Master's thesis

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

Didrik Grove

Multi-sensor multi-target tracking using LIDAR and camera in a harbor environment

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke
Co-supervisor: Giorgio D. K. M. Kufoalor
June 2021

NTNU
Norwegian University of
Science and Technology

MARITIME
ROBOTICS

Didrik Grove

Multi-sensor multi-target tracking using LIDAR and camera in a harbor environment

Master's thesis in Cybernetics and Robotics
Supervisor: Edmund Førland Brekke
Co-supervisor: Giorgio D. K. M. Kufoalor
June 2021

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

Preface

This report presents the work done in my master's at the Norwegian University of Science and Technology (NTNU) spring of 2021. It is the final requirement to achieve a master's degree within Cybernetics and Robotics at NTNU.

A lot of work on the hardware setup has been done in my project thesis fall of 2021 [19]. In this project I quantify the consequences of different time synchronization methods in a target tracking setting.

I would like to thank my supervisors Giorgio D. K. M. Kufoalor from Maritime Robotics and Edmund Førland Brekke from NTNU for their guidance and advice during the period. I would also like to thank Maritime Robotics letting me use their hardware for this thesis, their employees have been of tremendous help when setting up and running their systems for the test scenarios.

Finally I would like to thank my family for all their patience and support, both while working on this thesis and during my time as a student.

Trondheim, June 14, 2021
Didrik Grove

Sammendrag

Andelen ulykker på havet som resultat av menneskelige feil er anslått å være mellom 60 og 90%. I situasjoner der mennesker ofte opptrer ulikt gitt samme utgangspunkt har et autonomt fartøy fordelen av å være forutsigbart, selv etter lengre perioder i drift. Det å kunne tolke omgivelsene på en pålitelig måte for å detektere andre fartøy er et høyst aktuelt tema innen forskning og utvikling av autonomi.

Denne avhandlingen presenterer en implementasjon av en JIPDA metode for å følge flere fartøy gitt målinger fra deteksjoner i kamerabilder og LIDAR. Det er samlet inn data fra ulike testscenario i områder med begrensede muligheter for trygg manøvrering. To båter er utstyrt med GPS-mottakere for logging av posisjon og deltar som en del av scenarioene. GPS-posisjonen sammenlignes med posisjonsestimatene fra målfølgingsalgoritmen for å kvantifisere ytelsen i de gitte scenarioene.

Prosessering av sensordata fra både LIDAR og kamera er beskrevet i detalj. Bildene fra testscenarioene er annotert og vi oppnår en gjennomsnittlig deteksjonsnøyaktighet på mindre enn 25% ved å bruke detekteringsalgoritmer som er forhåndstrent på andre datasett. Ved å trene vår egen algoritme fra grunnen av oppnår vi en deteksjonsnøyaktighet på 95%. Detekteringene i kamerabildene blir konvertert til en retningsmåling som blir prosessert i målfølgingsalgoritmen.

Denne implementasjonen av målfølgingsalgoritmen gir en økt ytelse dersom målene som skal følges er innenfor kameraenes synsfelt. Det vises også til at ytelsen til målfølgingsalgoritmen blir dårligere når målene er utenfor kameraenes synsfelt, ettersom algoritmen antar et 360 graders synsfelt for alle sensorene. Kameramålingene er i stand til å opprettholde et tilfredstillende posisjonsestimat uten hjelp fra LIDAR i rundt 10 sekunder. Etter hvert blir usikkerheten i estimatet stor nok til at flere retningsmålinger blir assosiert med et enkelt posisjonsestimat, noe som fører til at kovariansen vokser raskt og at estimatet blir termintert som følge av for stor usikkerhet. I tilfeller der algoritmen mottar sporadiske målinger fra LIDAR viser denne rapporten at kameramålingene sørger for tidligere initiering og å enklere opprettholde et estimat.

Abstract

The amount of accidents at sea happening as a consequence of erroneous human action is between 60 and 90%. While human operators often behave differently to the same situations, an autonomous vessel has the advantage of being more predictable even after long periods of operation. Being able to reliably sense the surroundings and detect other vessels to make quantified decisions is a hot research subject within autonomy.

This thesis presents an implementation of a JIPDA-based multi-sensor multi-target tracker where the data from a camera object detector and a LIDAR are fused together. Test data sets are collected for different scenarios in congested waters with limited room for safe navigation. Two targets in these test scenarios are equipped with GPS receivers that are used for validating the accuracy and performance of the tracking system.

The data processing pipelines are described in detail for both the LIDAR and the camera detector. The data from the test sets is annotated and we observe a mean average precision (mAP) of less than 25% when running a pre-trained detector on the set. Training our own object detector from scratch we are able to achieve an mAP of over 95%. The detections in the camera images are extracted as bearing measurements which are used in the tracking pipeline.

Tests show that the current implementation of the tracker increases the performance while the targets are within the camera's field of view. The bearing measurements do however yield lower performance on the data sets when the targets are outside the camera's field of view, as the tracker assumes a full 360 degree field of view from all sensors. The camera is able to maintain a track for a few seconds without the help of a LIDAR, but gets terminated quickly as the uncertainty grows too large. The large uncertainty introduced in the bearing measurements causes multiple bearing measurements to be gated to a single target which adds uncertainty to the estimate. The LIDAR alone has trouble initiating a good track on a small target, but with sparse LIDAR measurements the camera is shown to help with initiating and maintaining the track.

Contents

List of figures viii

List of tables xi

List of abbreviations xii

1 Introduction 1

1.1 Motivation . . . 1

1.2 Problem description . . . 2

1.3 Related work . . . 3

1.4 Report outline . . . 4

2 Theory 5

2.1 Camera detections . . . 5

2.1.1 Convolutional neural networks . . . 5

2.1.2 Building blocks . . . 7

2.1.3 Metrics for bounding box detection . . . 11

2.1.4 Image detection to relative bearing . . . 13

2.2 Light Detection And Ranging (LIDAR) detections . . . 15

2.2.1 LIDAR measurements . . . 15

2.2.2 Filtering . . . 17

2.2.3 Clustering . . . 18

2.3 Relative positioning between sensors and detections . . . 19

2.4 Target tracking . . . 20

2.4.1 Kalman filter (KF) to extended Kalman filter (EKF) . . . 21

2.4.2 Interacting multiple models (IMM) . . . 24

2.4.3 Probabilistic Data Association (PDA) . . . 27

2.4.4 PDA to JIPDA . . . 29

2.4.5 Visibility state modelling . . . 32

2.4.6 Multi sensor tracking . . . 34

2.5 Tracking in autonomy . . . 36

2.5.1 Autonomy . . . 36

2.5.2 Maritime rules of the sea . . . 37

3 Method 38

3.1 Test scenarios . . . 39

3.1.1 Scenario one - Overtaking 40
3.1.2 Scenario two - Occlusion of target 41
3.1.3 Scenario three - Passing of target 42

4 Experiment setup **43**
4.1 Hardware setup 44
4.1.1 Otter USV 44
4.1.2 Sensor rig 45
4.1.3 Target boats 50
4.2 Pipeline implementation 52
4.2.1 Camera pipeline 53
4.2.2 LIDAR pipeline 59
4.2.3 Tracking pipeline 63

5 Results and discussion **68**
5.1 Sensor processing delays 68
5.2 JIPDA association delays 70
5.3 Scenario one - Overtaking 72
5.4 Scenario two - Occlusion of target 78
5.5 Scenario three - Passing of target 83
5.6 Loss of LIDAR measurements 87

6 Conclusion and future work **90**
6.1 Conclusion 90
6.2 Future work 91

Bibliography **93**

Appendix **I**

List of Figures

1 An object classifier would be able to tell that this image contains a boat but give no further spatial information. 5
2 Object detections with YOLO V3 using an IR-camera. 6
3 Fully connected layer in a neural network 7
4 Non-linearities in the Sigmoid and ReLU activation functions [13]. 8
5 A convolution applied to a input image 9
6 The max pooling operation. 10
7 Predicted bounding box P and ground truth T 11
8 Precision-Recall curve 12
9 Pinhole camera model [34] 13
10 Barrel distortion [48]. 14
11 Polar and Cartesian coordinates [17]. 15
12 Resolution cells in Cartesian and polar coordinate system [35] 16
13 Point cloud visualization of LIDAR data in rviz 16
14 LIDAR Clusters [16] 18
15 Transformation between two coordinate systems [15]. 19
16 The steps in a Kalman filter 21
17 The steps in a Kalman filter using mathematical notation 22
18 The steps in an extended Kalman filter 23
19 Two vessels that behave very differently seen through a LIDAR 24
20 The prediction steps in multiple model filtering. 25
21 Multiple model filtering. 26
22 Univariate Gaussian mixture. 26
23 Multivariate Gaussian mixture[26]. 26
24 Probabilistic Data Association Filtering. 27
25 Integrated Probabilistic Data Association Filtering. 30
26 Joint Probabilistic Data Association Filtering. 31
27 Otter 27 39
28 Movement of the boats in the first scenario. 40
29 Movement of the boats in the second scenario. 41
30 Movement of the boats in the third scenario. 42
31 Overview of the testing area with the three boats used for data collection. 43
32 Otter USV 44
33 Otter targa for autonomy development 45
34 Genie Nano C4040 46

35 Fujinon Cf8ZA-1S 46
36 Field of view of EO cameras. 46
37 Ouster OS-1. 47
38 Sentiboard 48
39 ToC vs. ToA timestamping precision. 49
40 Time synchronization setup 49
41 Juggernaut 50
42 Otter 9 51
43 Pipeline in ROS 52
44 The broken camera mount. 55
45 Camera images before (top) and after (bottom) the camera mount is broken. 56
46 The output from the object detector transformed into bearing measurements. 57
47 The raw point cloud output by the LIDAR. 59
48 The output from the voxel grid filtering and segmenter. 60
49 The output from Euclidean clustering and land filtering. 61
50 Newly initiated track based on bearing measurements 67
51 Processing delays in the LIDAR pipeline 68
52 Processing delays in the object detection pipeline 69
53 Delays without land filtering of the LIDAR points 70
54 Tracks initiated around measurements on land 71
55 Time delay in tracker relative to association hypotheses 71
56 Range from ownship to Juggernaut in scenario one 72
57 Estimated and ground truth position of Juggernaut using only LIDAR . 73
58 Estimated and ground truth position of Juggernaut using LIDAR and camera 73
59 Existence probability of Juggernaut 74
60 Estimation error of Juggernaut (with GPS position as ground truth) . . . 75
61 Range to Otter 9 in scenario one 76
62 Estimation error of Otter 9 (with GPS position as ground truth) 76
63 Mode probabilities for Otter 9 using LIDAR and camera 77
64 Mode probabilities for Otter 9 using only LIDAR 77
65 Range from ownship to Juggernaut in scenario two 78
66 Estimated and ground truth position of Juggernaut using only LIDAR . 79
67 Estimated and ground truth position of Juggernaut using LIDAR and camera 79
68 Existence probability of Juggernaut 80
69 Range to Otter 9 in scenario two 81
70 Estimation error of Juggernaut (with GPS position as ground truth) . . . 81

71 Raw point cloud showing the two target boats 82
72 Range from ownship to Juggernaut in scenario three 83
73 Estimated and ground truth position of Juggernaut using LIDAR 84
74 Estimated and ground truth position of Juggernaut using LIDAR and camera 84
75 Mode probabilities of the Juggernaut track using LIDAR and camera . . 85
76 Estimated and ground truth position of Otter 9 using LIDAR 86
77 Estimated and ground truth position of Otter 9 using LIDAR and camera 86
78 Existence probability after LIDAR dropout 87
79 Mode probabilities during sensor dropout 87
80 Camera detections a few seconds after LIDAR dropout 88
81 Camera detections a few seconds after LIDAR dropout 88
82 Position estimate error compared to GPS position 89
83 Position estimate error compared to GPS position 89

