b. \tag{4.21}$$

4.5 Visual Detection based on Faster R-CNN

The visual detection of boats in images used in this thesis is based on the framework of Faster R-CNN, developed by Ren et al. [62]. This framework was used due to the availability of pre-trained networks developed during a Master thesis project at NTNU [63]. A brief introduction to the framework of Faster R-CNN is given in this section, some background on how the implemented model was trained, and some details concerning the implementation of the network in MATLAB. For a detailed explanation of the full framework of Faster R-CNN the reader is referred to [62].

4.5.1 Faster R-CNN

Faster R-CNN is a region-based CNN, employing *Region Proposal Networks* (RPN) to form a single, unified network for object detection. Faster R-CNN is composed of two modules. The first module is a deep fully convolutional neural network which proposes regions, and the second module is the Fast R-CNN detector [64] that uses the proposed

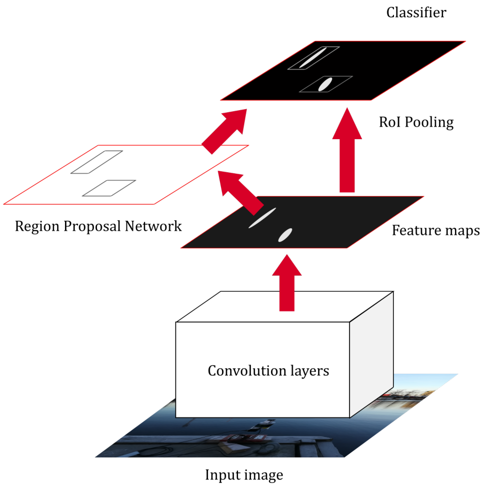


Figure 4.10: Faster R-CNN as a single unified network for object detection. The figure is adapted from [62].

regions in classification. Figure 4.10 illustrates the system of Faster R-CNN. The Region Proposal Network takes an image of any size as inputs, feeding it through convolutional layers to generate feature maps. The area of activation is then fed into the RPN, which outputs object proposals in the form of rectangular regions, each with a objectness score. The objectness score is a measure of membership to a set of object classes versus background. The rectangular regions are subsequently fed into the classification layers where the classification score is obtained. An important feature of Faster R-CNN is that the RPN shares convolutional layers with the object detection networks, reducing the cost for computing proposals, making it attractive for real-time applications, such as remote sensing for an ASV. In their paper, Ren et al. investigated two models for the classification network, the 5 convolutional layer deep Zeiler and Fergus model [65] dubbed ZF-net, and the 16 convolutional layer deep Simonyan and Zisserman model [66], dubbed VGG-16. In this project, the VGG-16 model is used, as this network achieved the best results in the Master thesis work of Tangstad [63].

4.5.2 Training and validation data

The network used in the thesis is the best-performing model developed in Tangstads thesis work. This model is trained as a single-class classifier, trained to detect boats in images. The network was trained using the 2007 and 2012 VOC (Visual Object Challenge) [67] dataset and a selection of images pulled from the image database imagenet [68]. In the images used from these image sets, the only ground-truth label and bounding boxes are of the boat-class. In addition, a custom dataset was generated during the thesis work, with images of boats from the Trondheim harbor and at open sea, which was included in the training and test sets. For details on the training, and the performance of the trained network, the reader is referred to [63].

4.5.3 Implementation Aspects

A complete MATLAB implementation of Faster R-CNN is available on Github. In order to run Faster R-CNN, there are some software requirements that need to be met. The steps are listed in the Github repository for the framework [69], and will be repeated here for convenience. The steps are setup-dependent, depending on whether or not the framework will be executing on a GPU or a CPU, and also on the version of GPU used. The Faster R-CNN implementation available at the Github repository uses Caffe [70], which is a deep-learning framework utilizing Nvidia CUDA for parallel computing, developed by Berkeley AI Research. Caffe is well documented online, with installation instructions, tutorials and more [71].

In order to use Faster R-CNN, a Caffe MEX file is needed. A MEX file is a dynamically linked subroutine that the MATLAB interpreter loads and executes, enabling MATLAB to utilize Caffe routines. A pre-compiled Caffe version for Nvidia CUDA version 6.5 is available in the repository, but for newer Nvidia GPUs a newer version of CUDA will need to be compiled along with Caffe. If a new compiling of Caffe is necessary, all the steps needed are provided in the Faster R-CNN branch of Caffe on Github [72]. If a GPU is utilized for parallel computing, a GPU with minimum 8 GB of memory is needed to run the VGG-16 version. In this project Caffe was recompiled with CUDA version 9.0, due to CUDA version 6.5 not supporting the newer Pascal architecture on the Nvidia GTX 1070 graphics card used. MATLAB is also needed to run the Faster R-CNN implementation. Once all requirements are met, the steps necessary before testing are

1. Run `faster_rcnn_build.m`
2. Run `startup.m`

Once this is done, the user must provide a valid network (pre-trained networks are available on the Github repository), and the model is ready for use.

4.5.4 Image Preprocessing

Before the images captured by the camera were fed into the Faster R-CNN classifier, some preprocessing was necessary. The images captured by the camera suffered from vignetting [73], causing the image brightness to drop in the image periphery. Figure 4.12a shows an example of an image captured, where the edges of the image are very dark compared to the center of the image. To remedy this situation, the Hadamard product (also known as the Schur product) between the image and a matrix $\mathbf{C}$ with dimensions matching the image resolution was taken. The Hadamard product between two matrices is the element-wise multiplication of the entries of each matrix. The matrix $\mathbf{C}$ approximates a inverted Gaussian function, where the ij^{th} entry was given by

$$\mathbf{C}_{ij} = 3.5 - 3.5 \exp\big(-\frac{(i-483)^2 + (j-624)^2}{2 \times 350^2}\big) + 1.5. \tag{4.22}$$

The image consists of three color channels, and for each color channel, the new corrected image I_c is given by

$$I_c = I \circ \mathbf{C} \tag{4.23}$$

where $\circ$ denotes the Hadamard product. The inverted Gaussian function used in the correction is illustrated in figure 4.11. The center of the Gaussian $(483, 624)$ coincides with the principal point in terms of pixel dimensions, found through the camera calibration. The other values in (4.22) were found through manual tuning of the parameters and visualizing their effect on the image. The effect of multiplying the image with the inverted Gaussian matrix can be seen in figure 4.12b. As seen in the figure, the vignetting effect is severely reduced.

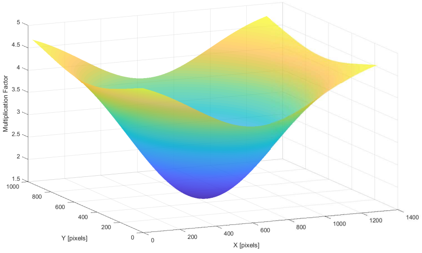
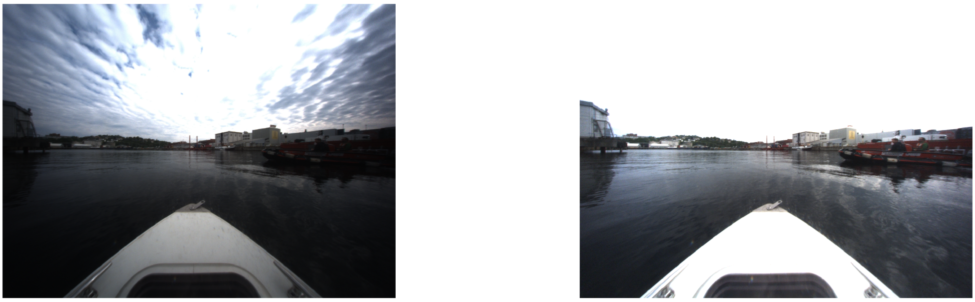


Figure 4.11: Inverse Gaussian function used to brighten edges of images.



(a) Image suffering from vignetting.

(b) Corrected image.

Figure 4.12: Darkened edges in the image caused by vignetting, and the corresponding corrected image.

4.6 Lidar Segmentation

The tracking filter employed in this thesis is based on the point target assumption, meaning that each target generate at most one measurement. The lidar generate a point cloud with potentially thousands of points in each scan, where a single target could be the source of hundreds of points. Figure 4.13 illustrates an example where a ship was in close proximity to the lidar, resulting in many point measurements. For this reason, clustering is performed on the raw point cloud, to maintain the at-most-one-measurement assumption in the tracking framework. Clustering is the process of partitioning a finite unlabeled data set into a finite set of clusters, where the data in each cluster are similar in some sense [74]. For the task of clustering points originating from a target, it makes sense to assume that points that are spatially close to one another belong to the same object.

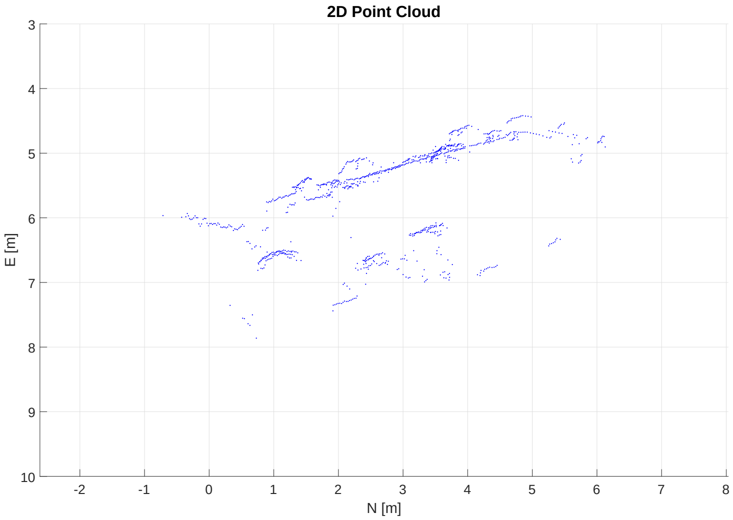


Figure 4.13: Lidar returns from a ship in close proximity, projected down to the x-y plane.

While there are many different clustering techniques, they can roughly be classified as either hierarchical clustering or partitional clustering [75]. Hierarchical clustering methods partition the data into a nested series of partitions based on a criterion for merging or splitting clusters based on similarity. Partitional clustering methods on the other hand, identify the partition that optimizes a clustering criterion, with no hierarchical structure. In this thesis, a density-based partitional clustering approach is employed. Density-based means that for each point in a cluster the neighborhood of a given radius must contain at least a specified minimum number of points. This makes the clustering algorithm robust against random noise, where the hypothesis is that random noise will seldom appear as dense clusters, but rather randomly spread out. The algorithm used is Density Based Spatial Clustering of Applications with Noise (DBSCAN) by Martin Ester et al. [76]. The

DBSCAN algorithm is a well known and much used clustering algorithm, one example being [77], where they use DBSCAN to cluster lidar data to identify road lane markings.

4.6.1 The DBSCAN Algorithm

Before we delve into the algorithm itself, some concepts central to the algorithm must be explained. The concepts and algorithm presented in this section is based on the original paper by Martin Ester et al. [76] as well as [78]. The DBSCAN algorithm relies on two parameters; ε, which determines how close points should be to be considered a part of the same cluster, and *minPts*, which determine how many neighbors within a ε neighborhood a point should have to be considered a part of the same cluster. Furthermore, it uses the concept of *direct density reachability*, *density reachability*, and *density connectivity*.

A point p is *direct density reachable* from a point q with respect to ε, *minPts* if p is in the ε-neighborhood of q, and the ε-neighborhood of q contains at least *minPts* points.

A point p is *density reachable* from a point q with respect to ε, *minPts* if there is a chain of points $p_1, \ldots, p_n$, $p_1 = q$, $p_n = p$ such that p_{i+1} is direct density reachable from p_i.

Two border points in the same cluster may not be density reachable from each other, since the core point condition might not hold for both. Two points p and q are considered *density connected* if there is a point o such that both p and q are density reachable from o with respect to ε and *minPts*. DBSCAN also use the definition of *core points* and *border points*. A core point is a point inside the cluster with at least *minPts* points in the ε-neighborhood, while a border point is, as the name implies, a point on the border of the cluster where this condition may not hold. These concepts are illustrated in figure 4.14, where points labeled C are core points, points labeled B are border points, and the point N is considered as noise. In this example, *minPts*=3, and the ε radius for each point is illustrated with circles. Direct density reachability is indicated with arrows.

The basic steps of extracting a single cluster in the DBSCAN algorithm can be summarized as follows: First select an arbitrary point from the dataset satisfying the definition of a core point as a seed. Second, retrieve all points that are density reachable from the seed obtaining the cluster containing the seed. The DBSCAN algorithm is illustrated as pseudocode in Algorithm 1. The algorithm is adapted from [78]. The RangeQuery-function finds all neighbors of the point p within the radius ε. The distance used in the range query does not necessarily have to be euclidean distance, however for the application of clustering lidar points, euclidean distance makes the most sense, and is what is used in this thesis.