List of Tables

1 The steps in an Euclidean clustering method [36]. 18
2 The notation used in this report relating to target tracking. 20
3 The sub- and superscripts used in this report relating to target tracking. 20
4 LIDAR Specifications. 47
5 Simrad HS60 GPS Specifications [25]. 50
6 Zed F9P Specifications [46]. 51
7 Performance of a pre-trained YOLO model on the data set. 53
8 Spread of annotated data from EO cameras. 53
9 The size of each dataset used for training, validation and testing. 54
10 Performance of trained object detector on data set. 54
11 Parameters for training the neural network. 55
12 Initial camera specific tracker parameters. 58
13 Parameters for voxel grid filtering. 59
14 Parameters for the Euclidean cluster segmenter. 60
15 Parameters for the Euclidean clustering. 61
16 Initial LIDAR specific tracker parameters. 62
17 Noise in the process models 63
18 Initial probability for the process models 64
19 Final tracker specific parameters. 67

List of abbreviations

LIDAR	Light detection and ranging
ROS	Robotic operating system
AIS	Automatic identification system
RADAR	Radio detection and ranging
YOLO	You only look once
IMM	Interative muliple models
JIPDA	Joint integrated probabilistic data association algorithm
GPU	Graphics processing unit
SSD	Single shot detector
IoU	Intersection over union
TP	True positive
FP	False positive
AP	Average precision
RANSAC	Random sample consensus
SLAM	Simultaneous localization and mapping
(E)KF	(Extended) Kalman filter
MC	Markov chain
SEA	Society of automotive engineers
COLREG	Convention on the international regulations for preventing collisions at sea
GPS	Global positioning system
USV	Unmanned surface vehicle
RTK	Real time kinematic
INS	Inertial navigation system
MRU	Motion reference unit
PPS	Pulse per second
EO	Electro optical
IR	Infrared
GNSS	Global navigation satellite systems
I/O	Input / Output
PPM	Parts per million
ToC	Time of capture
ToA	Time of arrival
CT	Constant turn
CV	Constant velocity

1 Introduction

1.1 Motivation

Over the last years the funding and commercial interest for autonomy has skyrocketed. With the technology that has been developed during the last two decades, autonomy has gone from being a term only used in scientific context to being something we interact with and deal with in our daily life. Many of us now have some form of artificial intelligence in our living rooms, we have access to self-driving cars and companies are able to land rocket boosters returning from space, all by using some form of feed back control.

Because of the oil industry and a thriving fish industry, Norway is one of the leading countries when it comes to research and development of maritime solutions. With the increase in demand for autonomy in the maritime industry, Maritime Robotics seek to increase the situational awareness of their vessels and become one of the leading developers within the field. Being able to detect and get an understanding of the behaviour of other vessels is a vital part of autonomy, and while human operators behave differently and often respond in different ways to the same situations, an autonomous vessel has the advantage of being more predictable even after longer periods of operation. Being able to reliably sense the surroundings to make such quantified decisions is still a hot research subject, and this thesis aims to explore the limitations and test the robustness of a multi sensor tracking system using cameras and a LIDAR in congested waters.

Maritime Robotics is making a sensor hub which will be used in their autonomy development project. This hub is mountable on vessels of different sizes and is built with flexibility in mind, with interchangeable sensors and the possibility to make adjustments and change data synchronization methods. The results from this thesis will contribute to the development and optimization of this sensor hub.

1.2 Problem description

Having multiple sensors detecting objects and sensing the surrounding environment is a recognized method for obtaining a reliable and robust tracking system. The main goal of this thesis is to quantify and test the robustness of a multi sensor multi target tracker using camera and LIDAR. Tuning and setup of the sensors to get reliable sensor data is also a part of the thesis to avoid poor sensor data cluttering our results when judging the quality of the tracking system. The work done in this thesis can be split into four parts.

First, raw sensor data for development, validation and testing of the algorithms is gathered. The data sets are collected at the start of the period for continuous testing during development and to avoid surprises which could occur if it was done later in the period. The goal is to gather data in congested waters with limited room for safe navigation where the LIDAR and the camera are utilized to the fullest.

After collection we move over to processing of the sensor data. Interpreting and getting as much information from the data as possible will give a better foundation for the target tracker. Having stable sensor processing algorithms will give an optimal input to the tracker and give less noise in the results. While data collection is done beforehand for continuous development of processing algorithms, some data sets are stored as test sets that will be reviewed towards the end.

The VIMMJIPDA tracker developed by Audun Hem in his project and masters thesis [22] will be implemented into the situational awareness pipeline at Maritime Robotics. The tracker is a python module and will be restructured into a ROS-node. In its current state the tracker supports RADAR and AIS measurements in a Cartesian coordinate plane but will be extended to support bearing only measurements from an object detector.

At last the performance of this multi sensor multi target tracker will be tested in different scenarios with a focus on robustness and reliability. Situations where LIDAR drop out is experienced will be reviewed and the accuracy and performance of the multi sensor tracker in these states will be quantified and discussed.

1.3 Related work

There has been a lot of development within the field of sensor fusion the last years, and while LIDAR and RADAR have often been the sensors of choice, for the last few decades the camera has become a commonly used sensor. The recent development of open-source computer vision platforms have made powerful and optimized neural networks available for the average user. Fusing the measurements of LIDAR and camera has been a popular approach the last twenty years. In "Sensorfusion using spatiotemporal aligned video and LIDAR for improved vehicle detection [33]" regions of interest for the object detector is generated using LIDAR measurements and in "Perception for collision avoidance and autonomous driving [3]" a high level fusion approach for object tracking is implemented using LIDAR and cameras. "A multi-sensor fusion system for moving object detection and tracking in urban driving environments [11]", "LIDAR and vision based pedestrian detection systems [37]" and "Vehicle tracking with lane assignment by camera and LIDAR sensor fusion [47]" are other articles where sensor fusion between camera and LIDAR are reviewed. YOLO [39] is used as an object detector in the situational awareness pipeline developed by Maritime Robotics and is explained more extensively in subsection 2.1.1.

The work in this thesis is a continuation of the work in the project done in the fall of 2020 [19]. This project thesis quantifies the consequences of poor time synchronization and time stamping on a target tracking system using LIDAR and camera. For more information about the time synchronization setup and constraints in the sensor rig the reader is refereed to this project thesis. The tracker used is this masters thesis is an extension of Audun Hems tracker developed in his masters [22] and more thoroughly described in his research article [7]. This is a joint integrated probabilistic data association (JIPDA) tracker applying interactive multiple models (IMM) where the visibility of the targets is modelled. This way the targets can enter an invisible state. The work flow and the modules used in this tracker is explained extensively in section 2.4.

Similar implementations have been done by Vegard Kvamsgård in his masters thesis [29] and by Knut Turøy in his project and [45] and masters thesis [45]. Turøy implemented a multi sensor tracker using RADAR, LIDAR, electro-optical and infrared cameras. While his thesis investigates the quality of tracks using the multi sensor tracker, this thesis will quantify the quality and robustness of the tracks in congested waters where the system loses access to a sensor, either knowingly or unknowingly.

1.4 Report outline

Section 2 will give an introduction to theory relevant for the later sections. Sections 2.1 and 2.2 explains relevant data processing methods for camera and LIDAR that are used in this project. It also gives a short introduction to other, similar processing algorithms. Section 2.4 describes different modules and the work flow of a Bayesian target tracker doing measurement fusion on multiple targets using multiple sensors. Section 2.5 gives a short introduction to the rules of the sea, specifically for vessels in the areas where scenarios are being conducted.

Section 3 expands on the methods used for validating the theory and verifying the results from the scenarios. In Section 3.1 we described the three different scenarios for data collection in detail.

In Section 4.1 the hardware setup and the boats used in the test scenarios are expanded upon. The software pipeline for the camera and LIDAR data is described in Section 4.2 and the implementation of the target tracker with bearing measurements is described in Section 4.2. The tuning of the pipelines and thought-process behind the implementations are also expanded upon in this Section.

In Section 5 we present the results that were gained through the test scenarios. In Section 5.1 we present the delays in the processing pipeline and their impact on the target tracker. In Section 5.2 we show how the association hypotheses impact the processing time and in sections 5.3, 5.4 and 5.5 we discuss the tracker performance on the three test scenarios. In Section 5.6 we review how the tracker behaves when LIDAR measurements are lost.

In Section 6 the results are concluded and suggestions for future work is presented.

2 Theory

2.1 Camera detections

After AlexNet achieved a 16% error rate in the PASCAL VOC challenge the use of GPUs running convolutional neural networks became the standard for solving computer vision tasks. While most developers of self-driving cars use LIDAR as the short-range sensor of choice it is most often used in conjunction with a camera detection system. With the skyrocketing development within the field of vision systems, large companies like Tesla even see this as the main sensor for sensing the environment [40].
Most modern vision systems use some sort of convolutional neural network. These networks are a class of deep neural networks, most commonly used in image analysis. Neural networks are built on the concept of neurons behaving somewhat similar to the neurons in the human brain, where different inputs trigger different reactions and outputs. This section will explain the workflow of convolutional neural networks, explain some of the most used building blocks, scratch the surface of some of the most common object detection algorithms and explain how these image detections can be used in sensor fusion.

2.1.1 Convolutional neural networks

The goal of a regular classification algorithm is to classify the content of an image like Figure 1, giving feedback on whether the image is of a car, boat or a plane. An object detection algorithm also attempts to draw a bounding box around the object to describe where in the image it is located like Figure 2.

Figure 1: An object classifier would be able to tell that this image contains a boat but give no further spatial information.

A regular classification algorithm does not take into account more than one object within the image, while an object detection algorithm can detect several objects of interest within a single image without knowing the amount of objects beforehand. A naive approach to this problem would be to split the original image into a very large amount of smaller images and then run the object detection algorithm on each image. Because the objects can have different aspect ratios and be located in different sections of the image this would not be computationally tractable on a real-time system, as the amount of possible locations would be extremely large. Doing this in an efficient manner is the problem that modern object detection algorithms attempt to solve.

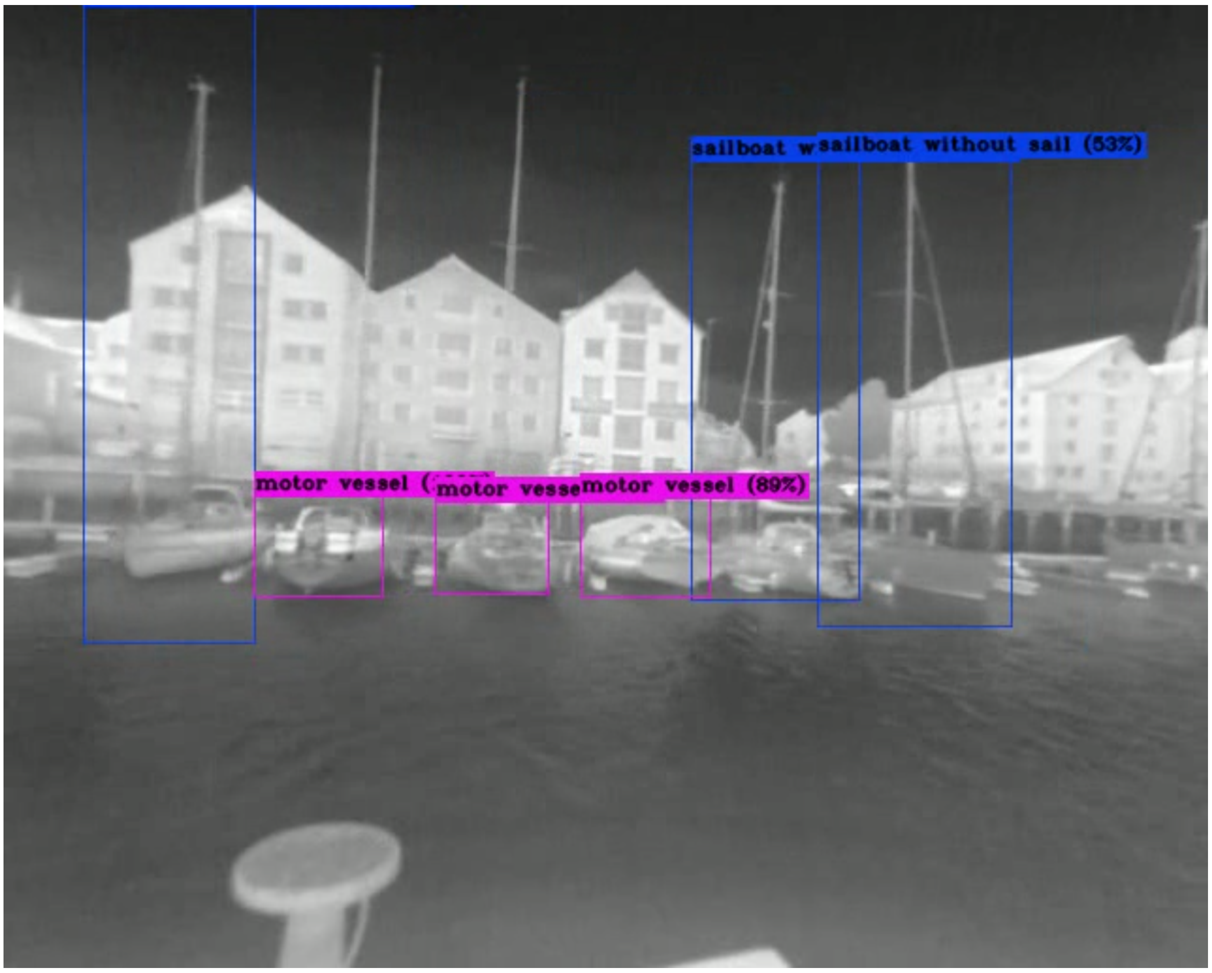

Figure 2: Object detections with YOLO V3 using an IR-camera.

Two of the most popular object detection algorithms for real-time use are SSD and YOLO. SSD [32] uses extra convolutional layers of different resolutions that predict where the bounding boxes are located and their size, then SSD runs a classification algorithm on each suggested bounding box. YOLO [39] instead applies a single neural network to the full image (hence the name "You Only Look Once"), which divides the image into regions and predicts bounding boxes and probabilities for each region. Because running a SSD is a two-step process it is a bit slower than YOLO, however image processing still happens within reasonable time for a real-time system. The trade-off for YOLO is limited accuracy on objects that are small and smaller objects located close to each other due to the limited size of the region proposals.

2.1.2 Building blocks

Classic convolutional neural networks usually apply the same building blocks, although in different orders, sizes, structures and with different optimization methods.

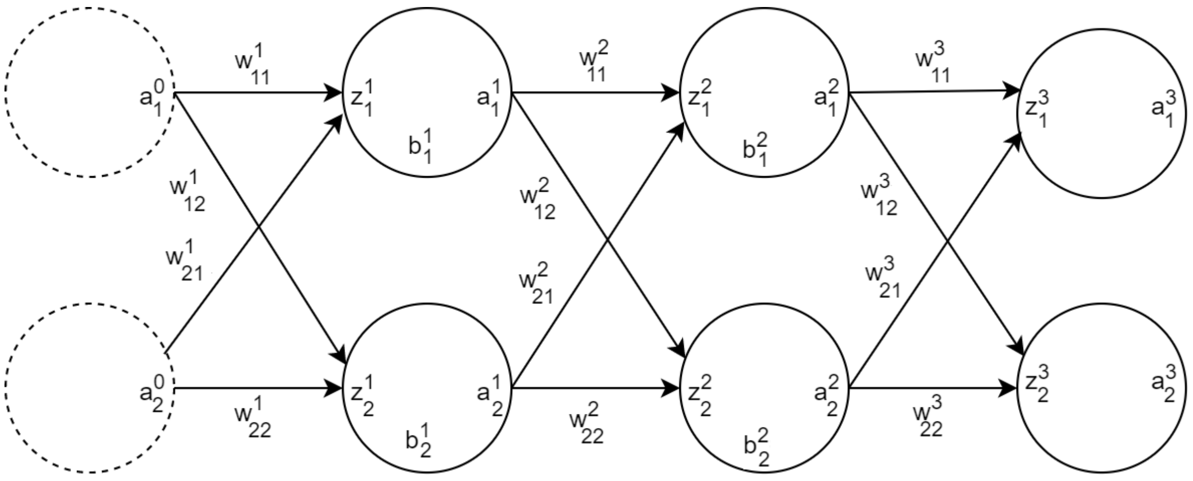

Figure 3: Fully connected layer in a neural network

A regular fully-connected layer like the one shown in Figure 3 is the most basic building block. Each neuron in such a layer is connected to all neurons in the previous layer. Each connection between the neurons is weighted and each neuron applies an activation function and has a bias.

$$w^1 = \begin{bmatrix} w^1_{11} & w^1_{12} \\ w^1_{21} & w^1_{22} \end{bmatrix} \quad w^2 = \begin{bmatrix} w^2_{11} & w^2_{12} \\ w^2_{21} & w^2_{22} \end{bmatrix} \quad w^3 = \begin{bmatrix} w^3_{11} & w^3_{12} \\ w^3_{21} & w^3_{22} \end{bmatrix}$$

$$b^1 = \begin{bmatrix} b^1_1 \\ b^1_2 \end{bmatrix} \quad b^2 = \begin{bmatrix} b^2_1 \\ b^2_2 \end{bmatrix}$$

A forward pass through the weights and the biases is done with the function

$$z^i = (w^i)^T a^{i-1} + b^i$$

As this function is linear the output of the forward pass is not confined to any range and will not be able to learn non linearities in the data. Simply adding a non-linear activation function solves these problems and improves learning. The sigmoid and the ReLU (with modifications) are some basic and often used activation functions shown in Figure 4.

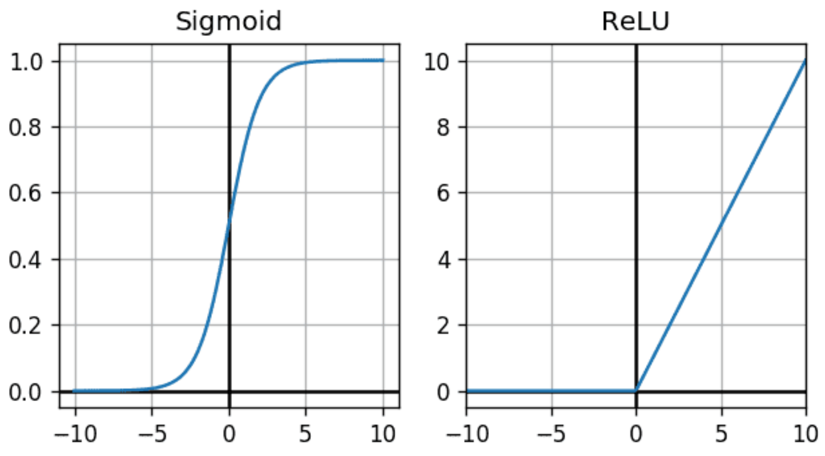

Figure 4: Non-linearities in the Sigmoid and ReLU activation functions [13].

$$\textit{Sigmoid} \qquad a = f(z) = \frac{1}{1+e^{-z}} \qquad \frac{df(z)}{dz} = f(z)(1-f(z))$$

$$\textit{ReLU} \qquad a = f(z) = max(0,z) \qquad \frac{df(z)}{dz} = \begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{otherwise} \end{cases}$$

The bias and the weights are modifiable parameters that are tuned during training by doing a backward pass. After inputting data to the first layer and performing a forward pass through the network it produces an output value for each of the classes it is trained to recognize. A cost function is used to calculate the cost of the error when comparing the output to the actual ground truth values. Using the cost the backwards pass iterates backwards into the network and tunes the weights and biases by calculating the gradients. The gradient at each weight/bias gives a value on how much it impacted the final result and how much the weight needs to be changed. Doing this over many iterations with different inputs one gets at a network that learns to recognize different classes.

While the fully connected layer is the most basic building block the convolution is the most commonly used building block for a convolutional neural network. A convolutional layer applies filters of different sizes to the input data. During the convolution each block in the filter is multiplied with the input data, the blocks are summed together and gives an output value as in Figure 5.

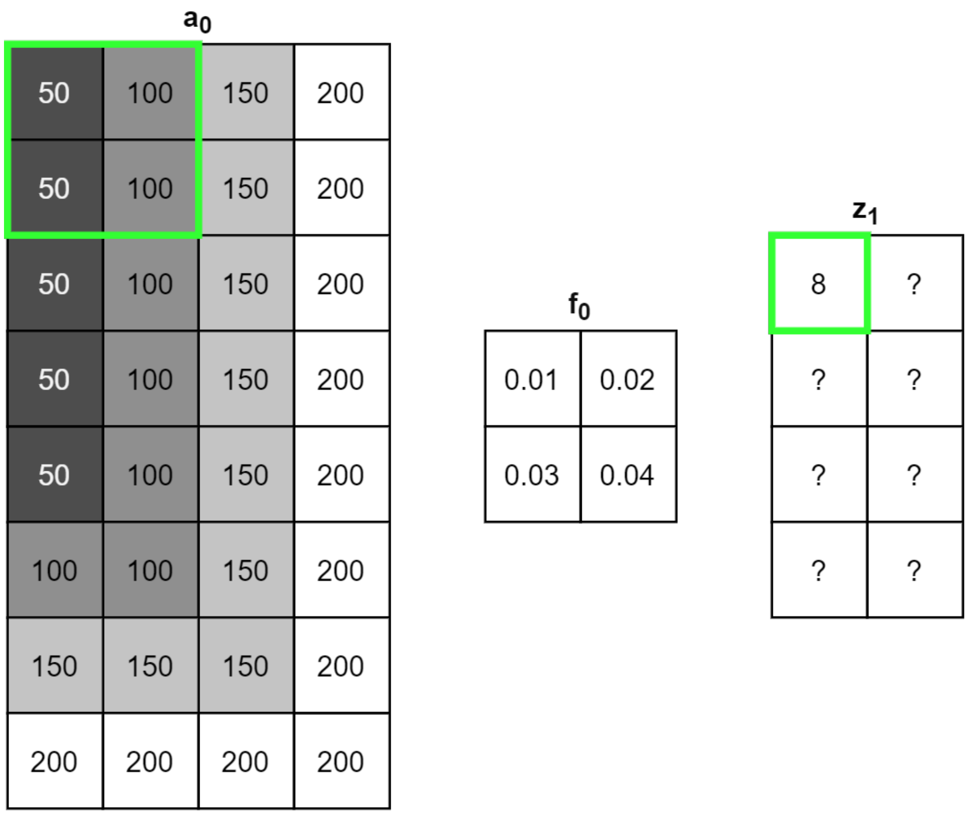

Figure 5: A convolution applied to a input image

$$z_{1,11} = 0.01 \cdot 50 + 0.02 \cdot 100 + 0.03 \cdot 50 + 0.04 \cdot 100 = 8$$

While the fully connected layer mainly looks at individual pixel values, the filters in this layer are able to learn more complex features such as edges and corners. The convolutional layer can also contain weights for depth channels, and in such a way take into account color combinations for the features. They are hence very useful and often used for analyzing images.