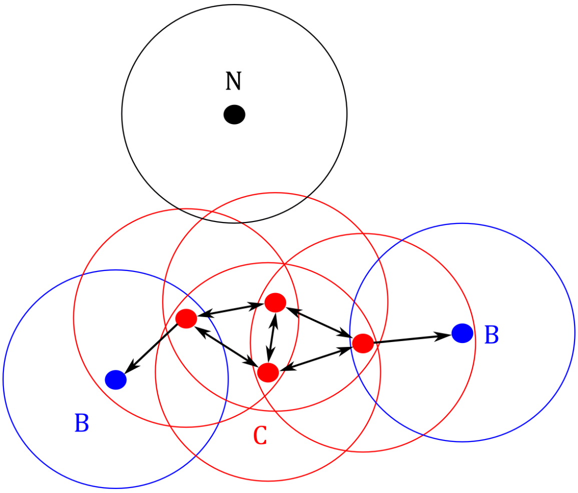


Figure 4.14: Illustration of the DBSCAN cluster model. The figure is adapted from [78].

```
input  : Dataset $P$, Radius $\varepsilon$, Density threshold $minPts$, Distance function $dist$
output: Point labels $label$
foreach $point\ p$ in $P$ do                          // Iterate over all points
    if $label(p) \neq undefined$ then               // Skip prev.  visited points
        continue
    end
    Neighbors N ← RangeQuery (P, p, $dist$, $\varepsilon$);           // Get neighbors
    if $|N| < minPts$ then                               // Label as noise
        label(p) ← Noise;
        continue
    end
    c ← next cluster label;                              // Start new cluster
    label(p) ← c;
    Seed set S ← N \{p};                               // Expand neighborhood
    foreach $q$ in $S$ do                           // Iterate over all neighbors
        if $label(q)$ = $Noise$ then label(q) ← c;
        if $label(q) \neq undefined$ then continue;
        Neighbors N ← RangeQuery (P, q, $dist$, $\varepsilon$);
        label(q) ← c;
        if $|N| < minPts$ then continue;                 // Core-point check
        S ← S ∪ N
    end
end
```

Algorithm 1: The DBSCAN Algorithm

4.6.2 The Modified Distance Parameter

Since the laser beams from the lidar get spread out as the range increases, the constant parameter ε was replaced with a function which calculates ε based on the range from the lidar to the point. Close targets will give many returns closely spaced together, while returns from targets further away will be more spread out. A similar idea was proposed by [77]. In this thesis, it was observed that a constant ε led to over-segmentation of targets when they were close, where the boats were clustered together with wakes. Increasing ε linearly with the range led to the same problem, where two separate targets were clustered together despite being separated by several meters, due to the large clustering radius ε. A non-linear function was therefore used. The distance function used in this thesis is

$$\varepsilon(r) = \frac{1}{2}\ln(r) \tag{4.24}$$

where r is the range from the lidar to the point in consideration. This function was found through experimentation. The function for $r \in [0, 100]$ meters is shown in figure 4.15. The distance function doesn't make sense for $r \leq 1$, however the lidar does not give returns below $r = 2$ meters, to avoid returns originating from the ReVolt ship itself. Figure 4.16 shows the result of clustering the points in figure 4.13 using DBSCAN, with *minPts*=3. The convex hull of the cluster is drawn as the red polygon, with its center as the red dot. As can be seen in the figure, all points belonging to the boat are captured in the cluster. In figure 4.17, a target at greater distance is clustered with DBSCAN. The returns from the lidar are more spread out in this case.

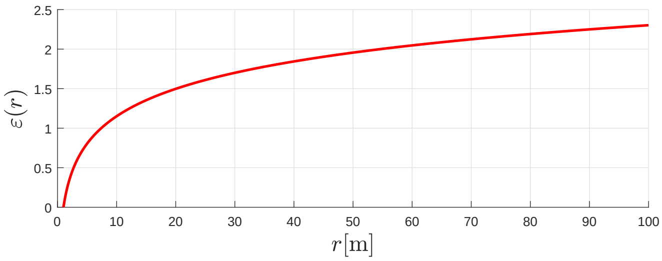


Figure 4.15: $\varepsilon(r)$ from 0 to 100 meters.

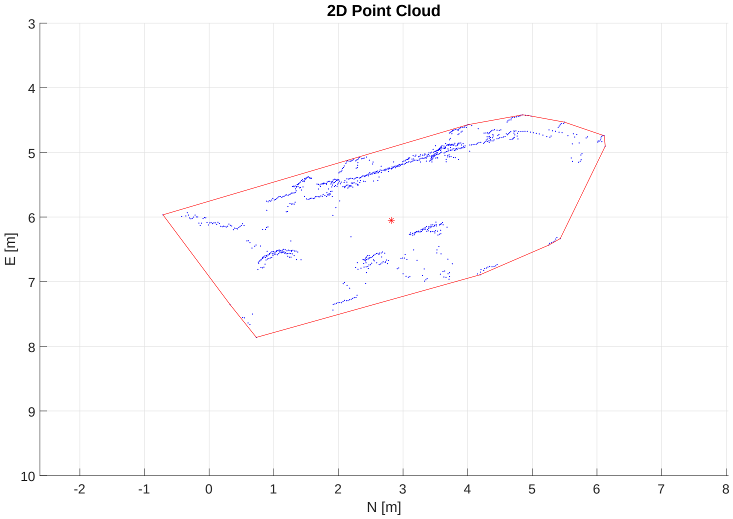


Figure 4.16: Lidar returns from figure 4.13, clustered with DBSCAN.

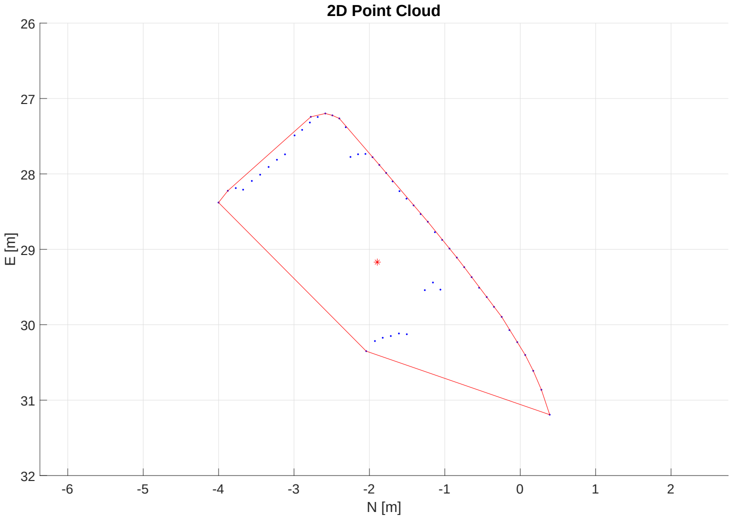


Figure 4.17: Lidar returns from a target at distance, clustered with DBSCAN.

4.7 Target Tracking Framework

This section aims to tie together the material presented thus far, describing the complete implemented target tracking system. First, the motion and sensor models used are presented, before going into some detail on the JIPDA implementation. The whole system at a high level is illustrated in figure 4.21.

4.7.1 Motion and Sensor Models

Constant Velocity Motion Model

The motion model used in the target tracking framework is the *constant velocity model*. The state vector of a target in the constant velocity model is given as $\mathbf{x} = [N, E, V_N, V_E]^T$, where N, E, V_N and V_E are the north and east positions and velocities of the target in a stationary world-fixed reference frame, respectively. The constant velocity (white noise acceleration) model in discrete time can be written as [41]

$$\mathbf{x}_{k+1} = \mathbf{F}_{cv}\mathbf{x}_k + \mathbf{w}_k \qquad p(\mathbf{w}_k) = \mathcal{N}(\mathbf{w}_k; 0, \mathbf{Q}_{cv}) \tag{4.25}$$

where the state transition matrix $\mathbf{F}_{cv}$ and process noise covariance matrix $\mathbf{Q}_{cv}$ is given as

$$\mathbf{F}_{cv} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.26}$$

$$\mathbf{Q}_{cv} = \sigma_{cv}^2 \begin{bmatrix} T^4/4 & 0 & T^3/2 & 0 \\ 0 & T^4/4 & 0 & T^3/2 \\ T^3/2 & 0 & T^2 & 0 \\ 0 & T^3/2 & 0 & T^2 \end{bmatrix} \tag{4.27}$$

where $T = t(k+1) - t(k)$ is the system sample time. The process noise strength σ_{cv} is selected according to the targets' expected maneuverability. The name ”white noise acceleration model” stems from the fact that accelerations along the north and east directions are modeled as white noise.

Sensor Models

As described in section 4.1.2, the lidar software driver converts the measurements received to cartesian coordinates. The measurements received are given in the lidar frame of reference, and transformed to the NED frame using (4.20). The object tracking system tracks objects in the 2D North-East plane, so only the N and E coordinates of points are used in tracking objects. The measurement model for the lidar is therefore a linear Cartesian measurement model of the form

$$\mathbf{z}_{k,l} = \mathbf{H}_l\mathbf{x}_k + \mathbf{v}_{k,l} \qquad p(\mathbf{v}_{k,l}) = \mathcal{N}(\mathbf{v}_{k,l}; 0, \mathbf{R}_l) \tag{4.28}$$

where $\mathbf{v}_k$ is the measurement noise, and $\mathbf{H}$ is given by

$$\mathbf{H}_l = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.29}$$

The measurement noise covariance matrix $\mathbf{R}_l$ was determined based on the extent of the targets used in the experiments described in the next chapter. Depending on the orientation of the target, the centroid of the cluster given by DBSCAN could wander along the length of the target in question. For example, if the aft of the target is directly facing the lidar, a scenario such as illustrated in figure 4.18 could occur. Here, the cluster centroid is centered on the stern of the target. As the largest target has a length of approximately 7.1 meters, the measurements were assumed to be distributed along the length of the target with 5σ probability. The measurement noise covariance matrix was therefore chosen to be

$$\mathbf{R}_l = \sigma_l^2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{4.30}$$

with $\sigma_l^2 = 0.5$. Experimental tuning also showed that this value for the measurement noise covariance gave good tracking results in comparison with other values.

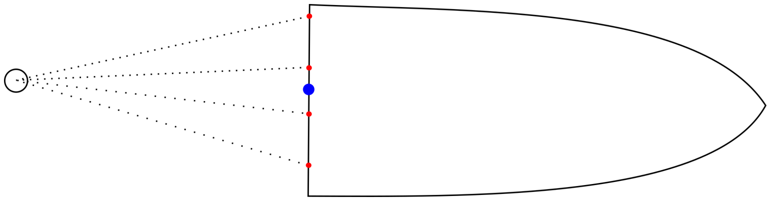


Figure 4.18: The cluster centroid centered on the stern of the target.

The camera measurement model is somewhat more complicated. As explained in section 2.1.1, there is a loss of information when projecting 3D points into the 2D image plane, and without additional information, a measurement in the image plane can only give a line of sight in the 3D world coordinate frame. Let $\bar{\mathbf{x}}_n = [N, E, D]^T$ be the predicted position of a target in the NED frame. Furthermore, let the coordinates in the camera frame be given by $\bar{\mathbf{x}}_c = [x_c, y_c, z_c]^T$. The line-of-sight angle relative to the z-axis in the camera frame of reference is then given by (see figure 4.19)

$$\psi_c = \tan^{-1}\left(\frac{x_c}{z_c}\right). \tag{4.31}$$

In order to update the innovation covariance, the Jacobian in terms of the state in NED coordinates is needed, as described in section 3.2.4. In terms of NED coordinates, the measurement can be expressed using the inverse transformation of (4.21). If

$$\mathcal{T}_c^n = \begin{bmatrix} \mathcal{R}_c^n & \mathbf{t}_c^n \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.32}$$

is the transformation from the camera frame to the NED frame, then the inverse transformation is given by

$$\mathcal{T}_n^c = \begin{bmatrix} (\mathcal{R}_c^n)^T & -(\mathcal{R}_c^n)^T\mathbf{t}_c^n \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} = \begin{bmatrix} \mathcal{R}_n^c & \mathbf{t}_n^c \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4.33}$$

Multiplying (4.33) with the position in NED coordinates, where D is taken to be zero (since the plane $D = 0$ is the plane parallel to the sea level, where the targets are tracked), and substituting into (4.31), we get

$$\psi_c = \tan^{-1}\left(\frac{r_{11}N + r_{12}E + t^n_{n,1}}{r_{31}N + r_{32}E + t^c_{n,3}}\right) \tag{4.34}$$

where r_{ij} is the ij^{th} entry of $\mathcal{R}^c_n$, $t^c_{n,i}$ is the i^{th} coordinate of $\mathbf{t}^c_n$, and N and E are the north and east NED coordinates.

The measurement model for the camera, assuming the measurement noise enters additively, can then be expressed as

$$\mathbf{z}_{k,c} = \mathbf{h}_c(\mathbf{x}_k) + \mathbf{v}_{k,c} \qquad p(\mathbf{v}_{k,c}) = \mathcal{N}(\mathbf{v}_{k,c}; 0, \mathbf{R}_c) \tag{4.35}$$

where the nonlinear measurement function $\mathbf{h}_c(\mathbf{x}_k)$ is given by (4.34). The Jacobian of $\mathbf{h}_c(\mathbf{x}_k)$ evaluated at the predicted state is used in computing the innovation covariance matrix, as described in section 3.2.4.

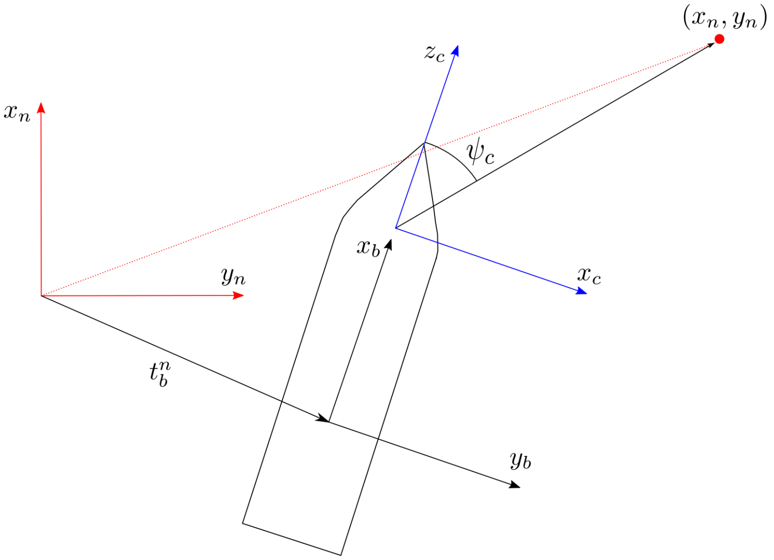


Figure 4.19: A point in the NED frame, and its relation to the line-of-sight angle in the camera frame.

4.7.2 Fusing the Measurements

The images captured by the camera are run through the Faster R-CNN convnet, which gives a set of bounding boxes with accompanying scores for each detected object in the image. The center of each bounding box is projected to the normalized image plane using the inverse camera calibration matrix $\mathbf{K}^{-1}$, and the line-of-sight angle for the center of the bounding box for each detection can subsequently be found by $\psi_c^d = \tan^{-1} x_c^d$, where x_c^d is the x-coordinate of the center of the d^{th} bounding box in the normalized image plane. In the normalized image plane we have that $z_c = 1$. The association probabilities between the predicted measurement angle for each target and the line-of-sight angle(s) for each detection are then computed using JIPDA in the same manner as for the lidar measurements. Using these association probabilities to update the target state has not been implemented in this thesis, and remains as future work.