A pooling layer is often applied between successive convolutional layers. This layer reduces the spatial size of the network to reduce the amount of parameters and computations done. The most commonly used pooling layer is performing a max pooling operation like that shown in Figure 6.

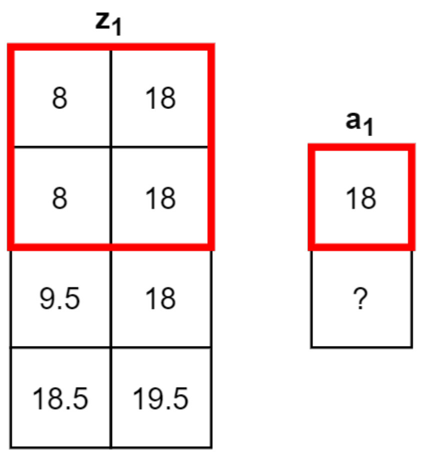

Figure 6: The max pooling operation.

Reducing the amount of parameters in the network helps against overfitting. Overfitting happens when the network keeps specializing on recognizing the images in its training dataset but at the same time gets lower performance on objects that are outside of the dataset. The goal of an object detection network is most often to be general and able to recognize as many objects within its scope as possible, and techniques such as max pooling layers are applied to reduce said overfitting. The gradient for the max pooling layer is also equal to 1 for the chosen neuron and 0 for all others. During the backwards pass this gradient ensures that only the weights and neurons triggered and used towards the output will be modified. This helps splitting the network into different regions where each part learns different features. Because of this gradient only the weights that are relevant for the output will be modified, which increases generalization.

While max pooling is the most commonly used pooling layer, we also have average and L2-norm pooling layers. Average pooling simply takes the average over the region, and the L2-norm takes the square root of the sum of the squares of the activation in the region.

2.1.3 Metrics for bounding box detection

Given a set of images with predicted bounding boxes and truth values we can calculate metrics for bounding box detection to estimate the accuracy of our network. For each detection we calculate the Jaccardi overlap, also called intersection over union (IoU).

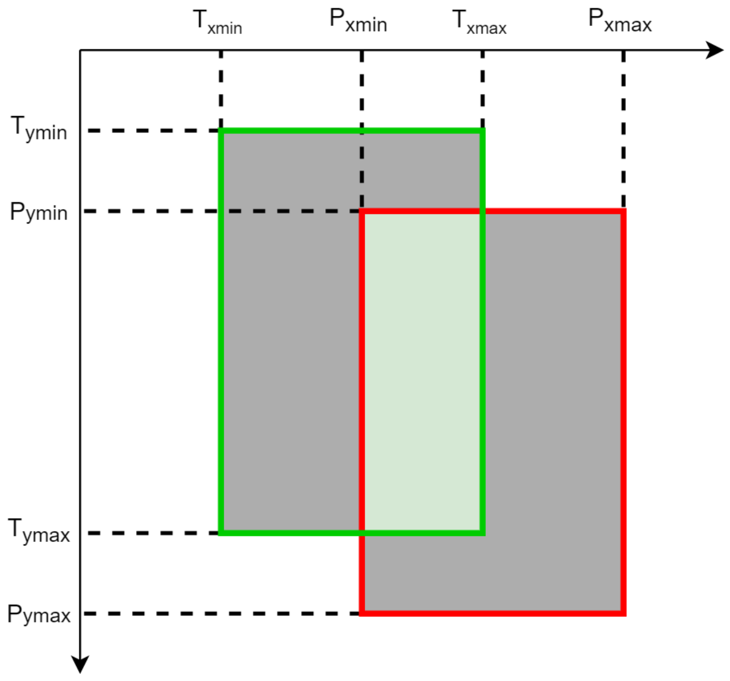

Figure 7: Predicted bounding box P and ground truth T

With the detection in Figure 7 we calculate the overlap with the following equations

$$I_{width} = min(P_{xmax}, T_{xmax}) - max(P_{xmin}, T_{xmin})$$
$$I_{height} = min(P_{ymax}, T_{ymax}) - max(P_{ymin}, T_{ymin})$$
$$I_{area} = I_{width} * I_{height}$$
$$U_{P,area} = (P_{xmax} - P_{xmin})(P_{ymax} - P_{ymin})$$
$$U_{T,area} = (T_{xmax} - T_{xmin})(T_{ymax} - T_{ymin})$$

$$IoU = \frac{I_{area}}{U_{P,area} + U_{T,area}}$$

If the IoU gives a score over the detection threshold we classify the detection as a true positive (TP) while detections under the treshold are classified as false positives (FP).

After calculating the total number of true positives and false positives, precision and recall can be calculated using the following formulas.

$$Precision = \frac{\text{TP}}{\text{All Detections}}$$

$$Recall = \frac{\text{TP}}{\text{Numer of ground truths}}$$

Precision gives the rate of correct predictions in the total set of predictions. Recall gives a ratio of correct detections compared to the amount of expected detections.

As we iterate through the set of predictions for a class we can plot the precision against the recall value like in Figure 8. The general definition of the average precision (AP) for a class is the area under this curve.

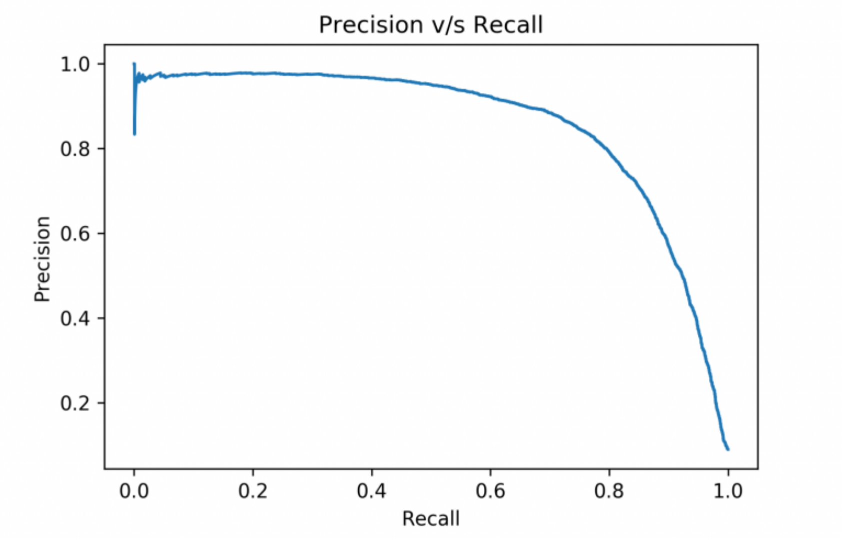

Figure 8: Precision-Recall curve

$$AP = \frac{1}{11} \sum Precision(recall)$$

While AP is the average precision for a single class, we also use mean average precision (mAP) for the total set of classes for a neural network.

$$mAP = \frac{\sum AP}{n}$$

2.1.4 Image detection to relative bearing

As detections with cameras in a mono-setup are done in 2D space they can only give accurate information about an object's height and angular position relative to the camera, and give limited information about the distance to a target. With the recent development in convolutional neural networks, however, they are also a very convenient tool for object classification to find information about the dynamic properties of the target. They are also often applied as a tool for a human-machine interface [28].

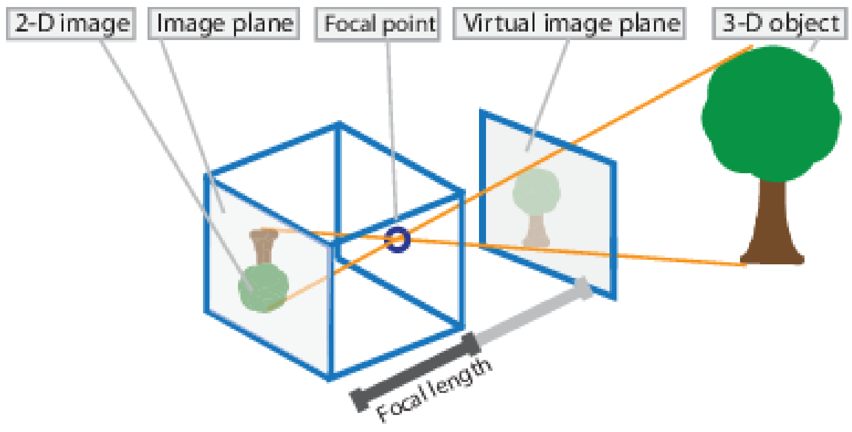

Figure 9: Pinhole camera model [34]

Accurate static transformations to the camera and its exintric, intrinsic and distortion parameters are needed to precisely describe the transformation of an object in the 2D image relative to the camera in 3D space. Newer cameras have specialized lenses that are built to minimize distortion caused by the curvature of the camera lens, but in general there is always a certain degree of distortion to account for. This is more thoroughly explained in [21].

The pinhole camera model in Figure 9 describes transformations between three different coordinate frames. The world coordinate frame represents coordinates of the 3D object in the real world, and the virtual image plane represents 3D coordinates in the camera, with the origin at the center of projection and the Z-axis as the optical axis. The 2D image plane is pixel coordinates within the image. The transformation from the 3D object in the world frame to the 3D object in the camera frame is given as a standard 3D coordinate transformation like the one described in Section 2.3, although it is often denoted as the extrinsic calibration matrix M_{ex}. The transformation between the virtual image plane and the 2D image plane is given with the intrinsic calibration matrix, which is given on the following form.

$$M_{in} = \begin{pmatrix} fs_x & 0 & o_x \\ 0 & fs_y & o_y \\ 0 & 0 & 1 \end{pmatrix} \tag{1}$$

Where the parameters are as follows

f	Focal length
(o_x, o_y)	Piercing point coordinates
s_x	Pixel size in x-axis
s_y	Pixel size in y-axis

This can be used to achieve a mapping between a vector in the world frame $\overrightarrow{X}$ to pixel coordinates $\overrightarrow{p}$ as a set of linear transformations

$$\overrightarrow{p} = M_{in}M_{ex}\overrightarrow{X} \tag{2}$$

The intrinsic parameters describe the camera field of view. If this is known beforehand one can also use the field of view together with pixel width to create a linear mapping between bearing angle and pixel coordinates. Neither of these methods does however take into account any distortion of the pixels. Because of the natural curvature of the camera lens one must often correct for radial lens distortion like barrel distortion shown in Figure 10. Letting $(\hat{x}, \hat{y})$ be the undistorted image coordinates and k_1, k_2 be the distortion coefficients, a simple radial distortion model can be given by the following equations:

$$\begin{aligned} \hat{x} &= x(1 + k_1(x^2 + y^2) + k_2(x^2 + y^2)^2) \\ \hat{y} &= y(1 + k_1(x^2 + y^2) + k_2(x^2 + y^2)^2) \end{aligned} \tag{3}$$

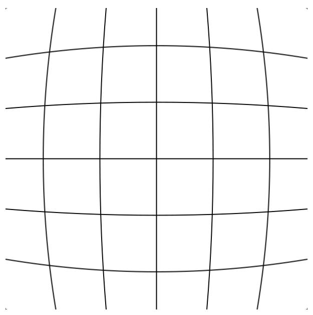

Figure 10: Barrel distortion [48].

2.2 Light Detection And Ranging (LIDAR) detections

While detecting objects and sensing the environment using a camera is relatively new within autonomy, the LIDAR has been the work horse at short ranges for decades. Complex tasks like docking to the international space station (ISS) has been done using LIDAR technology because of its high precision at shorter ranges [24].
New state-of-the-art LIDAR technology like solid state sensors is reducing the cost by tenfolds, increasing sensor range up to 200 meters and giving the ability to sense the velocity of an object in 3D space [14]. While object detectors can be run on relatively cheap cameras, the LIDAR is however hindered by its significant cost.

2.2.1 LIDAR measurements

While an EO-camera captures the reflected natural light in its lens, the LIDAR detects nearby objects by emitting light at a certain wavelength and measures the time of flight of the reflections. The LIDAR contains filters that excludes all light outside the desired wavelength, and in an automotive application the beam is often emitted and reflected through a rotating mirror to achieve a high field of view. The LIDAR outputs detections in a spherical coordinate frame, with range calculated by time of flight.

$$\mathrm{R} = \frac{\text{Speed of light} \times \text{Time of flight}}{2} \tag{4}$$

The rotated angle of the mirror gives the azimuth angle α. Newer LIDARs also have some vertical resolution with mirrors and reflectors mounted at different angles to add elevation ω.

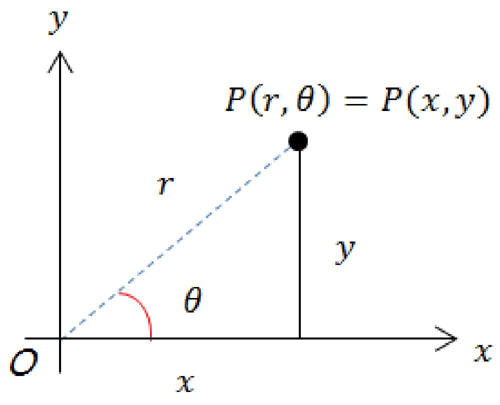

Figure 11: Polar and Cartesian coordinates [17].

As seen in Figure 11 these can be transformed into a Cartesian coordinate frame using some basic trigonometric functions

$$\begin{aligned} x &= R\cos\omega\sin\alpha \\ y &= R\sin\omega\sin\alpha \\ z &= R\cos\alpha \end{aligned} \tag{5}$$

It should however be noted that because a sensor with measurements in polar coordinates does not accurately represent squares in a Cartesian coordinate system, there is a loss of accuracy when doing such a transformation. While the resolution cells in the Cartesian coordinate systems is represented as squares, the resolution in the polar coordinate system can be represented as rectangles from the range and bearing resolution a shown in Figure 12.

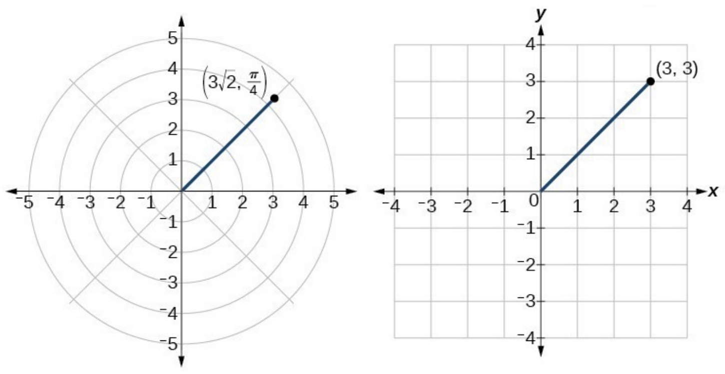

Figure 12: Resolution cells in Cartesian and polar coordinate system [35]

A state of the art LIDAR can output several hundred thousand such reflections/data points at a high frequency. These data points are usually gathered in a point cloud which is then filtered and processed through techniques mentioned in sections 2.2.2 and 2.2.3.

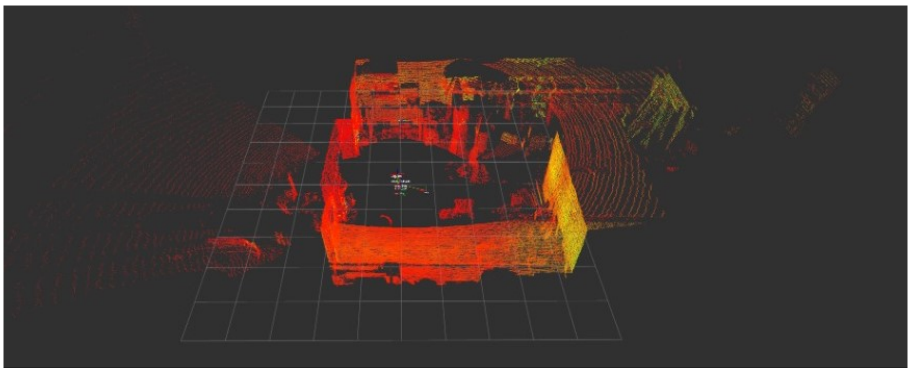

Figure 13: Point cloud visualization of LIDAR data in rviz

2.2.2 Filtering

Processing the hundreds of thousands of data points in a point cloud can be a very computationally heavy task. It is therefore very often applied pre-processing methods to the point cloud to filter out data points that are irrelevant to the application. If the point cloud is used to detect obstacles ground filtering is often applied. Using the height of the LIDAR, the ground plane is estimated and calculated using nearby data points. Random sample consensus (RANSAC) is an algorithm that is often used for ground filtering. Given a mathematical model with parameters

$$y = f(x; a) \qquad where \quad a = (a_1, a_2, ..., a_n)$$

it iterates over a random set of subsamples to find the best model parameters. This is done in five steps.

1. Estimate the model parameters from a randomly sampled subset of data points
2. Determine the set of inliers to be the data points within a distance to the model
3. If this is the largest set of inliners, store the model parameters
4. If the model precision is over some threshold value, stop. If not, repeat
5. Stop after N iterations

For a ground vehicle the sample is selected using the relative position of the vehicle to the ground. For an autonomous application at sea we can also apply land filtering by using sea charts to filter out buildings and other objects on land that are irrelevant to the application. These maps can also be made dynamically using simultaneous localization and mapping (SLAM) methods.

Two popular methods for down-sampling a point cloud is also the grid-based and the minimal distance methods. In the minimal distance method the data points are removed so that no data point is closer to another data point than the minimum distance specified. In the grid method a grid structure is created and a representative data point is selected from within that grid. Each grid cell is called a voxel and will only keep one representative point which can be selected in different ways. This point can be the center of all the points located within the voxel or the point closest to the center.

2.2.3 Clustering

Because of the high resolution most objects are reflected and registered as several LIDAR data points. Different clustering techniques are applied to organize the points that correspond to the same object into clusters like those in Figure 14. These clusters are generally used as a detection in a tracking pipeline.

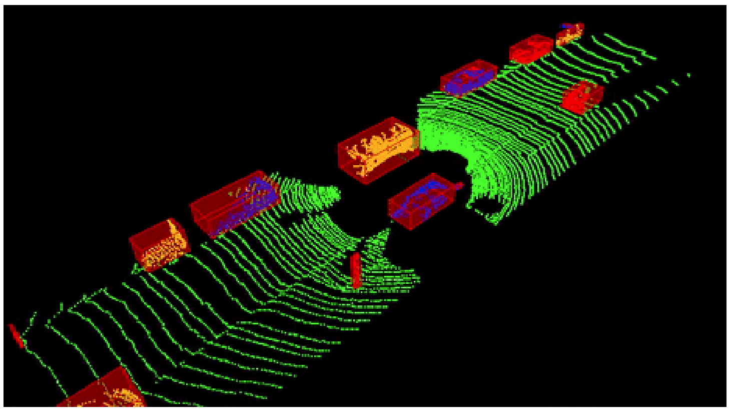

Figure 14: LIDAR Clusters [16]

The goal of such a clustering method, similar to filtering methods, is to reduce the overall processing time and complexity when processing the data and extract objects of interest. A simple Euclidean data clustering approach can be implemented by splitting the LIDAR operating area into subdivisions similar to an voxel grid filter. We can however make use of nearest neighbours and kd-trees to make a more precise method.

1. Create a Kd-tree representation for the input cloud dataset P
2. Set up an empty list of clusters C and a queue of the points that needs to be checked Q
3. For every point $p_i \in P$, perform the steps:
    - add p_i to the queue Q
    - For every point $p_i \in Q$:
        - Search for the set P_i^k of point neighbours of p_i in a sphere with radius R
        - For every neighbor $p_i^k \in P_i^k$, check if the point has already been processed, if not add it to Q
    - When the list of all points in Q have been processed, add Q to the list of clusters C and reset Q to an empty list
4. The algorithm terminates when all points $p_i \in P$ have been processed and are now part of the list of point clusters C

Table 1: The steps in an Euclidean clustering method [36].

2.3 Relative positioning between sensors and detections

To be able to judge where different sensors are relative to each other one needs to describe their placement. Such transformations are done by homogeneous transformation matrices that describe the position and orientation between different coordinate frames. Such a transformation matrix is shown in Equation 2.3 and can also be used to give the relative position of detections.

$$T_b^a = \begin{pmatrix} R_b^a & T_{ab}^a \\ 0\,0\,0 & 1 \end{pmatrix} \qquad T_{ab}^a = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Here R_b^a describes the rotation and T_{ab}^a describes the translation of point b relative to point a. In ROS, which is the framework used in the test scenarios described in Section 4, these transformations are defined as rotations in roll, pitch, yaw and translation in x, y and z-direction. The rotation matrix R_b^a is constructed from three smaller matrices each representing a rotation around an axis.

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} R_y(\omega) = \begin{bmatrix} \cos\omega & 0 & \sin\omega \\ 0 & 1 & 0 \\ -\sin\omega & 0 & \cos\omega \end{bmatrix} R_z(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Multiplying these together we get the rotation matrix $R_b^a(\phi, \omega, \psi)$.

$$R_{xyz}(\phi, \omega, \psi) = \begin{bmatrix} \cos\psi\cos\omega & -\sin\psi\cos\phi + \cos\psi\sin\omega\sin\phi & \sin\psi\sin\phi + \cos\psi\sin\omega\cos\phi \\ \sin\psi\cos\omega & \cos\psi\cos\phi + \sin\psi\sin\omega\sin\phi & -\cos\psi\sin\phi + \sin\psi + \sin\omega\cos\phi \\ -\sin\omega & \cos\omega\sin\phi & \cos\omega\cos\phi \end{bmatrix}$$

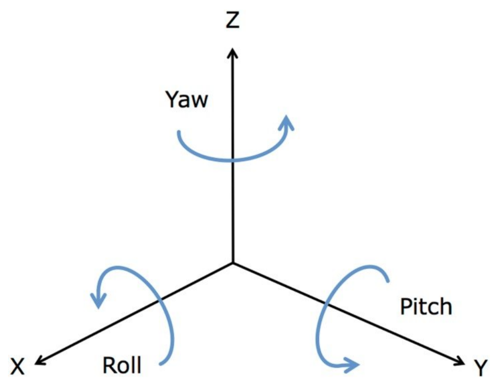

Figure 15: Transformation between two coordinate systems [15].

2.4 Target tracking

The goal of a target tracker is to estimate the motion of objects and to confirm or confute tentative hypotheses about target presence using sensor measurements. How a system can detect objects using sensors like a camera and a LIDAR has already been described and this Section will describe methods for fusing and interpreting these measurements. There are two schools of sensor fusion, namely track level fusion and measurement level fusion. Track level fusion runs a tracker for each sensor and generates multiple tracks, performing fusion on these tracks to find a combined solution to the estimation problem. Meanwhile measurement level fusion will use a single tracker that fuses the measurements together into a single track. In this Section and this thesis we will focus on the latter. Most of the information in this Section is from "Fundamentals of Sensor Fusion [6]".

The symbols listed below are used to describe different elements within target tracking.

x	Kinematic state	π	Transition probability matrix
P	State covariance matrix	μ	Mode probability
F	State transition matrix	P_D	Detection probability
z	Measurement	λ	Clutter intensity
R	Measurement covariance matrix	a	Association hypothesis
H	Measurement matrix	g	Validation gate scaling parameter

Table 2: The notation used in this report relating to target tracking.