4.7.3 Implementing JIPDA

A key element of the JIPDA filter is the computation of the marginal association probabilities (3.66), as described in section 3.3.3. A complicating factor in this is that the number of association hypotheses grows exponentially with respect to the number of measurements and tracks. Based on the number of measurements and tracks, a complete calculation of the marginal association probabilities may be infeasible, and complexity-mitigating steps should be made.

Calculating the Marginal Association Probabilities

A complexity-mitigating step in the multi-target case is to cluster tracks that share measurements together, and approximate the marginal association probabilities within each cluster. This avoids the computation of association probabilities that are unlikely, considering only plausible associations. The concept of a cluster of tracks is illustrated in figure 4.20. The number of tracks and measurements within each cluster may still make the calculations infeasible, so approximations may be necessary. Several methods for approximative evaluation of the association probabilities exist, such as in [79], where they exploit the fact that a lot of the structure is repeated among the different association hypotheses. This is however a patented method. The method used in the implemented tracking system is Murtys' M-best method [80], which generates the M-most probable association hypotheses for a given set of measurements and tracks.

Murtys' M-Best Method

Murtys' M-best method builds upon the classical assignment problem from combinatorial optimization, by casting the problem of finding the most probable assigment hypothesis as a weighted bipartite matching problem. The nodes on one side are the measurements, while the nodes on the other side are the targets, and each arch gives the log-likelihood that a given measurement should be assigned to a given target. Finding the best hypothesis is then reduced to the problem of finding the assignment that maximises the sum of the

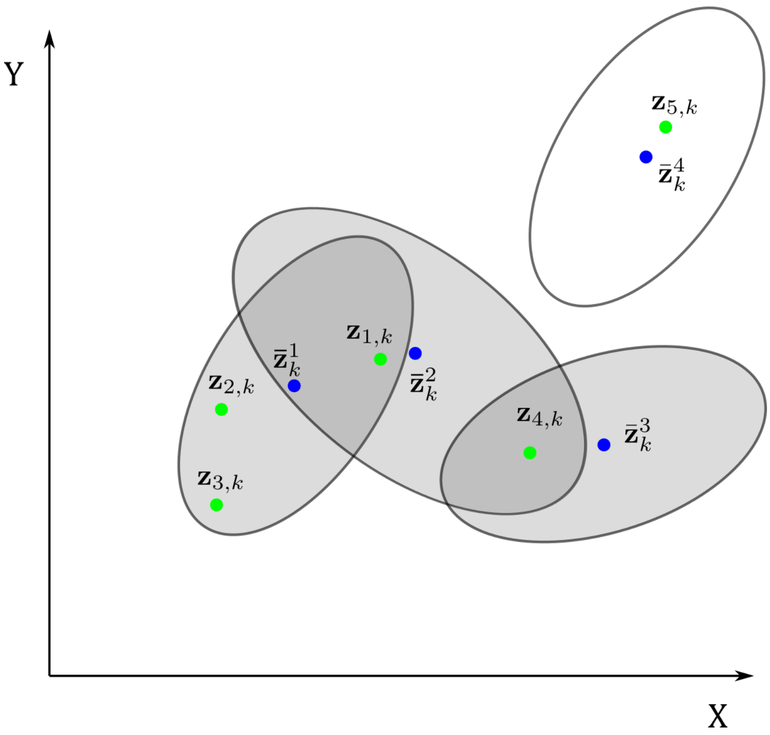


Figure 4.20: A set of tracks sharing measurements within their gate grouped into a cluster, indicated by the shaded validation regions.

log-likelihoods of the given assignment, which can be found in polynomial time. Murtys method maintains a list of of problem/solution pairs. Each pair consists of an assignment of measurements to tracks and a score matrix based on the association probabilities from the assignment hypothesis. The list is initialized with the initial problem to be solved, and in each iteration the best solution from the problem/solution pairs is found, which is then removed from the list. In the next iteration, the problem set contains the initial problem minus the previously found best solution, which will subsequently give the second-best solution. The method continues in this fashion until the M-best assignments are found, or the list of problems is empty.

Track Management

The JIPDA filter builds on the assumption that tracks already exists, and does not consider the problem of track formation (or termination). Note that the existence of tracks does not necessarily imply that the targets exist. In this thesis, a simple, yet effective method for track formation is employed. The track formation procedure was inspired by [81], with a modification with regard to when tracks are considered confirmed. The formation of preliminary tracks goes as follows: At each time step, measurements which are not associated with any tracks are stored to the next time step. At the next time step, for each measurement not associated with any target, the distance to every previously non-associated measurement is calculated. If this distance is below a threshold, a new preliminary track is formed with a set of predefined initial values.

Denote the set of unassociated measurements at the prevous time step as $\neg\mathbf{Z}_{k-1}$. The criterion for starting a new preliminary track can then be described as

$$|\neg\mathbf{Z}_k^p - \neg\mathbf{Z}_{k-1}^q| < Tv_{\text{max}} \qquad p = 1, \ldots, \varphi_k,\ q = 1, \ldots, \varphi_{k-1} \tag{4.36}$$

where $T = t_k - t_{k-1}$ is the system sample time, and v_{max} is the maximum velocity of the target. If a set of measurements satisfy (4.36), the measurements are used to generate the initial values of the new track. The initial existence probability is set to $\varepsilon_{\text{init}}^t = 0.5$, and the initial visibility probability is set to $\nu_{\text{init}}^t = 0.8$. To avoid the formation of many spurious tracks in cluttered situations, the fact that JIPDA calculates existence probabilities is exploited in track confirmation. A preliminary track is considered confirmed if $\varepsilon_k^t \geq 0.8$, and added to the list of confirmed tracks. In a similar fashion, tracks that have $\varepsilon_k^t \leq 0.01$ are considered dead, and removed from the list of confirmed (or preliminary) tracks.

Moreover, tracks within a pre-specified distance threshold d_T of one another are hypothesized to belong to the same target, and the track with the shortest lifetime is removed from the list of confirmed tracks.

Tracking Parameters

The JIPDA algorithm was run with the following set of parameters:

Parameter	Value
σ_{cv}^2	0.5
σ_l^2	0.5
P_G	0.99
P_D	0.90
$v_{\max}$	2.5
$\varepsilon_{\text{init}}^t$	0.5
ν_{init}^t	0.8
d_T	3 m
p_{11}^{ε}	0.99
p_{12}^{ε}	0.01
p_{21}^{ε}	0
p_{22}^{ε}	1
p_{11}^{η}	0.9
p_{11}^{η}	0.1
p_{11}^{η}	0.48
p_{11}^{η}	0.52
λ	0.01

Table 4.3: Parameters used in the tracking system.

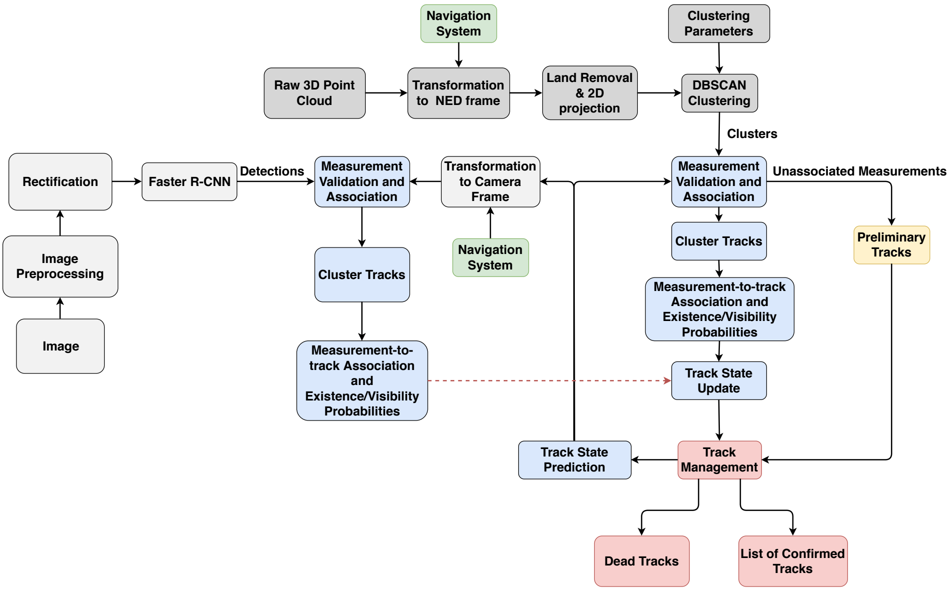

Navigation System
Clustering Parameters
Raw 3D Point Cloud
Transformation to NED frame
Land Removal & 2D projection
DBSCAN Clustering
Clusters
Rectification
Faster R-CNN
Detections
Measurement Validation and Association
Transformation to Camera Frame
Measurement Validation and Association
Unassociated Measurements
Image Preprocessing
Cluster Tracks
Navigation System
Cluster Tracks
Preliminary Tracks
Image
Measurement-to-track Association and Existence/Visibility Probabilities
Measurement-to-track Association and Existence/Visibility Probabilities
Track State Update
Track State Prediction
Track Management
Dead Tracks
List of Confirmed Tracks

Figure 4.21: The implemented tracking framework. Using camera measurements in the state update remains as future work. This is indicated by the dashed red line.

Chapter 5

Experimental Results and Discussion

5.1 Experimental Setup

In order to evaluate the proposed target tracking and sensor fusion framework, a series of experiments with the purpose of gathering real sensor data from different scenarios was performed. All experiments were performed in the Dora harbor basin in Trondheim, see figure 5.1. The satellite image was taken from Google maps.

Due to problems with the dynamic positioning system during the experiments, the ReVolt was moored to the harbor during the maneuvers. The mooring spot is indicated by a circle in figure 5.1. In the following, the ReVolt will be referred to as the *ownship*, while the targets will be referred to as *target 1* and *target 2*. Two boats were used as targets, one Polarcirkel 560 work with a length of 5.6 meters (target 1), and one Nidelv 690 sport (target 2) with a length of 7.14 meters. An overview of the experiment area with the two target boats can be seen in figure 5.2.

As a step in the data post-processing, all returns originating from land or other moored vessels in the area were manually removed from the lidar point cloud before they were fed into the tracking framework. In order to evaluate the performance of the tracking system, a position ground-truth (GT) for each target is helpful. This was generated by a mobile phone equipped with a GNSS receiver on each target, saving the position once per second. The mobile phones do not have differential GNSS capabilities, so there is significant uncertainty in the position data, however it will serve as an indication of the performance of the tracker when visually comparing the GT tracks with the tracks output

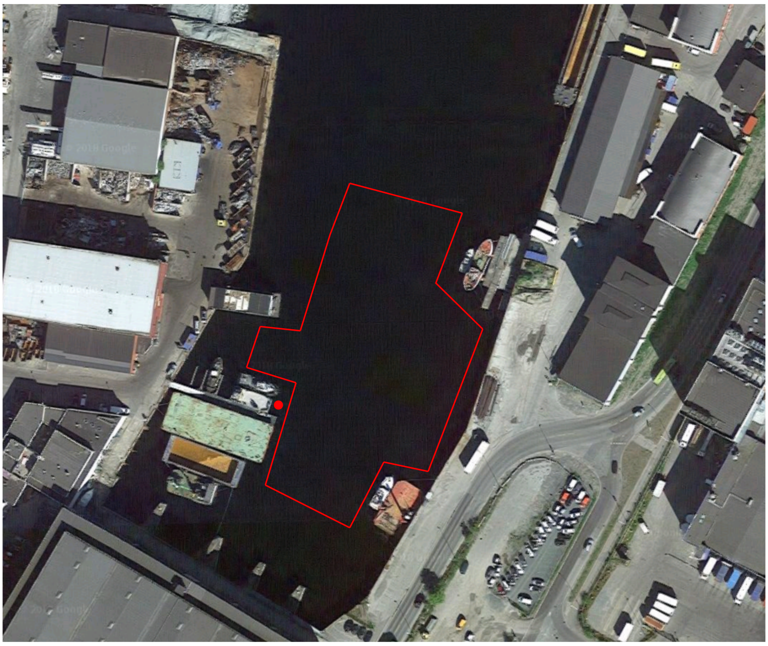


Figure 5.1: The experiment area in the Dora harbour basin. Maneuvers were performed inside the marked area.

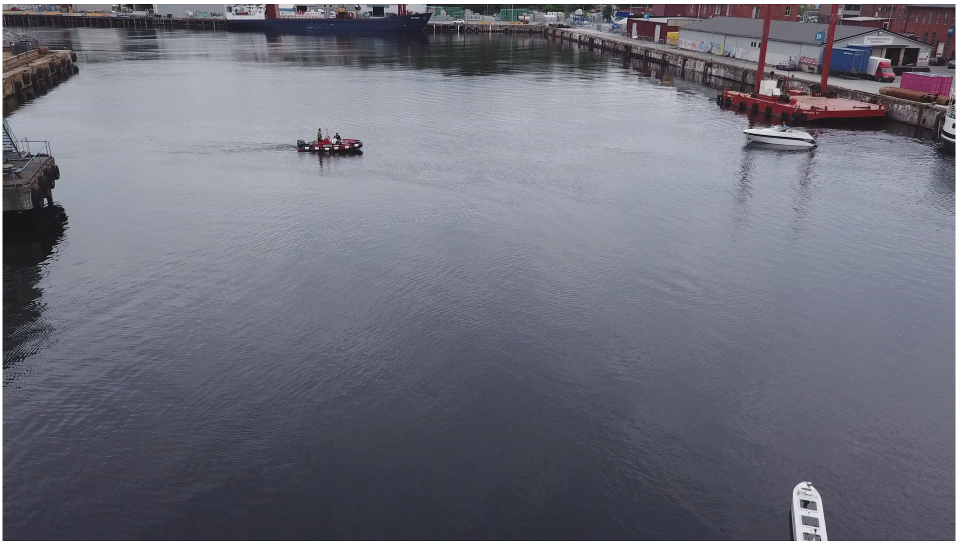
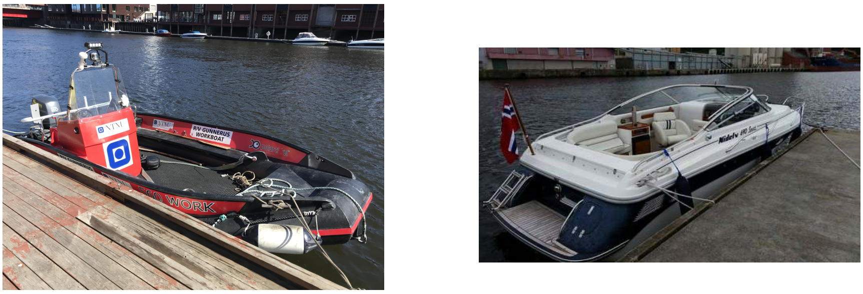


Figure 5.2: An overview of the experiment area. Target 1 can be seen on the left, and target 2 is the rightmost boat. Image courtesy of Tom Arne Pedersen, DNV GL.