$\hat{}$	Estimate	t	Track index
$\dot{}$	Time derivative	s	Kinematic mode
k	Time step	$k\|k-1$	Conditional on previous time

Table 3: The sub- and superscripts used in this report relating to target tracking.

2.4.1 Kalman filter (KF) to extended Kalman filter (EKF)

The Kalman filter provides a closed-form solution to the estimation problem if the model and its likelihood is Gaussian and linear. The Kalman filter is the base for most modern estimation methods and can be viewed as an iterative two-step process. We perform a prediction using a mathematical model of the system before we perform an update step, updating and correcting the predictions using measurements. The steps in the Kalman filter can be shown in Figure 16.

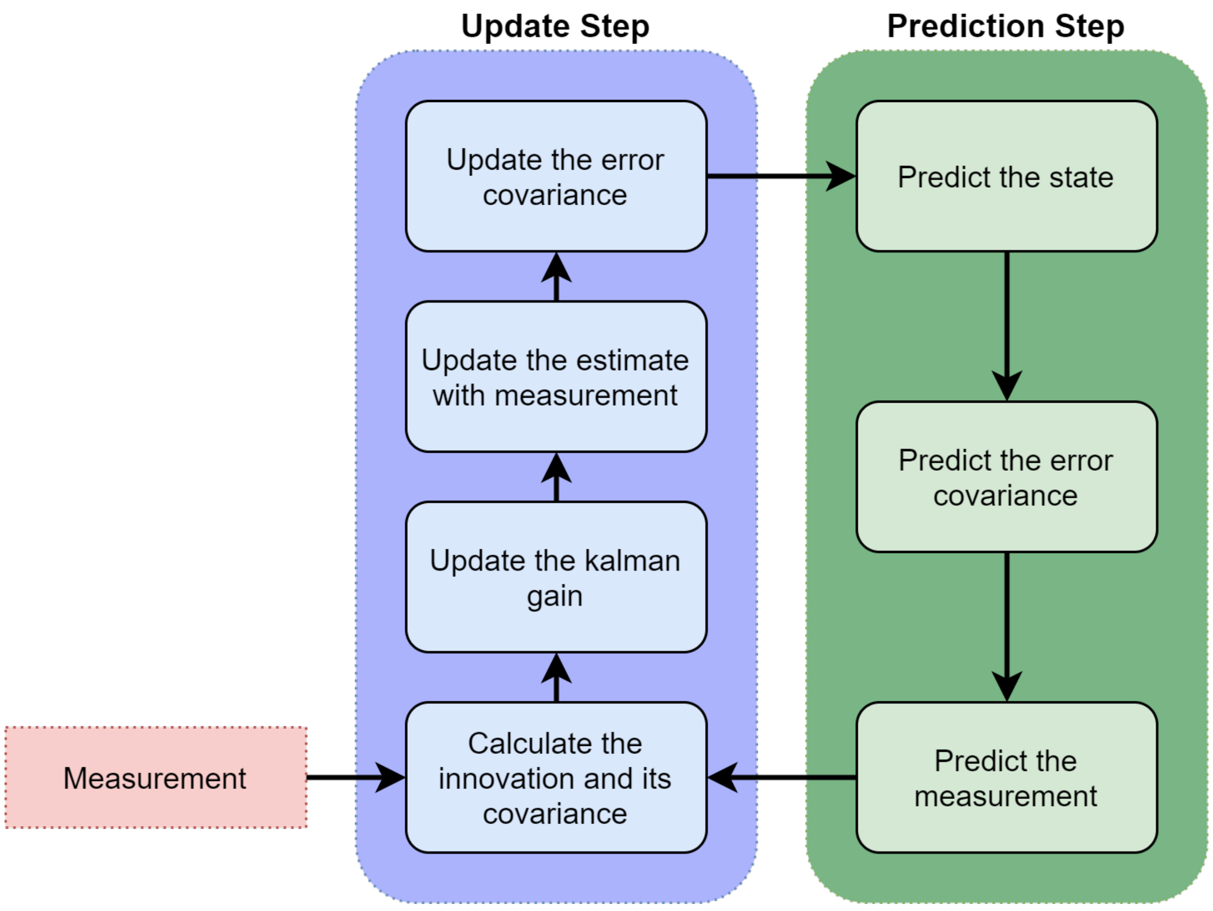
Update Step
Prediction Step
Update the error covariance
Predict the state
Update the estimate with measurement
Predict the error covariance
Update the kalman gain
Measurement
Calculate the innovation and its covariance
Predict the measurement

Figure 16: The steps in a Kalman filter

The Kalman gain is a parameter that decides how much we should weigh the predicted and the measured value. This is calculated using the uncertainty of the prediction and the error in the measurements. The equations used in the Kalman filter are shown in Figure 17. Note that the state transition matrix F and the measurement matrix H are linear.

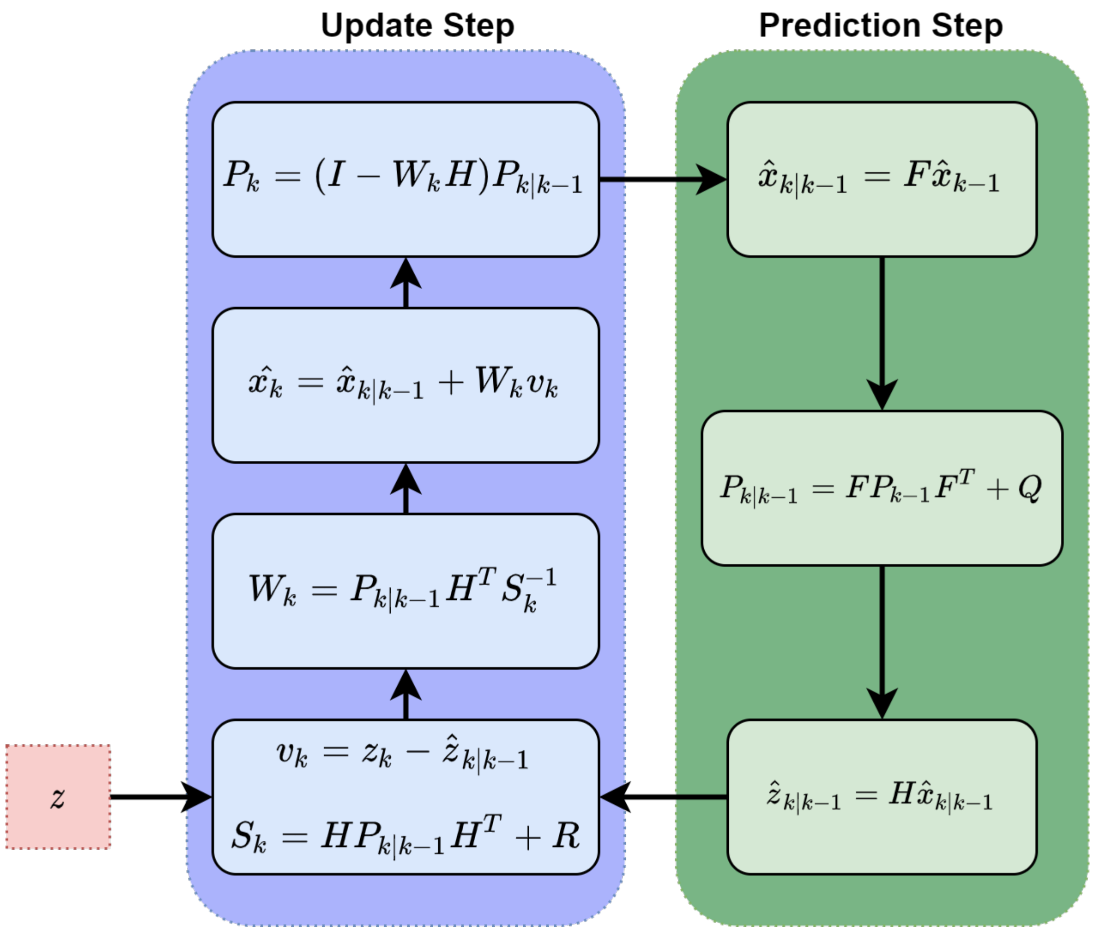
Update Step
Prediction Step
$P_k = (I - W_k H) P_{k|k-1}$
$\hat{x}_k = \hat{x}_{k|k-1} + W_k v_k$
$W_k = P_{k|k-1} H^T S_k^{-1}$
$v_k = z_k - \hat{z}_{k|k-1}$
$S_k = H P_{k|k-1} H^T + R$
z
$\hat{x}_{k|k-1} = F \hat{x}_{k-1}$
$P_{k|k-1} = F P_{k-1} F^T + Q$
$\hat{z}_{k|k-1} = H \hat{x}_{k|k-1}$

Figure 17: The steps in a Kalman filter using mathematical notation

Most real world problems involve non linear functions which break the assumptions for the Kalman filter. As can be seen in sections 2.1.4 and 2.2.1 the position of the objects relative to the vessel is often given as a bearing angle which in turn brings trigonometric functions and non-linearities into the picture. One can convert the measurements into a Cartesian coordinate system before using the tracker, but even if the measurement model is linear the system model is often non-linear. Feeding a Gaussian into a non-linear function will always cause the output distribution to be non-Gaussian. In order to solve this problem the extended Kalman filter applies linearisation by means of the first order derivative, giving a linear approximation to the non-linear functions by using the tangent around the mean. This process is visualized in Figure 18 where we have added two linearization steps marked in orange. The prediction of the state and measurement means, marked with red, are now non-linear functions. Because of this linearization we can achieve a Gaussian distribution while having a non-linear state transition matrix and measurement matrix.