(a) Polarcirkel 560 work.

(b) Nidelv 690 sport.

Figure 5.3: The two boats used as targets. The image of the Polarcirkel 560 work was supplied by Tom Arne Pedersen, DNV GL, while the image of the Nidelv 690 sport was supplied by Rune Green, DNV GL.

by the tracking system. There was calm seas and no precipitation during the experiments.

5.2 Tracker Evaluation Metrics

In order to evaluate the tracking system in a consistent and systematic way, performance measures must be defined. Different classes of performance metrics can be defined based on the availability of data. Gorji et al. [82] define two classes of metrics; sensor-related measures, and track-related measures, where the latter also takes the tracker into consideration. The track-related measures can further be divided into algorithm-based measures and algorithm-free measures. The algorithm-based measures are developed for individual types of trackers, whereas algorithm-free measures can be applied to every tracker. This makes the algorithm-free metrics attractive to use, since they can be used to compare the performance of different tracking algorithms in the future. Gorji et al. list a number of different metrics that can be used in the performance evaluation, while some rely on having an accurate ground truth and thus are mostly suited for simulations, the most relevant ones for our experiments will used in evaluating the tracking results. The definitions of each individual metric is taken from [82]. The consistency of the tracking system are also checked using statistical measures.

5.2.1 Cardinality Metrics

The cardinality metrics measure numerical characteristics of the obtained results. The cardinality metrics used in this thesis are listed in the following.

Number of Valid Tracks (NVT) A track is validated if it is associated with only one target, and also, the assigned target is not associated with any other track.

Number of Missed Targets (NMT) A target is missed if it is not associated with any track. The number of missed targets is incremented for every missed target at every time step through the whole duration of the scenario. This metric is similar to the *track loss rate* as defined in [83].

Number of False Tracks (NFT) A track is counted as a false one if it is not associated with any target. This value is incremented by the number of false tracks at every time step throughout the duration of the scenario.

Number of Broken Tracks (NBT) The number of broken tracks is found by checking each track associated with a target, and if the last time k of the track is smaller than the last appearance time of the target, the number of broken tracks is incremented by one.

Track Continuity (TC) The continuity for the l^{th} target is defined as

$$Tc_l = \frac{1}{N_l^t} \sum_{n=1}^{N_l^t} \frac{\Delta k_l^n}{\Delta k_l} \tag{5.1}$$

where N_l^t is the total number of tracks assigned to the target, Δk_l^n is the duration of the n^{th} track, and Δk_l corresponds to the appearance time of the l^{th} target.

5.2.2 Time Metrics

The time metrics provide information about the persistency of a track. Three time metrics are used in this thesis, listed in the following.

Rate of False Alarm (RFA) The rate of false alarm is defined as the number of false tracks per time step. Using the definition of the number of false tracks, this measure can be stated as

$$RFA = \frac{NFT}{\Delta t} \tag{5.2}$$

where Δt is the time duration of the scenario in question.

Track Probability of Detection (TPD) The track probability of detection is calculated by summing the number of time steps that a target is associated with a valid track, and dividing by the number of time steps that the target is present in the scenario. Mathematically, this can be expressed as

$$P_d^l = \frac{T_l}{\Delta k_l} \tag{5.3}$$

where P_d^l is the probability of detection for the l^{th} track, and T_l is the duration that the l^{th} target is assigned to a track. Δk_l is the appearance time of the l^{th} target. The final track probability of detection is then found by taking the average over all individual probabilities as

$$P_d = \frac{1}{L} \sum_{l=1}^{L} P_d^l \tag{5.4}$$

where L is the total number of targets.

Rate of Track Fragmentation (RFT) The rate of track fragmentation is found by examining the number of changes in the assigned track IDs for each target.

5.2.3 Statistical Filter Consistency Measures

A state estimator is *consistent* if the estimation error is unbiased, i.e. zero-mean, and that the actual mean square error of the filter matches the filter-calculated covariance. As a measure of filter consistency, the time-average *normalized innovation squared* (NIS) [84] can be used,

$$\bar{\epsilon}_\nu = \frac{1}{K}\sum_{k=1}^{K} \nu_k^T \mathbf{S}_k^{-1} \nu_k. \tag{5.5}$$

Under the hypothesis that the filter is consistent, $K\bar{\epsilon}$ has a χ^2-distribution with Kn_z degrees of freedom, where n_z is the dimension of the measurement vector. The hypothesis can be tested by using values from the χ^2 distribution table, and defining an acceptance region

$$\bar{\epsilon} \in [r_1, r_2] \tag{5.6}$$

where $[r_1, r_2]$ is the two-sided 95% probability region for the χ^2 distribution related to the corresponding NIS. Filter bias can be estimated by calculating the *Average Innovation* (AI) for all states [85], which should ideally be zero. The average innovation can be calculated by

$$\bar{\nu} = \frac{1}{K}\sum_{k=1}^{K} \nu_k. \tag{5.7}$$

5.3 Tracking Results

This section presents the maneuvers that was performed, as well as the tracking results using the lidar as the sensor. While the maneuvers performed are interesting from a strictly target tracking perspective, future applications, such as collision avoidance or extended object tracking was kept in mind when designing the scenarios. In the track plots, the start of a track is marked with a circle, and track termination is marked by a cross. Tracks associated with the targets are annotated with track ID as well as the duration of the track. The track-to-target association is done by hand in post-processing for performance analysis purposes.

5.3.1 Scenario 1

The first scenario involves a single target, target 1. In this scenario, the boat moved in a zig-zag pattern, performing a full stop at each turn. This scenario involves a maneuvering target, while the motion model used in the tracking assumes (nearly) constant velocity. The whole scenario ran over 1046 time steps. The target started close to the ownship, passing on a crossing course relative to the ownship heading. In the straight line segments the target had a velocity between 1.5 and 2 meters/second. Figure 5.4 shows the GNSS data

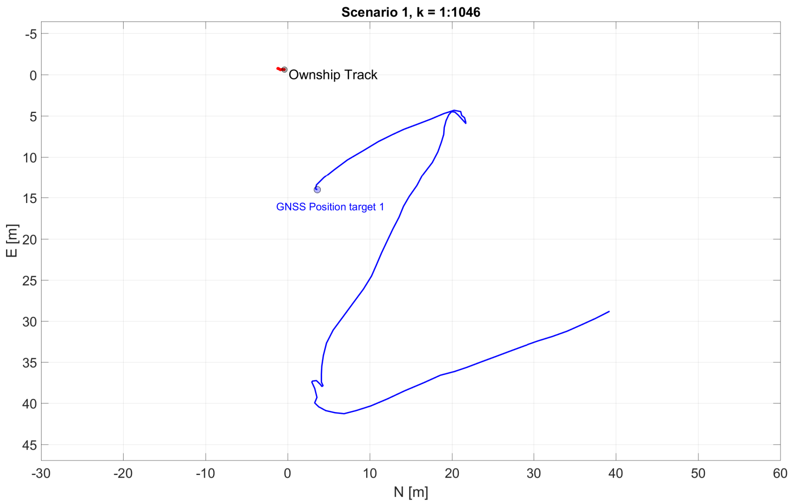


Figure 5.4: Scenario 1 GNSS track.

recorded for this run. Figure 5.5 shows the tracks output by the tracker for this scenario. The track labeled track 1 is the track from the target, starting at the second time step. As can be seen in the figure, there are several short tracks in the vicinity of the ownship. These tracks originated from waves generated by the target. Track 1 died at $k = 838$ due to a

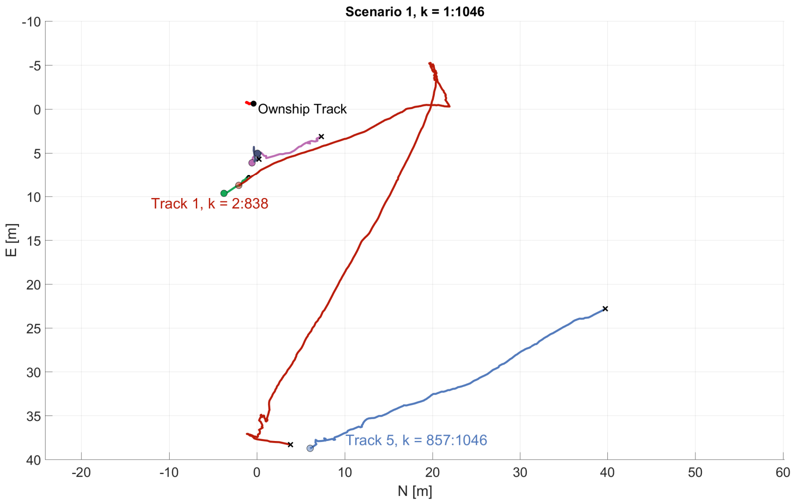


Figure 5.5: Tracks for scenario 1.

lack of detections from the target over 14 time steps, which lead to the track existence probability dropping below the track death threshold. The target became visible again shortly after, and was subsequently correctly tracked (track 5) until the end. The tracking system does not seem to be affected by the fact that the target maneuvers in this scenario, suggesting that the constant velocity model is robust to (non-aggressive) maneuvers. The number of false tracks in this scenario is high due to the many false wake-originated tracks at the start. This also leads to a high rate of false alarm. The number of missed targets is due to the missing detections between tracks 1 and 5.

Metric	**Value**
Number of valid tracks, target 1	2
Number of missed targets	20
Number of false tracks	103
Number of broken tracks, target 1	1
Track continuity, target 1	0.49
Rate of false alarm	0.0985
Track probability of detection, target 1	0.9799
Rate of track fragmentation, target 1	1

Table 5.1: Evaluation of tracking performance, scenario 1.

5.3.2 Scenario 2

In this scenario, the two targets started at opposite sides of the harbour basin, and moving across the basin passing one another in the middle. This scenario was performed in order to test the effect of occlusion, where the closer target blocks the farther one from the lidar's perspective, as well as testing how the tracker responds to close tracks at distance where the detections might be spurious. The larger target, target 2, passed on the far side from the ownship's point of view. Figure 5.6 shows the recorded GNSS tracks for the target in this scenario. Both targets had an approximate velocity of 1.6 meters/second through the scenario. The resulting tracks are shown in figure 5.7. There are no false tracks recorded in this scenario, however the tracks for both targets are fragmented. The gap between track 2 and track 3 is due to target 2 being detected for a short period, giving birth to track 2, before the target remained undetected for 11 time steps, causing the track to die. It was subsequently detected again at $k = 48$, and was successfully tracked for the remainder of the experiment. A similar situation caused the gap between tracks 1 and 4, where the smaller target 1 was spuriously detected at this distance causing the existence probability for track 1 to drop below the death threshold before the target was aquired again. Due to the relative size difference between the targets, target 2 was only partially occluded by target 1. This is illustrated in figure 5.8, where the partial occlusion lead to two clusters from target 2.

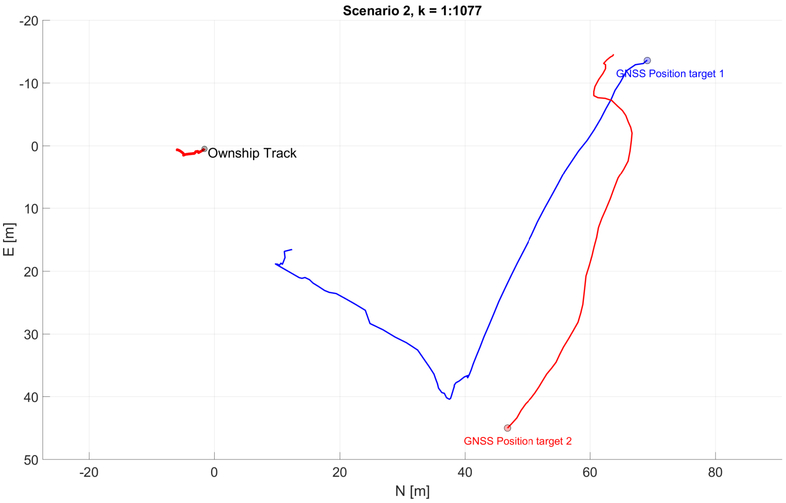


Figure 5.6: Scenario 2 GNSS track.

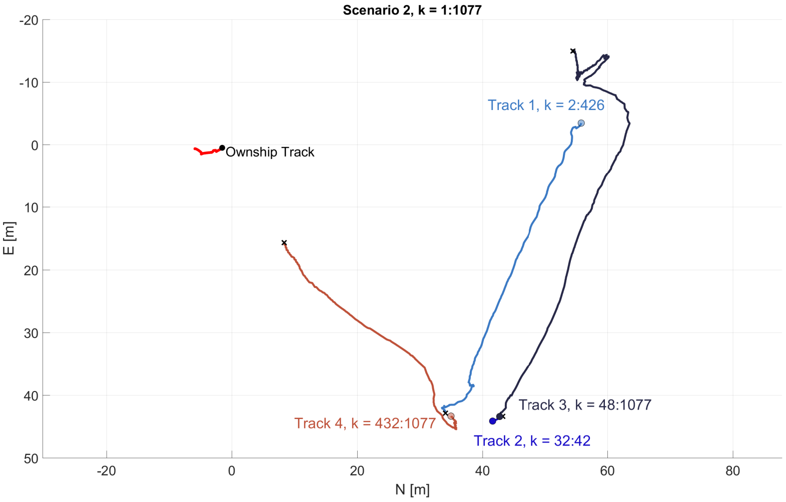


Figure 5.7: Tracks for scenario 2.