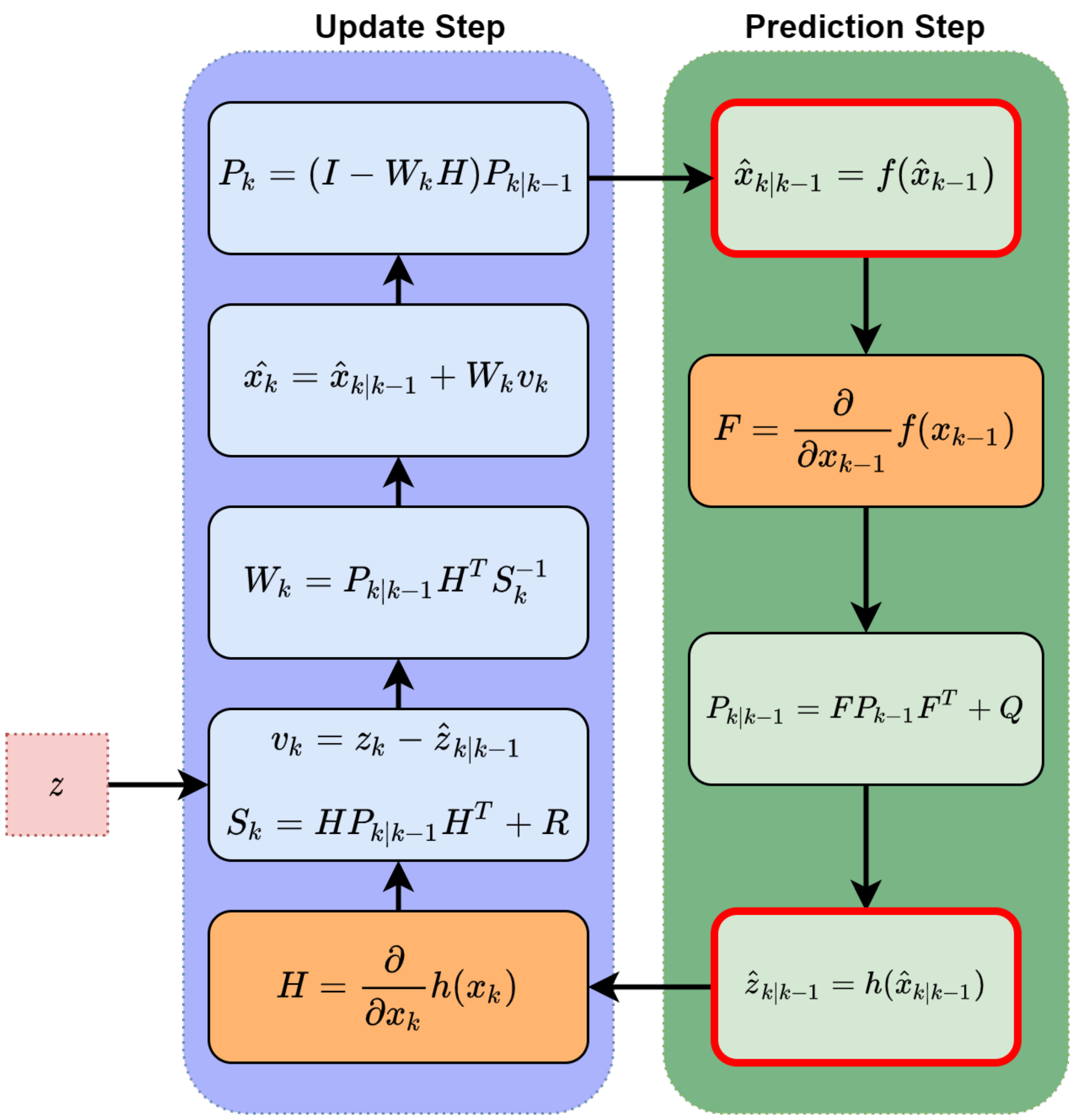
Update Step
Prediction Step
$P_k = (I - W_k H) P_{k|k-1}$
$\hat{x}_k = \hat{x}_{k|k-1} + W_k v_k$
$W_k = P_{k|k-1} H^T S_k^{-1}$
$v_k = z_k - \hat{z}_{k|k-1}$
$S_k = H P_{k|k-1} H^T + R$
$H = \frac{\partial}{\partial x_k} h(x_k)$
z
$\hat{x}_{k|k-1} = f(\hat{x}_{k-1})$
$F = \frac{\partial}{\partial x_{k-1}} f(x_{k-1})$
$P_{k|k-1} = F P_{k-1} F^T + Q$
$\hat{z}_{k|k-1} = h(\hat{x}_{k|k-1})$

Figure 18: The steps in an extended Kalman filter

2.4.2 Interacting multiple models (IMM)

In a maritime setting, a target tracking algorithm needs to be versatile and dynamic enough to be able to detect and track everything from large ships to smaller debris and buoys. Targets of the same size have different dynamic properties. A fishing boat and a recreational boat of the same size are expected to act very differently, however looking at these though the eyes of a LIDAR or a RADAR, it is hard to securely apply either of these two models to the target. Figure 19 shows an example of this. While the LIDAR point clouds are relatively similar, we humans would know that the vessel on the left side of the top picture would behave very different from the vessel on the right.

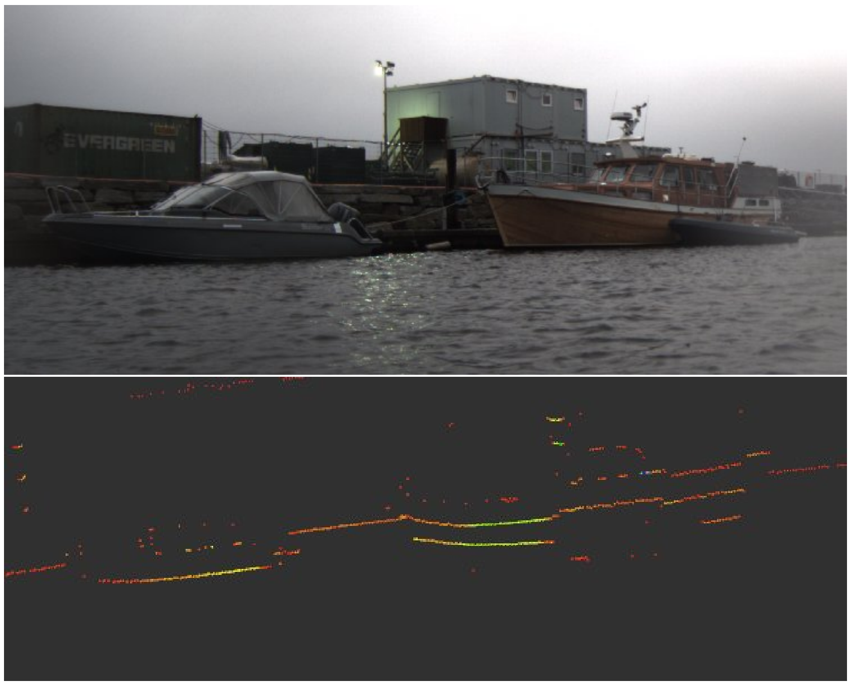

Figure 19: Two vessels that behave very differently seen through a LIDAR

Interactive multiple models is a filtering technique that performs predictions of a set of models in parallel where each model represent different dynamical behaviour. The method then applies probability theory to weigh the output from each model and mixture reduction to merge the weighted filter estimates into one common state estimate. It was presented by Henk A. P. Blom and Yaakov Bar-Shalom in 1988 in "The interacting multiple model algorithm for systems with markovian switching coefficients [5]."

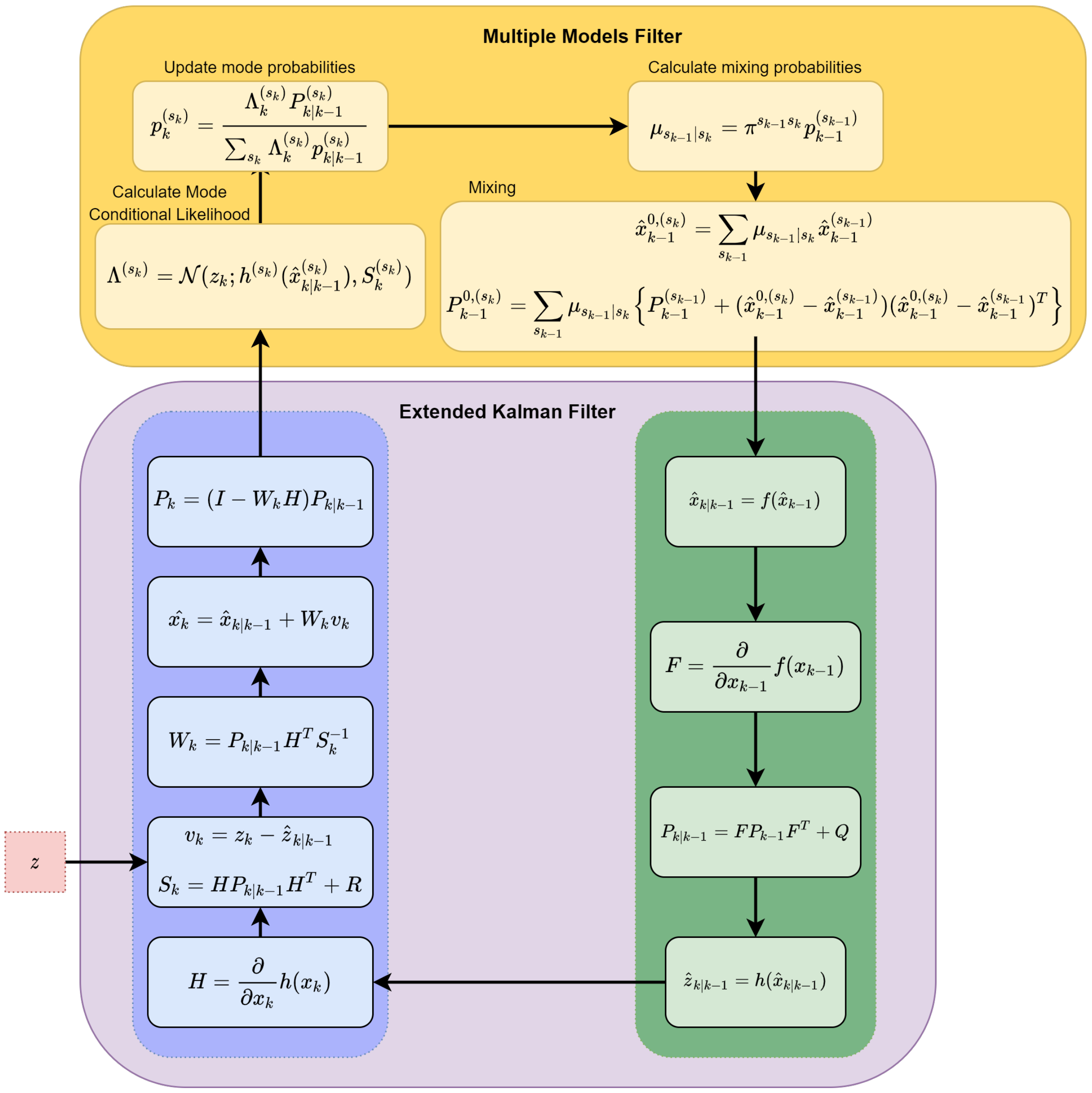
Multiple Models Filter
Update mode probabilities
$p_k^{(s_k)} = \frac{\Lambda_k^{(s_k)} P_{k|k-1}^{(s_k)}}{\sum_{s_k} \Lambda_k^{(s_k)} p_{k|k-1}^{(s_k)}}$
Calculate mixing probabilities
$\mu_{s_{k-1}|s_k} = \pi^{s_{k-1}s_k} p_{k-1}^{(s_{k-1})}$
Calculate Mode Conditional Likelihood
$\Lambda^{(s_k)} = \mathcal{N}(z_k; h^{(s_k)}(\hat{x}_{k|k-1}^{(s_k)}), S_k^{(s_k)})$
Mixing
$\hat{x}_{k-1}^{0,(s_k)} = \sum_{s_{k-1}} \mu_{s_{k-1}|s_k} \hat{x}_{k-1}^{(s_{k-1})}$
$P_{k-1}^{0,(s_k)} = \sum_{s_{k-1}} \mu_{s_{k-1}|s_k} \left\{ P_{k-1}^{(s_{k-1})} + (\hat{x}_{k-1}^{0,(s_k)} - \hat{x}_{k-1}^{(s_{k-1})})(\hat{x}_{k-1}^{0,(s_k)} - \hat{x}_{k-1}^{(s_{k-1})})^T \right\}$
Extended Kalman Filter
$P_k = (I - W_k H) P_{k|k-1}$
$\hat{x}_k = \hat{x}_{k|k-1} + W_k v_k$
$W_k = P_{k|k-1} H^T S_k^{-1}$
$v_k = z_k - \hat{z}_{k|k-1}$
$S_k = H P_{k|k-1} H^T + R$
$H = \frac{\partial}{\partial x_k} h(x_k)$
z
$\hat{x}_{k|k-1} = f(\hat{x}_{k-1})$
$F = \frac{\partial}{\partial x_{k-1}} f(x_{k-1})$
$P_{k|k-1} = F P_{k-1} F^T + Q$
$\hat{z}_{k|k-1} = h(\hat{x}_{k|k-1})$

Figure 20: The prediction steps in multiple model filtering.

Figure 20 shows the steps in a multiple model filter as an extension of the extended Kalman filter. In reality it is more like Figure 21 where we have a set of M filters that all do predictions on separate models for the target. The mode conditional likelihood for each mode represents the precision of the estimate. This is found using the measurement, the innovation and the innovation covariance. This is used to update the mode probabilities with the new information gained through the measurements, which in turn is used to calculate mixing probabilities for the Gaussians.