The number of missed targets in this scenario is, similarly to scenario 1, due to missed detections of the targets at a range of approximately 55 meters for target 1, and approximately 60 meters for target 2. The number of false tracks, and thus the number of false alarms is zero in this scenario, as no false tracks were observed. The tracks of both targets are fragmented, due to the aforementioned missed detections.

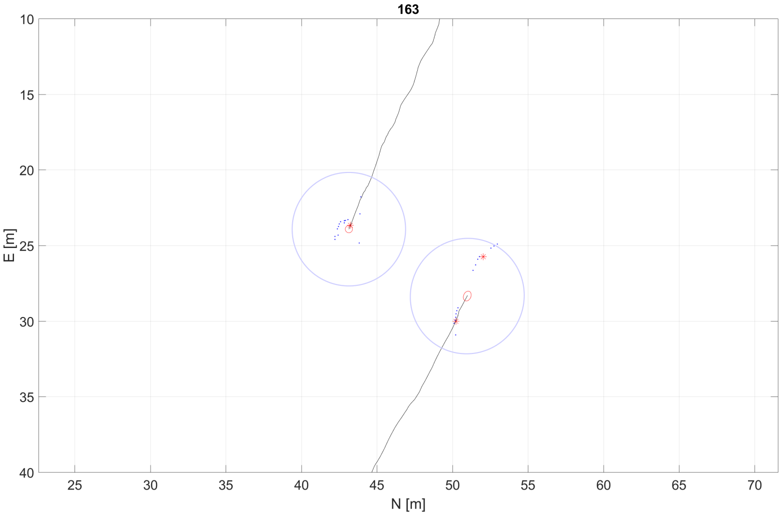


Figure 5.8: Target 2 partially occluded by target 1. The light blue ellipse represents the validation gate for each target, while the red ellipse represents the state covariance. The red asterisks are the cluster centroids.

Metric	**Value**
Number of valid tracks, target 1	2
Number of valid tracks, target 2	2
Number of missed targets	44
Number of false tracks	0
Number of broken tracks, target 1	1
Number of broken tracks, target 2	1
Track continuity, target 1	0.4963
Track continuity, target 2	0.4824
Rate of false alarm	0.0
Track probability of detection, target 1	0.9926
Track probability of detection, target 2	0.9647
Total track probability of detection	0.9787
Rate of track fragmentation, target 1	1
Rate of track fragmentation, target 2	1

Table 5.2: Evaluation of tracking performance, scenario 2.

5.3.3 Scenario 3

In scenario 3, the two targets started at a distance on a collision path towards one another. Both targets veered off before collision, going separate ways. The main purpose with this scenario is to test the tracking systems robustness with regard to track coalescence, as well as this being an interesting scenario from a collision avoidance standpoint for future projects. The GNSS track for each target is shown in figure 5.9. Target 1 had an approximate average velocity of 1.8 meters/second, and target 2 had an average velocity of 1.7 meters/second throughout the scenario. Again we see track fragmentation at the start, this time for target 1, due to spurious detections at distance. Target 2 was detected and subsequently tracked from $k = 93$. The short track labeled 4 in figure 5.10 was due to a wave in the wake of target 2, which died when the track came within the minimum track distance, keeping track 3 due to the longer lifetime of this track. This scenario also displays a fairly high number of missed targets. Both targets started at a distance on the limits of the range of the lidar, leading to delayed track aquisition time. A single false track was observed over 15 time steps, giving the number of false tracks in this scenario. Due to a delayed track aquisition time, both targets display a lower track probability of detection.

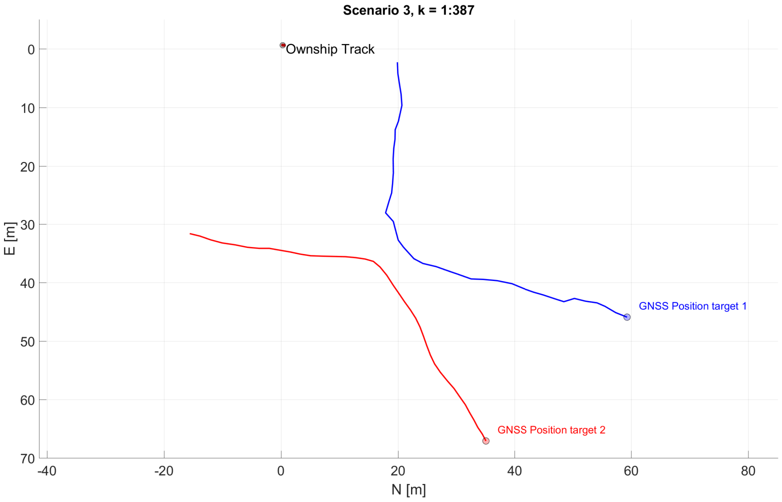


Figure 5.9: Scenario 3 GNSS track.

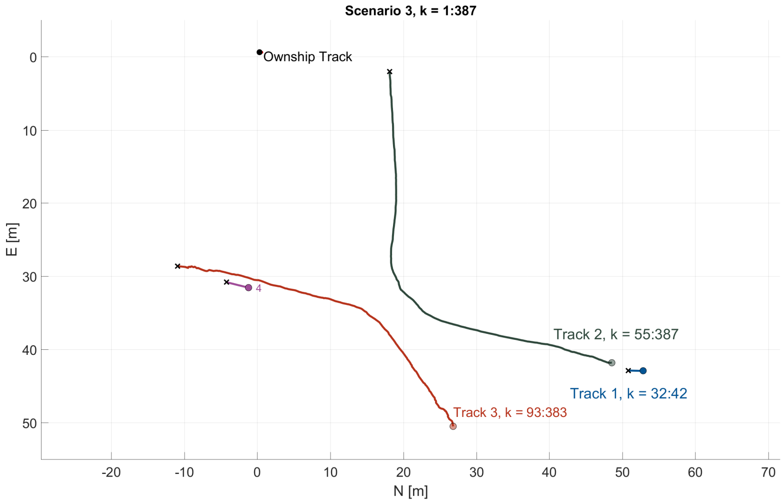


Figure 5.10: Tracks for scenario 3.

Metric	Value
Number of valid tracks, target 1	2
Number of valid tracks, target 2	1
Number of missed targets	140
Number of false tracks	15
Number of broken tracks, target 1	1
Number of broken tracks, target 2	0
Track continuity, target 1	0.4419
Track continuity, target 2	0.7494
Rate of false alarm	0.0388
Track probability of detection, target 1	0.8837
Track probability of detection, target 2	0.7494
Total track probability of detection	0.8166
Rate of track fragmentation, target 1	1
Rate of track fragmentation, target 2	0

Table 5.3: Evaluation of tracking performance, scenario 3.

5.3.4 Scenario 4

In scenario 4, the two targets are moving on parallel paths crossing in front of the ownship. Target 1 then veers off its path, moving away from the ownship. In addition to examining the effects of occlusion, this scenario also tests how the tracking system responds to a sudden maneuver with the constant velocity model. Both targets had an approximate average velocity of 2 meters/second for this scenario. In this scenario the smaller target is on the far side of the ownship, causing it to be fully occluded for an extended period. This is evident in the gap between track 2 and track 3, as seen in figure 5.12. There is little spatial gap between the tracks, this is due to the cluster centroid being close to the bow of the target before full occlusion, and the aft was the first part of the target becoming visible again. The track fragmentation at the end is due to spurious visibility of the target at distance. Target 2 is successfully tracked through the maneuver, the broken track at the end is due to the target disappearing behind a building in the vicinity. No false tracks were observed in this scenario, but the number of missed targets is fairly high. This is due to the aforementioned occlusion of target 1, causing the target to be lost over several time steps, as well as track loss as the target moved outside the lidar range. This also led to a low track continuity for target 1.

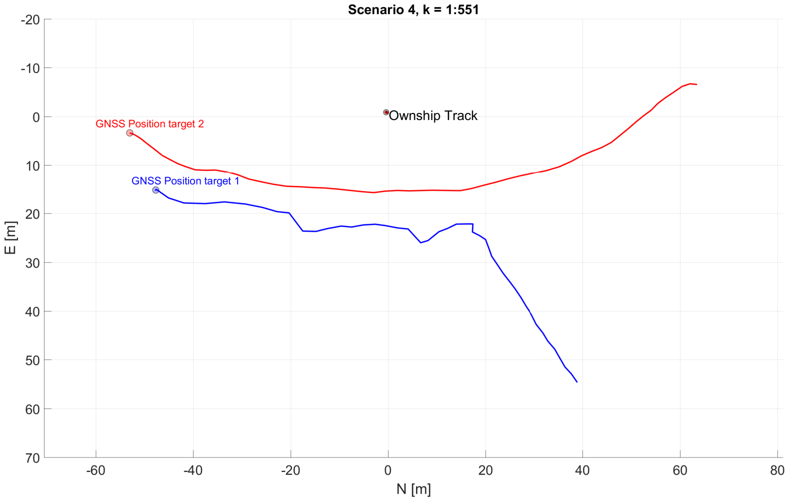


Figure 5.11: Scenario 4 GNSS track.

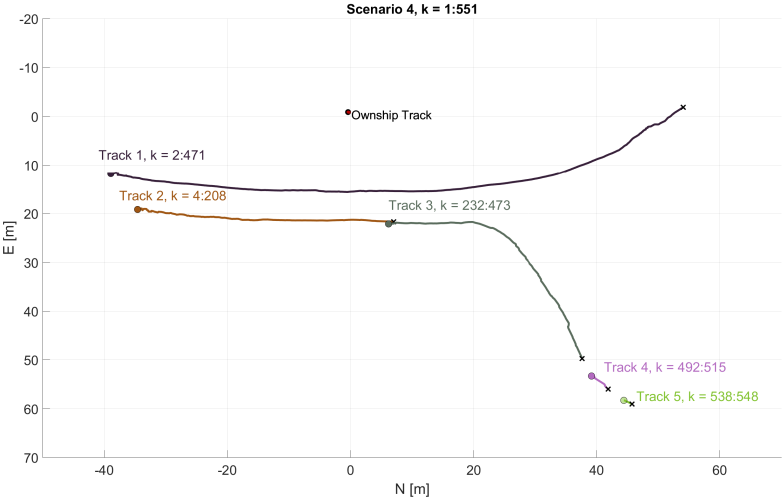


Figure 5.12: Tracks for scenario 4.

Metric	**Value**
Number of valid tracks, target 1	4
Number of valid tracks, target 2	1
Number of missed targets	153
Number of false tracks	0
Number of broken tracks, target 1	4
Number of broken tracks, target 2	1
Track continuity, target 1	0.2169
Track continuity, target 2	0.8512
Rate of false alarm	0.0
Track probability of detection, target 1	0.8675
Track probability of detection, target 2	0.8512
Total track probability of detection	0.8594
Rate of track fragmentation, target 1	3
Rate of track fragmentation, target 2	0

Table 5.4: Evaluation of tracking performance, scenario 4.

5.3.5 Scenario 5

In scenario 5, the two targets started close to the ownship, travelling on parallel paths away from the ownship. This scenario is designed to test the range limitations of the lidar as a tracking sensor, and in future applications how the camera can aid in maintaining tracks when the targets are outside the lidar range. The targets moved with an average velocity of 2.3 meters/second away from the ownship. As seen in figure 5.14, there is a lot of false tracks around the ownship. This is caused by wakes from the targets being detected by the lidar, which lead to several spourious, short-lived tracks as the targets passed. Figure 5.15 is a snapshot of the lidar point cloud at time step $k = 148$ illustrating the wakes behind the targets. Each red asterisk represent a cluster being sent to the tracker as a measurement, causing spourious tracks. It is also seen that target 1 is lost before target 2 in this scenario. This is not surprising, since target 1 has a lower profile in the sea than target 2, leading to fewer detections than from the larger target 2. Target 2 is lost at a range of approximately 85 meters away from the ownship. This scenario has both a very high number of missed targets, as well as a large number of false tracks. The high number of missed targets is due to the scenario running until the targets are well outside the lidar range. This also lead to a low total track probability of detection. This scenario also illustrates the lidars sensitivity to target extent, as the larger target 2 was tracked much longer than the smaller target 1. The high number of false tracks is again due to the wakes, causing a high rate of false alarm.

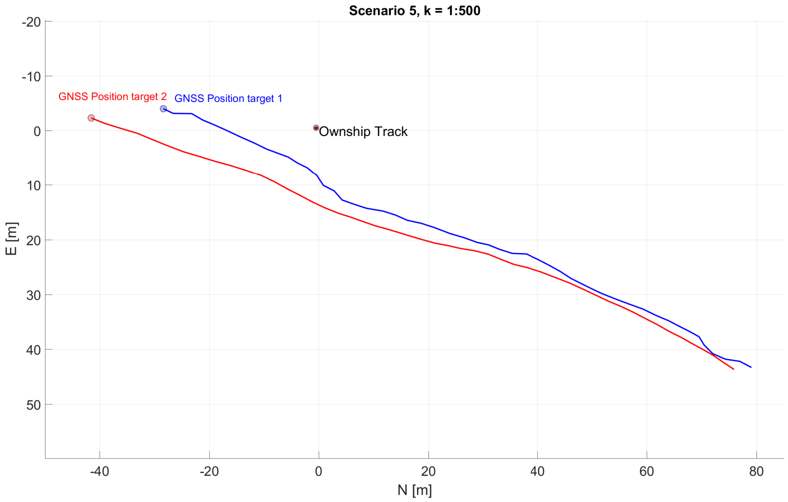


Figure 5.13: Scenario 5 GNSS track.

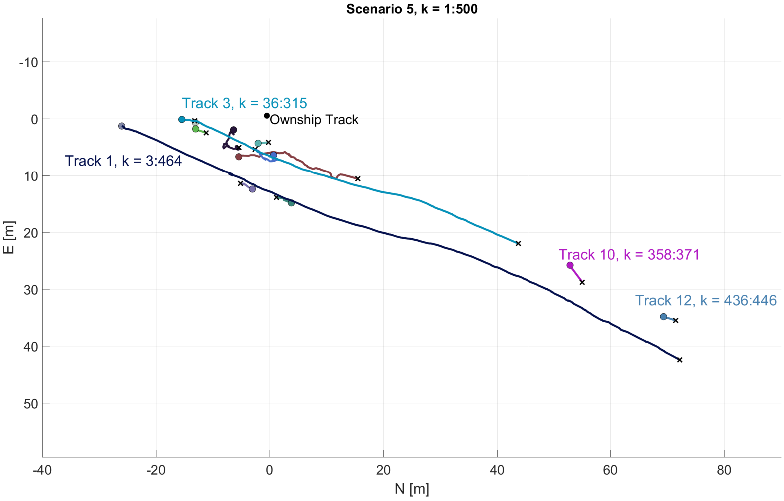


Figure 5.14: Tracks for scenario 5.

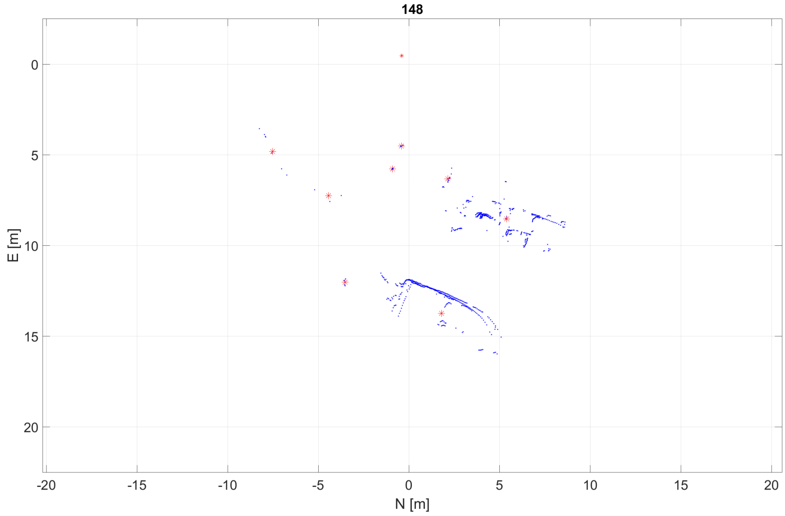


Figure 5.15: Wakes behind the targets in scenario 5.

Metric	**Value**
Number of valid tracks, target 1	2
Number of valid tracks, target 2	1
Number of missed targets	389
Number of false tracks	135
Number of broken tracks, target 1	1
Number of broken tracks, target 2	1
Track continuity, target 1	0.2379
Track continuity, target 2	0.3473
Rate of false alarm	0.2987
Track probability of detection, target 1	0.4757
Track probability of detection, target 2	0.6947
Total track probability of detection	0.5852
Rate of track fragmentation, target 1	1
Rate of track fragmentation, target 2	1

Table 5.5: Evaluation of tracking performance, scenario 5.

5.3.6 Scenario 6

In scenario 6 the targets approach the ownship from a distance, approaching on parallel paths. Scenario 6 is the second half of scenario 5, where the two targets return along the same path as they left. The targets approached with an average velocity of approximately 2.4 meters/second. The recorded GNSS path for the scenario is illustrated in figure 5.16, and the tracks output by the tracking system is shown in figure 5.17.

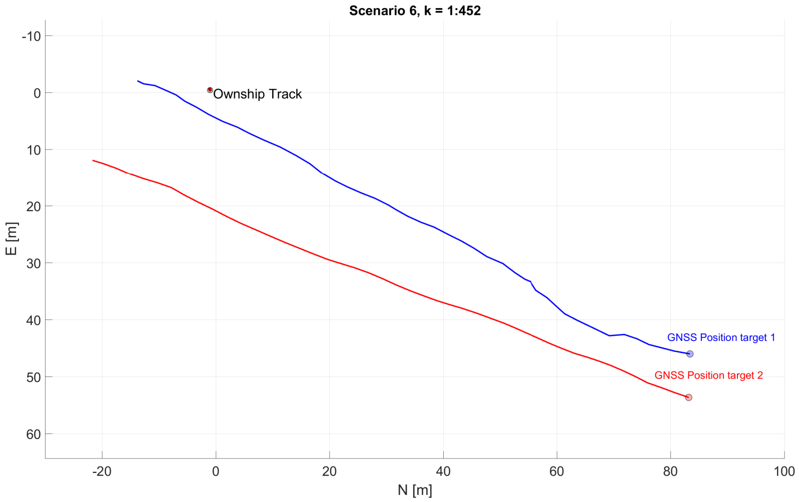


Figure 5.16: Scenario 6 GNSS track.

Again it can be observed that target 2 is detected and tracked at a greater distance than target 1. There is also a large amount of false tracks around the ownship in this scenario, resulting from the wakes generated by the two targets passing the ownship. The track labeled track 6 actually originated from wakes from target 2, and track 1 was lost due to occlusion by target 1. Track 6 subsequently started tracking the target afterwards. This scenario is similar to scenario 5, with both a high number of missed targets and a high number of false tracks. The rate of false alarm is again high due to the false tracks.

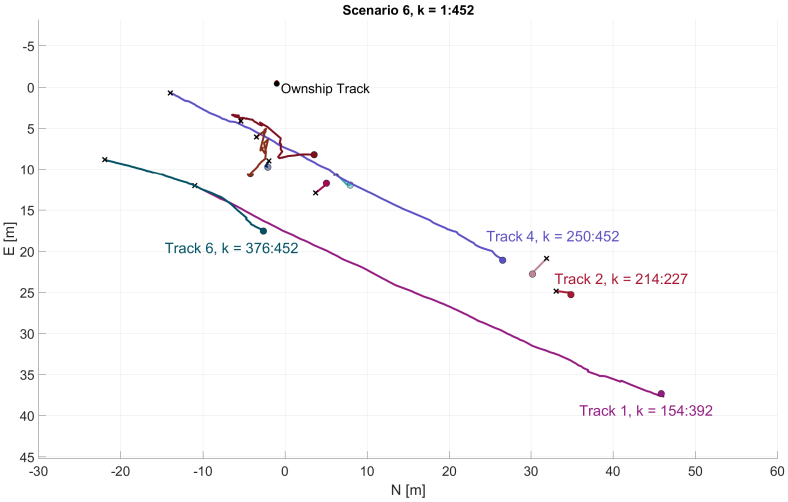


Figure 5.17: Tracks for scenario 6.

Metric	Value
Number of valid tracks, target 1	3
Number of valid tracks, target 2	1
Number of missed targets	235
Number of false tracks	242
Number of broken tracks, target 1	3
Number of broken tracks, target 2	1
Track continuity, target 1	0.2013
Track continuity, target 2	0.9220
Rate of false alarm	0.4840
Track probability of detection, target 1	0.6040
Track probability of detection, target 2	0.9220
Total track probability of detection	0.7630
Rate of track fragmentation, target 1	2
Rate of track fragmentation, target 2	0

Table 5.6: Evaluation of tracking performance, scenario 6.

5.3.7 NIS and Average Innovation

Scenario 5 was used in the calculation of the NIS and the average innovation (AI) to evaluate the covariance consistency of the filter. This scenario was used since it tracks the targets all the way from up close until they are out of range. The NIS and AI were calculated for a variety of process noise values σ_{cv}^2 over 200 time steps, and the results are shown in table 5.7 and table 5.8. The 95% probability region for the NIS is found from tables of the χ^2-distribution to be $[r_1, r_2] = [1.7324, 2.2865]$, and the NIS values closest to this region is emphasized in bolt for both targets. The process noise variance in the tracking framework was selected to be $\sigma_{cv}^2 = 0.5$ based on these results, and is a compromise between the values closest to being covariance-consistent for both targets. Notice that the AI for target 1 has a slight negative bias, this could be due to wake measurements falling within the validation gate of the target early on in the scenario.

Target	$\sigma_{cv}^2 = 0.05$		$\sigma_{cv}^2 = 0.1$	
	NIS	AI	NIS	AI
Target 1	4.11	$(-0.02, -0.28)^T$	3.28	$(-0.05, -0.23)^T$
Target 2	**1.94**	$(0.04, -0.09)^T$	1.92	$(0.00, -0.08)^T$

Table 5.7: NIS and AI for target 1 and target 2 in scenario 5.

Target	$\sigma_{cv}^2 = 0.5$		$\sigma_{cv}^2 = 1.0$	
	NIS	AI	NIS	AI
Target 1	2.82	$(-0.23, -0.20)^T$	**2.79**	$(-0.24, -0.18)^T$
Target 2	1.85	$(0.02, -0.07)^T$	1.61	$(0.00, -0.08)^T$

Table 5.8: NIS and AI for target 1 and target 2 in scenario 5, continued.

5.4 The Camera as a Complimentary Sensor

As described in section 4.7, the predicted measurements for each target are transformed into the camera frame using (4.33), and converted into an line-of-sight angle referenced to the z-axis in the camera frame. The center of the bounding boxes given by Faster R-CNN are also converted to a line-of-sight angle in the camera frame, and the measurement-to-track association probabilities are subsequently calculated by the JIPDA filter. The field of view of the camera is 125°.

For the associations to make sense, the transformations between the coordinate frames have to be accurate, which again relies on the accuracy of the calibration between the sensors, as well as the camera calibration. Moreover, the measurements from each sensor must be synchronized in time. Images were captured with a sample frequency of 10 Hz. In figure 5.18, a single image recorded from scenario 2 is displayed, with the track locations given by the tracking framework marked as yellow pluses.

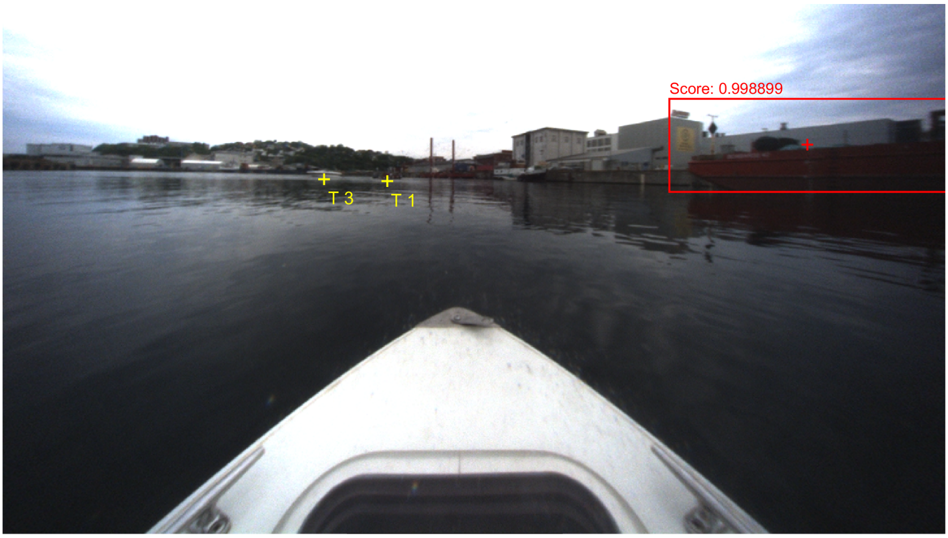


Figure 5.18: Image from scenario 2. The range to the track labeled T 3 is approximately 60 meters, and the range to the track labeled T 1 is approximately 50 meters, both measured by the lidar.

Figure 5.19 is a cropped version of figure 5.18, which shows that the predicted track location is aligned with the center of each target, giving a visual indication that the calibration procedure (and the transformations between the coordinate frames) is accurate.

The Faster R-CNN model was trained as a single-class detector to detect boats in the images, and it was found that most of the detections originated from the surroundings, such as from houses or structures in the vicinity, rather than from the actual targets. The detections from scenario 5 was examined to get an estimate of the practical range of the camera as a detector using the implemented Faster R-CNN model. The results are displayed in table 5.9. The range estimate to the targets was based on the measurements from the lidar. As is evident from table 5.9, none of the detections originated from any of the targets at ranges over 20 meters. The other scenarios have similar detection results, except for some detections when the broadside of the targets are facing the camera in scenario 2. Moreover, at close ranges, most of the detections originated from the surroundings rather than from the targets. This is not necessarily a problem, as seen in figure 5.20, the association probabilities calculated by JIPDA (displayed as Beta: T 1:0.873 in the image) only associates the track with the actual target detection. Since a lot of the misdetections originate from structures such as houses in the background, adding a few ”background” classes such as *house*, *building*, *car* and so on to the image classifier could reduce such misdetections. Figures 5.20 through 5.23 shows some of the cases where good detections

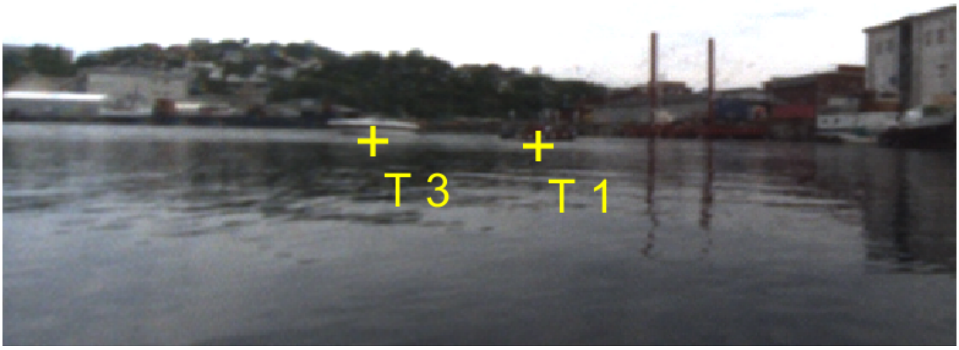


Figure 5.19: Cropped version of 5.18, showing the targets and their position given by the tracking system projected into the image.

Range	**Target detections**	**False detections**	**Images with no target detections**	**Images**
$\leq$ 10 meters	44	62	2	43
10-20 meters	20	60	20	40
20-30 meters	0	93	40	40
30-40 meters	0	140	45	45

Table 5.9: Evaluation of image detections in scenario 5.

of the targets was achieved, showing that the computation of the association probabilities could be helpful in discriminating between false and true tracks. The measurement noise influencing the gating distance has not been optimized in these examples, neither has the clutter intensity λ. Since the Faster R-CNN detector is a direct reimplementation of the work done by Tangstad in his thesis [63], and has not been the main focus of this thesis, improving the visual detection of boats remains as future work.

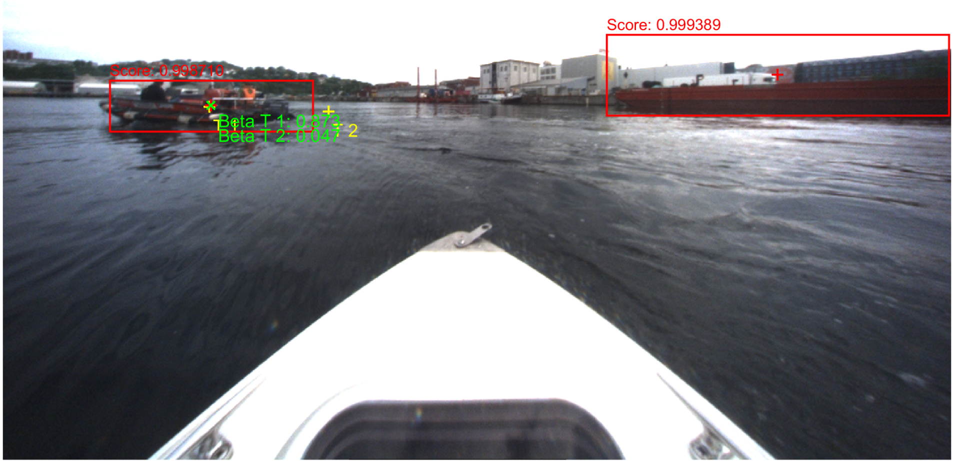


Figure 5.20: Computed association probabilities for image detection, scenario 1.

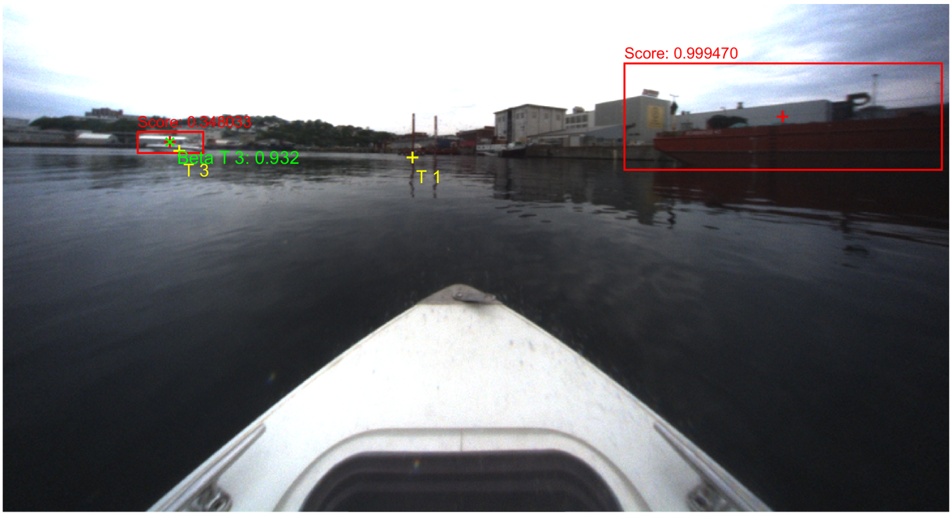


Figure 5.21: Computed association probabilities for image detection, scenario 2.

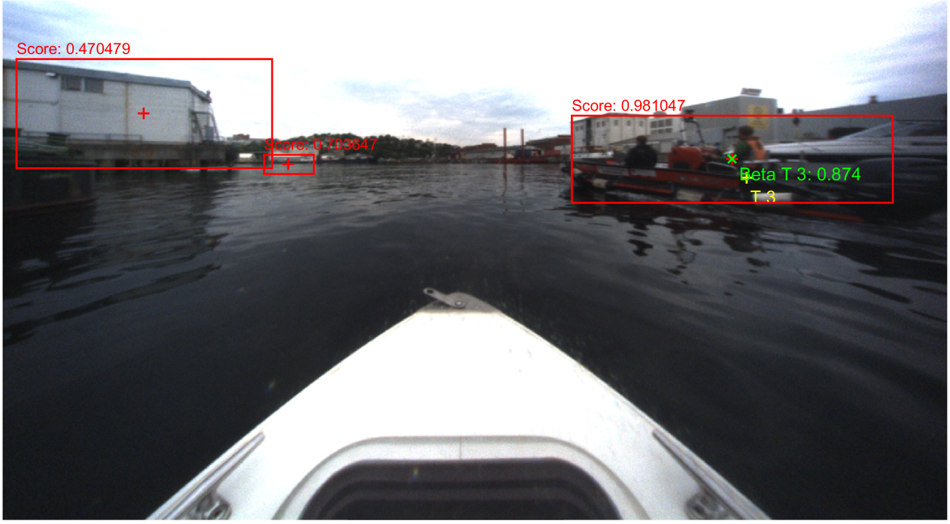


Figure 5.22: Computed association probabilities for image detection, scenario 5.

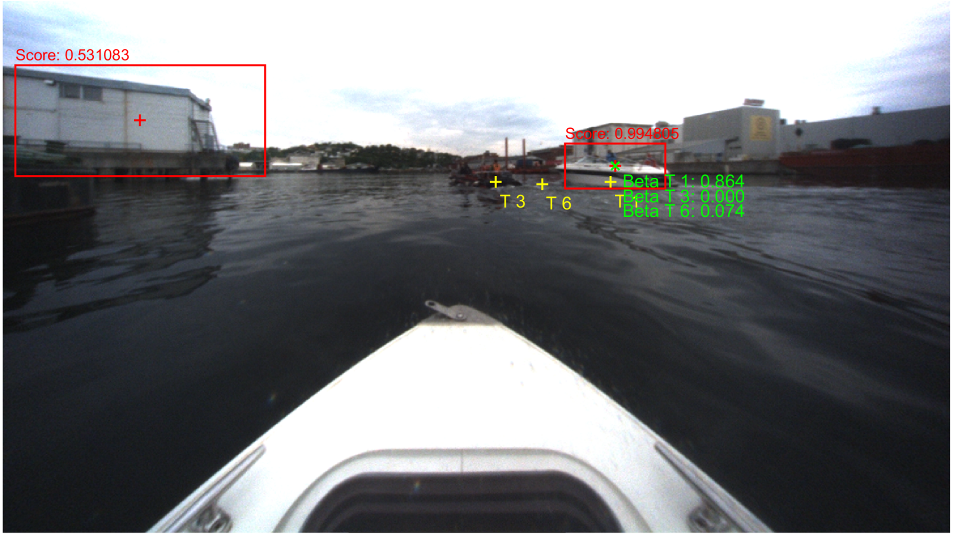


Figure 5.23: Computed association probabilities for image detection, scenario 5.

5.5 Discussion of the Results

This section discusses the results found in the previous sections, and assesses the overall performance of the tracking system.

5.5.1 The Wake Problem

Wakes behind the targets when they were close led to many false tracks output by the tracking system. This is evident from the number of false tracks in scenarios 1, 5 and 6 (103, 135 and 242, respectively). Other than when the targets were close, very little clutter was observed in the data. This suggests that the parametric Poisson clutter model might not be the best suited clutter model for the tracking system with the lidar as the sensor. The Poisson clutter model assumes that the clutter is independent of the target, which does not seem to be the case for boats at sea. Brekke et al. [86] developed a way to model the wakes as a probability density function behind the target for the nonparametric PDAF, an extension of the JIPDA filter to include such a model could aid in the tracking when wakes are present. The detections from the camera could also be used to discriminate between wakes and targets, by using the association probabilities between detections and tracks. Figure 5.20 shows an example where the detection of the target has a higher association probability with the actual target track (T 1) than the track originating from wake (T 2). Even though the wakes led to many false tracks, the tracks of the actual targets were never lost in these cluttered scenarios.

5.5.2 Point Targets versus Extended Targets

The detection model for the lidar sensor is a very simple one, where points that are close enough with a specified spatial density are clustered together and presented to the tracking system as a measurement. When the targets are close to the ownship, the point cloud is rich with information about the extent and shape of the target(s) (see e.g. figure 5.15), which could be utilized to discriminate the clusters between actual targets and wakes with methods like feature extraction, using e.g. spin images [87], or other geometrical features [88], and subsequently classifying the cluster as clutter or target. Other methods include utilizing extended object tracking methods such as using Kalman filter approaches, e.g. [89], or random matrix approaches, e.g. Koch [90], Feldman et al. [91] and Granström et al. [92]. Extended object tracking methods are attractive due to the fact that they estimate the extent of a target in addition to its state, which is valuable information for close-range maneuvering.

5.5.3 The Range Limitations of the Lidar

The results also show that the detection probability of smaller targets, such as leisure crafts used in the experiments, depends on the range from the lidar. The smaller of the two targets used in the experiments had a low probability of detection at ranges beyond 50 meters. At 50 meters the laser beams from the lidar has a vertical spread of $h = r \sin \omega_i \approx 1.75$ meters, and a smaller target could easily come between the vertical spread of the beams. The larger target, target 2, was detected at longer ranges, as seen in scenario 5. In this case, the track was maintained successfully up to a range of approximately 80 meters. The lidar manufacturer states that each individual laser beam has a effective range of approximately 100 meters. A lidar sensor with a higher vertical angular resolution could improve the effective range. As the aim is to track and avoid targets in urban environments, the detection of small targets such as kayakers or even swimmers is important from a safety standpoint, and could be difficult at range using a lidar with low vertical angular resolution.

5.5.4 Modeling Occlusions

As seen in scenario 4, occlusion of a target over several time steps may lead to track loss for the occluded target. Granström et al. [93] model occlusions by modelling the probability of detection as non-homogeneous. The basic idea is to lower the probability of detection if a point is located behind a target estimate from the sensors point of view. Such an approach depends on knowledge about the extent of the target blocking the other points, and is thus most suited for extended target tracking methods, unless the extent of targets is known through some other means.

5.5.5 Overall Tracking Results

The main limitation with regard to tracking the actual targets seems to be the range between the lidar and the target, as well as the extent of the target itself, as described in section 5.5.3. Even in situations where wakes give birth to many false tracks, the targets are successfully tracked while they are within range of detection. However, the birth of many false tracks, especially close to the ownship, is undesirable in a situation where the tracking system is used in making sense-and-avoid and navigation decisions for an autonomous vessel. Methods to mitigate the birth of false tracks from wakes using the camera is a possibility to investigate further. The motion model used in the tracking framework assumes near constant velocity, however this has not appeared to be a limitation when tracking maneuvering targets.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

A ROS-based software achitecture for data reception from the camera and the lidar has been implemented on the ReVolt model ship. The lidar and camera have been calibrated, giving an accurate transformation between the two coordinate systems, as well as transformation to a common world frame. Measurement models for both sensors have been formulated, and a modified version of the DBSCAN algorithm has been utilized to cluster the lidar data to satisfy the point-target-assumption in the tracking system. A series of tracking scenarios using real targets at sea has been planned, organized and executed, giving a comprehensive data set which may be valuable also in future research.

The target tracking system has been built upon an existing JIPDA implementation, and targets are tracked using the lidar as the primary sensor. The tracking results show that the lidar is sensitive to wakes from the targets at close range, leading to many false tracks output by the tracking system. The targets are not lost due to wakes. The target probability of detection with the lidar is reduced at range, depending on the extent of the target in question. At ranges between 10-50 meters, the targets are mostly successfully tracked with few false tracks. Some track fragmentation is observed, mostly at ranges where the target detection probability is low, leading to track deaths and births, as well as some cases where the smaller target is fully occluded by the larger target. The clustering of lidar data using DBSCAN performed satisfactorily, no noticeable over- or undersegmentation of targets was observed. The dynamic radius function used in stead of the normal constant radius aided in separating the wakes from the targets at close ranges, while still being able to cluster the more spread-out points at range.

The Faster R-CNN detector showed that it has a very limited range with the camera and lens used, where the detections were few and far between at ranges over 20 meters. The model used is a direct reimplementation of a previously trained model [63], and has not been the main focus in this thesis. At closer ranges, where the targets are steadily detected, the measurement-to-track association probabilities computed by JIPDA show potential to be used in the tracking framework. At close ranges wakes in the lidar data is a challenge, however this is not the case for the camera. The camera could be used at close range to discriminate between target-originated tracks and wake-originated tracks, as well as aiding in the track formation and confirmation. This remains as future work. Moreover, the camera used is a straight-ahead-looking single camera giving a reduced field of view, 360° coverage would give a better situational overview.

6.2 Suggestions for Future Work

This thesis cover a broad specter of research areas as well as much practical work, and it has not been possible to go into great depth in all topics covered. The thesis work has laid the basic foundations of a complete situational awareness system, and the data set produced opens up many possible avenues of further research. Some suggestions for possible future research topics are listed below.

- Implement the proposed target tracking framework in ROS on ReVolt, and examine its real-time capabilities.
- Examine the possibility of including more classes in the image detection framework, to reduce the amount of false detections from background objects.
- Perform experiments to determine the measurement noise covariance matrix $\mathbf{R}_l$ for the lidar. The method used in this thesis is based on an assumption on the size of the targets, a more rigorous way of determining the measurement noise covariance could be considered. A possible method is given in [85].
- Information about the extent of targets is useful at close range, for collision avoidance and path planning purposes, as well as occlusion modelling. Extended object tracking methods using the lidar data could give such information. Possible approaches include Kalman filter approaches, e.g. [89], or random matrix approaches, e.g. Koch [90], Feldman et al. [91] and Granström et al. [92]. A recent master thesis by Kristian Ruud at NTNU has also showed promising results by combining the Gaussian mixture extended Kalman filter (GMEKF) with the general probabilistic data association filter (GPDA) [94]. This method was tested on real data gathered with the lidar used in this thesis, and the results show that it outperformed the random matrix approach.
- Investigate the possibility of using detections from the camera to mitigate false tracks due to wakes, as well as aiding in track formation and confirmation. This could for instance be based on the measurement-to-target association probabilities computed by JIPDA.

- In this thesis, lidar measurements originating from static objects in the vicinity was manually removed in post-processing. A way to filter out static surroundings automatically is needed in a real-time application.

Bibliography

[1] V. Kamsvåg, “Fusion of lidar and camera for collision avoidance purposes”, 2017.

[2] E Brekke, *Fundamentals of Sensor Fusion: Target tracking, navigation and SLAM*. Teaching material for future course in sensor fusion, NTNU, 2018.

[3] E. Jokioinen, J. Poikonen, M. Hyvönen, A. Kolu, T. Jokela, J. Tissari, A. Paasio, H. Ringbom, F. Collin, M. Viljanen, R. Jalonen, R. Tuominen, and M. Wahlström, *Remote and autonomous ships - the next steps*, 2016. [Online]. Available: `http://www.rolls-royce.com/~/media/Files/R/Rolls-Royce/documents/customers/marine/ship-intel/aawa-whitepaper-210616.pdf`.

[4] W. Koch, *Tracking and Sensor Data Fusion: Methodological Framework and Selected Applications*. Springer, 2014.

[5] H. Alfheim and K. Muggerud, “Development of a dynamic positioning system for the revolt model ship”, Master Thesis, Norwegian University of Science and Technology, 2017.

[6] C. Stiller, J. Hipp, C. Rössig, and A. Ewald, “Multisensor obstacle detection and tracking”, *Image and Vision Computing*, vol. 18, no. 5, pp. 389 –396, 2000.

[7] M. Mahlisch, R. Schweiger, W. Ritter, and K. Dietmayer, “Sensorfusion using spatio-temporal aligned video and lidar for improved vehicle detection”, in *2006 IEEE Intelligent Vehicles Symposium*, 2006, pp. 424–429.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning*, 1st ed. 2006. Corr. 2nd printing, ser. Information science and statistics. Springer, 2006.

[9] R. Szeliski, *Computer Vision, Algorithms and Applications*. Springer, 2011.

[10] R. Aufrère, J. Gowdy, C. Mertz, C. Thorpe, C. Wang, and T. Yata, “Perception for collision avoidance and autonomous driving”, *Mechatronics*, vol. 13, no. 5, pp. 1149 –1161, 2003.

[11] H. Cho, Y. W. Seo, B. V.K. V. Kumar, and R. R. Rajkumar, “A multi-sensor fusion system for moving object detection and tracking in urban driving environments”, in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 1836–1843.

[12] C. Premebida, O. Ludwig, and U. Nunes, “Lidar and vision-based pedestrian detection system”, *Journal of Field Robotics*, vol. 26, no. 9, pp. 696–711, 2009.

[13] H. Weigel, P. Lindner, and G. Wanielik, “Vehicle tracking with lane assignment by camera and lidar sensor fusion”, in *2009 IEEE Intelligent Vehicles Symposium*, Jun. 2009, pp. 513–520.

[14] M. T. Wolf, C. Assad, Y. Kuwata, A. Howard, H. Aghazarian, D. Zhu, T. Lu, A. Trebi-Ollennu, and T. Huntsberger, “360-degree visual detection and target tracking on an autonomous surface vehicle”, *Journal of Field Robotics*, vol. 27, no. 6, pp. 819–833, 2010.

[15] L. Elkins, D. Sellers, and W. R. Monach, “The autonomous maritime navigation (amn) project: Field tests, autonomous and cooperative behaviors, data fusion, sensors, and vehicles”, *Journal of Field Robotics*, vol. 27, no. 6, pp. 790–818, 2010.

[16] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. New York, NY, USA: Cambridge University Press, 2003.

[17] Z. Zhang, “A flexible new technique for camera calibration”, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, Dec. 2000.

[18] O. Egeland and J. T. Gravdahl, *Modeling and Simulation for Automatic Control*. Marine Cybernetics, 2002.

[19] J. Wittenburg, *Kinematics: Theory and Applications*. Springer, 2016.

[20] D. S. Bernstein, *Geometry, Kinematics, Statics and Dynamics*. Princeton University Press, 2012.

[21] J. Heikkila and O. Silven, “A four-step camera calibration procedure with implicit image correction”, in *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Jun. 1997, pp. 1106–1112.

[22] *Matlab version 9.2.0.538062 (r2017a)*, The Mathworks, Inc., Natick, Massachusetts, 2017.

[23] G. Bradski, “The OpenCV Library”, *Dr. Dobb’s Journal of Software Tools*, 2000.

[24] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning”, *Nature*, vol. 521, 436 EP –, May 2015.

[25] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge”, *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[26] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, “Backpropagation applied to handwritten zip code recognition”, *Neural Computation*, vol. 1, no. 4, pp. 541–551, Jan. 1989.

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[28] M. D Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks", *ArXiv e-prints*, Nov. 2013. arXiv: `1311.2901 [cs.CV]`.

[29] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", *ArXiv e-prints*, Jul. 2012. arXiv: `1207.0580`.

[30] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *ArXiv e-prints*, Feb. 2015. arXiv: `1502.03167 [cs.LG]`.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", *ArXiv e-prints*, Dec. 2015. arXiv: `1512.03385 [cs.CV]`.

[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[33] P. McManamon, *Field Guide to Lidar*. Society of Photo-Optical Instrumentation Engineers (SPIE), 2015.

[34] *VLP-16 User Manual*, English, version 63-9243 Rev. D, Velodyne LiDAR, Inc., 138 pp., 2018.

[35] C. L. Glennie, A. Kusari, and A. Facchin, "Calibration and Stability Analysis of the VLP-16 Laser Scanner", *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, pp. 55–60, Mar. 2016.

[36] T. D. Barfoot, *State Estimation for Robotics*. Cambridge University Press, 2017.

[37] Y Bar-Shalom and X.-R. Li, *Multitarget-multisensor Tracking: Principles and Techniques*. YBS Publishing, 1995.

[38] M. Schreier, *Bayesian Environment Representation, Prediction, and Criticality Assessment for Driver Assistance Systems*. Darmstadt, 2016.

[39] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2005.

[40] S. Challa, M. R. Morelande, D. Musicki, and R. J. Evans, *Fundamentals of Object Tracking*. Cambridge University Press, 2011.

[41] X. R. Li and V. P. Jilkov, "Survey of maneuvering target tracking. Part I. Dynamic models", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 39, no. 4, pp. 1333–1364, Oct. 2003.

[42] R. E. Kalman, "A new approach to linear filtering and prediction problems", *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[43] R. G. Brown and P. Y. C. Hwang, *Introduction to Random Signals and Applied Kalman Filtering*. Wiley, 2012.

[44] S. Challa and D. Koks, “Bayesian and Dempster-Shafer fusion”, *Sadhana*, vol. 29, no. 2, pp. 145–174, Apr. 2004.

[45] Y Bar-Shalom, P. K. Willet, and X. Tian, *Tracking and Data Fusion: A Handbook of Algorithms*. YBS Publishing, 2011.

[46] D. Musicki and R. Evans, “Joint integrated probabilistic data association: JIPDA”, *IEEE Transactions on Aerospace and Electronic Systems*, vol. 40, no. 3, pp. 1093–1099, Jul. 2004.

[47] D. Musicki, R. Evans, and S. Stankovic, “Integrated probabilistic data association”, *IEEE Transactions on Automatic Control*, vol. 39, no. 6, pp. 1237–1241, Jun. 1994.

[48] *Technical Reference Manual FLIR BLACKFLY GigE Vision*, version 14.0, FLIR Integrated Imaging Solutions Inc, 2017, 146 pp.

[49] M. Quigley, B. P. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, “ROS : An open-source robot operating system”, 2009.

[50] A. El-Rabbany, *Introduction to GPS: The Global Positioning System*. Artech House, Inc., 2002.

[51] *Pointgrey camera driver - ROS Wiki*, `http://wiki.ros.org/pointgrey_camera_driver`, Accessed: 2018-06-21, Open Source Robotics Foundation.

[52] *FlyCapture SDK*, `https://www.ptgrey.com/flycapture-sdk`, Accessed: 2017-12-17, FLIR Integrated Imaging Solutions Inc.

[53] *Velodyne driver - ROS Wiki*, `http://wiki.ros.org/velodyne?distro=kinetic`, Accessed: 2018-06-21, Open Source Robotics Foundation.

[54] OpenCV, *Camera calibration and 3d reconstruction*, goo.gl/KyzQSo, 2016.

[55] A. Dhall, K. Chelani, V. Radhakrishnan, and K. M. Krishna, “LiDAR-Camera Calibration using 3D-3D Point correspondences”, *ArXiv e-prints*, May 2017. arXiv: `1705.09785 [cs.RO]`.

[56] S. Garrido-Jurado, R. Munoz-Salinas, F. Madrid-Cuevas, and M. Marin-Jimenez, “Automatic generation and detection of highly reliable fiducial markers under occlusion”, *Pattern Recognition*, vol. 47, no. 6, pp. 2280 –2292, 2014.

[57] R. Munoz-Salinas and S. Garrido-Jurado, *ArUco Library*, 2013. [Online]. Available: `https://sourceforge.net/projects/aruco/`.

[58] A. Dhall, K. Chelani, V. Radhakrishnan, and K. M. Krishna, *ROS package to calibrate a camera and a LiDAR*, `https://github.com/ankitdhall/lidar_camera_calibration`, Accessed: 2018-06-19.

[59] M. A. Fischler and R. C. Bolles, “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”, *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981.

[60] W. Kabsch, “A solution for the best rotation to relate two sets of vectors”, *Acta Crystallographica Section A*, vol. 32, no. 5, pp. 922–923,

[61] B. Vik, *Integrated Satelite and Inertial Navigation Systems*. Norwegian University of Science and Technology, 2014.

[62] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”, *ArXiv e-prints*, Jun. 2015. arXiv: `1506.01497 [cs.CV]`.

[63] E. J. Tangstad, “Visual detection of maritime vessels”, Master Thesis, Norwegian University of Science and Technology, 2017.

[64] R. Girshick, “Fast R-CNN”, in *The IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015.

[65] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks”, in *Computer Vision – ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Springer International Publishing, 2014, pp. 818–833.

[66] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition”, *ArXiv e-prints*, Sep. 2014. arXiv: `1409.1556 [cs.CV]`.

[67] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “Visual object classes challenge 2012 dataset (voc2012)”, 2012. [Online]. Available: `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/`.

[68] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database”, in *CVPR09*, 2009.

[69] *Faster R-CNN: Towards real-time object detection with region proposal networks*, `https://github.com/ShaoqingRen/faster_rcnn`, Accessed: 2017-12-15.

[70] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, “Caffe: Convolutional Architecture for Fast Feature Embedding”, *ArXiv e-prints*, Jun. 2014. arXiv: `1408.5093 [cs.CV]`.

[71] *Caffe deep learning framework*, `http://caffe.berkeleyvision.org/`, Accessed: 2017-12-17.

[72] *Caffe for Faster R-CNN*, `https://github.com/ShaoqingRen/caffe/tree/faster-R-CNN`, Accessed: 2017-12-17.

[73] M. Forsyth and J. Ponce, *Computer Vision, A Modern Approach*. Pearson, 2012.

[74] R. Xu and D. Wunsch, “Survey of clustering algorithms”, *IEEE Transactions on Neural Networks*, vol. 16, no. 3, pp. 645–678, May 2005.

[75] A. K. Jain, M. N. Murty, and P. J. Flynn, *Data clustering: A review*, 1999.

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

[77] D. C. Hernndez, V. D. Hoang, and K. H. Jo, “Lane surface identification based on reflectance using laser range finder”, in *2014 IEEE/SICE International Symposium on System Integration*, 2014, pp. 621–625.

[78] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN", *ACM Trans. Database Syst.*, vol. 42, no. 3, 19:1–19:21, Jul. 2017.

[79] P. Horridge and S. Maskell, "Real-time tracking of hundreds of targets with efficient exact jpdaf implementation", in *2006 9th International Conference on Information Fusion*, 2006, pp. 1–8.

[80] I. J. Cox and M. L. Miller, "On finding ranked assignments with application to multitarget tracking and motion correspondence", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 31, no. 1, pp. 486–489, 1995.

[81] E. S. Henriksen, "Automatic testing of maritime collision avoidance methods with sensor fusion", Master Thesis, Norwegian University of Science and Technology, 2018.

[82] A. A. Gorji, R. Tharmarasa, and T. Kirubarajan, "Performance measures for multiple target tracking problems", in *14th International Conference on Information Fusion*, 2011, pp. 1–8.

[83] E. Brekke, O. Hallingstad, and J. Glattetre, "Tracking small targets in heavy-tailed clutter using amplitude information", *IEEE Journal of Oceanic Engineering*, vol. 35, no. 2, pp. 314–329, 2010.

[84] Y. Bar-Shalom, X. Rong Li, and T. Kirubarajan, *Estimation with Applications To Tracking and Navigation: Theory Algorithms and Software*, 1st ed. John Wiley & Sons, Inc, 2001.

[85] E. F. Wilthil, A. L. Flåten, and E. F. Brekke, "A target tracking system for asv collision avoidance based on the pdaf", in *Sensing and Control for Autonomous Vehicles: Applications to Land, Water and Air Vehicles*, T. I. Fossen, K. Y. Pettersen, and H. Nijmeijer, Eds. Springer International Publishing, 2017, pp. 269–288.

[86] E. Brekke, O. Hallingstad, and J. Glattetre, "Improved target tracking in the presence of wakes", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 2, pp. 1005–1017, 2012.

[87] A. E. Johnson, "Spin-images : A representation for 3-D surface matching", PhD Thesis, Carnegie Mellon University, 1997.

[88] K. O. Arras, O. M. Mozos, and W. Burgard, "Using boosted features for the detection of people in 2d range data", in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, 2007, pp. 3402–3407.

[89] B. Ristic and D. J. Salmond, "A study of a nonlinear filtering problem for tracking an extended target", in *7th International Conference on Information Fusion*, 2004, pp. 503–509.

[90] J. W. Koch, "Bayesian approach to extended object and cluster tracking using random matrices", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 44, no. 3, pp. 1042–1059, 2008.

[91] M. Feldmann, D. Franken, and W. Koch, "Tracking of extended objects and group targets using random matrices", *IEEE Transactions on Signal Processing*, vol. 59, no. 4, pp. 1409–1420, 2011.

[92] K. Granström and U. Orguner, “New prediction for extended targets with random matrices”, *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, no. 2, pp. 1577–1589, 2014.

[93] K. Granström, S. Reuter, D. Meissner, and A. Scheel, “A multiple model phd approach to tracking of cars under an assumed rectangular shape”, in *17th International Conference on Information Fusion*, 2014, pp. 1–8.

[94] K. A. Ruud, “Lidar tracking of vessels at sea using an ellipsoidal contour model”, Master Thesis, Norwegian University of Science and Technology, 2018.