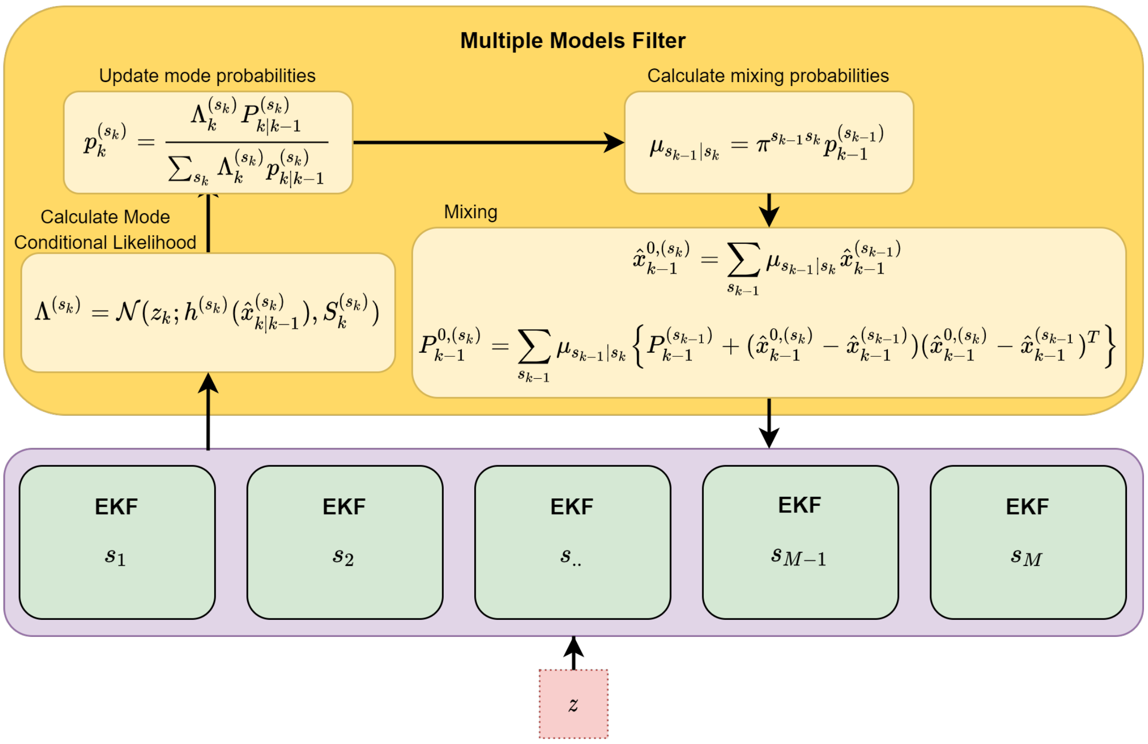
Multiple Models Filter
Update mode probabilities
$p_k^{(s_k)} = \frac{\Lambda_k^{(s_k)} P_{k|k-1}^{(s_k)}}{\sum_{s_k} \Lambda_k^{(s_k)} p_{k|k-1}^{(s_k)}}$
Calculate mixing probabilities
$\mu_{s_{k-1}|s_k} = \pi^{s_{k-1}s_k} p_{k-1}^{(s_{k-1})}$
Calculate Mode Conditional Likelihood
$\Lambda^{(s_k)} = \mathcal{N}(z_k; h^{(s_k)}(\hat{x}_{k|k-1}^{(s_k)}), S_k^{(s_k)})$
Mixing
$\hat{x}_{k-1}^{0,(s_k)} = \sum_{s_{k-1}} \mu_{s_{k-1}|s_k} \hat{x}_{k-1}^{(s_{k-1})}$
$P_{k-1}^{0,(s_k)} = \sum_{s_{k-1}} \mu_{s_{k-1}|s_k} \left\{ P_{k-1}^{(s_{k-1})} + (\hat{x}_{k-1}^{0,(s_k)} - \hat{x}_{k-1}^{(s_{k-1})})(\hat{x}_{k-1}^{0,(s_k)} - \hat{x}_{k-1}^{(s_{k-1})})^T \right\}$
EKF s_1
EKF s_2
EKF $s_{..}$
EKF s_{M-1}
EKF s_M
z

Figure 21: Multiple model filtering.

As each filter outputs a Gaussian we reduce them into a single estimate by means of the mixing probabilities and Gaussian mixture. Gaussian mixture combines several distributions into one single distribution that is the best representative for all modes. For clarity Figure 22 shows a mixture of the most commonly used univariate Gaussians. In this thesis we will do tracking in a Cartesian coordinate system which requires the use of the multivariate Gaussian. An example of this reduction is shown in Figure 23. In both Figures the probability mass is centered as the mean of the two original distributions.

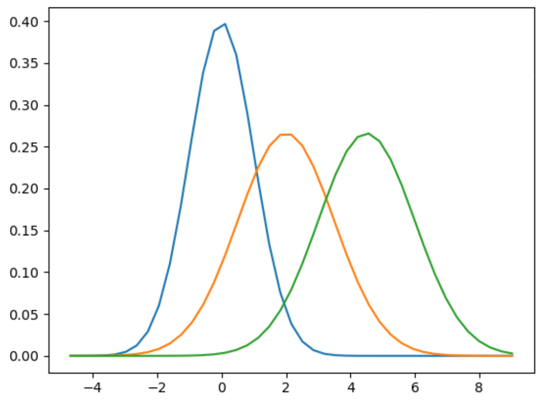

Figure 22: Univariate Gaussian mixture.

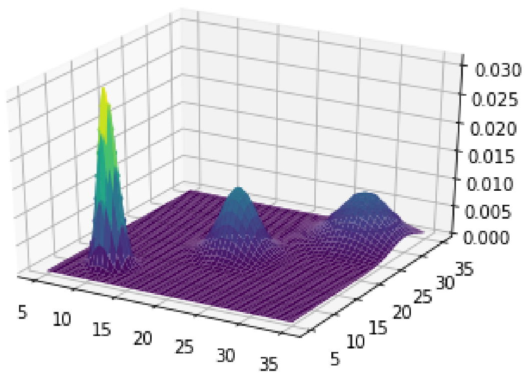

Figure 23: Multivariate Gaussian mixture[26].

2.4.3 Probabilistic Data Association (PDA)

In a real world scenario with sensors like LIDAR and RADAR there will be misdetections and times when multiple sensor measurements can be attributed to the same target. There will also be cases when the sensor does not register the target due to occlusion or unstable sensor processing methods. The probabilistic data association algorithm solves these problems by performing the update step using different data association hypotheses and doing mixture reduction on the results to find the innovation mean and covariance. Like in the multiple models filter we also here apply probability theory to weight the hypotheses.

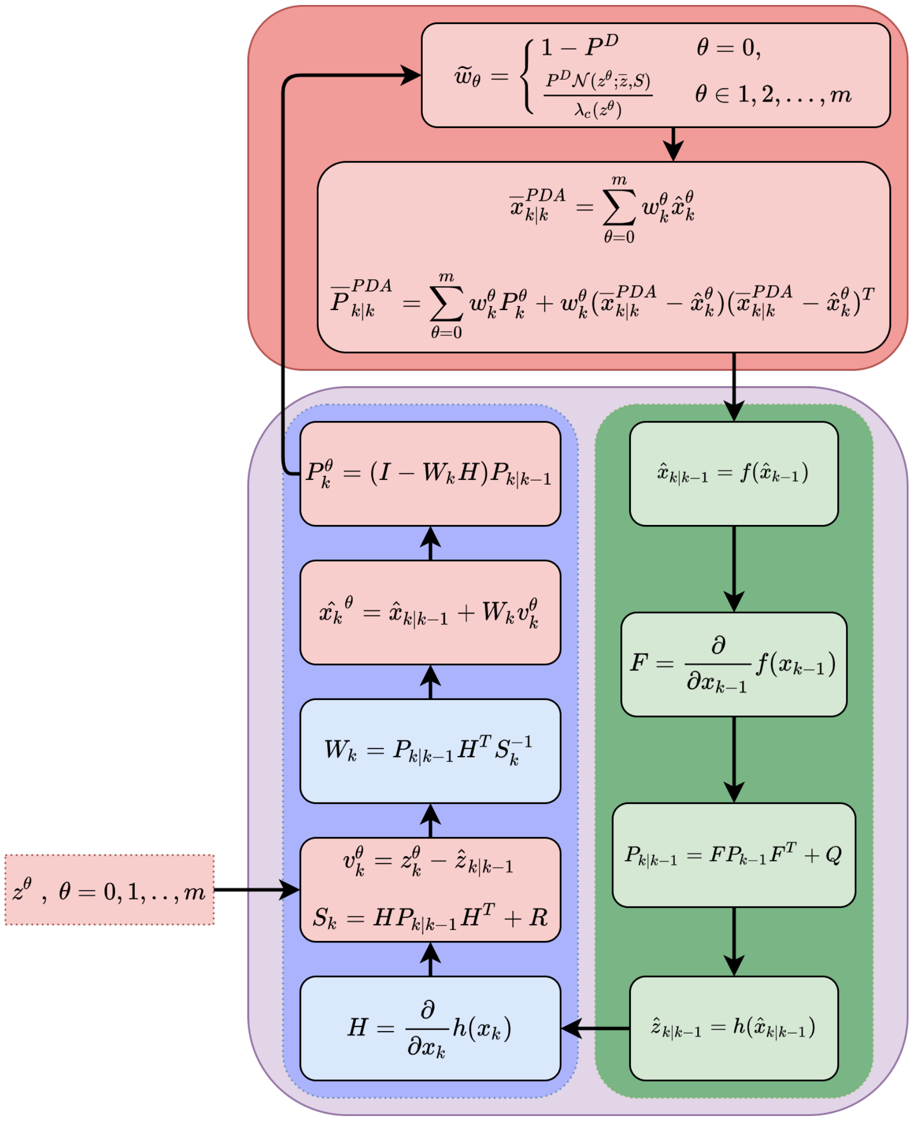
$\widetilde{w}_\theta = \begin{cases} 1 - P^D & \theta = 0, \\ \frac{P^D \mathcal{N}(z^\theta; \bar{z}, S)}{\lambda_c(z^\theta)} & \theta \in 1, 2, \ldots, m \end{cases}$
$\overline{x}^{PDA}_{k|k} = \sum_{\theta=0}^{m} w_k^\theta \hat{x}_k^\theta$
$\overline{P}^{PDA}_{k|k} = \sum_{\theta=0}^{m} w_k^\theta P_k^\theta + w_k^\theta (\overline{x}^{PDA}_{k|k} - \hat{x}_k^\theta)(\overline{x}^{PDA}_{k|k} - \hat{x}_k^\theta)^T$
$P_k^\theta = (I - W_k H) P_{k|k-1}$
$\hat{x_k}^\theta = \hat{x}_{k|k-1} + W_k v_k^\theta$
$W_k = P_{k|k-1} H^T S_k^{-1}$
$v_k^\theta = z_k^\theta - \hat{z}_{k|k-1}$
$S_k = H P_{k|k-1} H^T + R$
$H = \frac{\partial}{\partial x_k} h(x_k)$
$z^\theta \ , \ \theta = 0, 1, .., m$
$\hat{x}_{k|k-1} = f(\hat{x}_{k-1})$
$F = \frac{\partial}{\partial x_{k-1}} f(x_{k-1})$
$P_{k|k-1} = F P_{k-1} F^T + Q$
$\hat{z}_{k|k-1} = h(\hat{x}_{k|k-1})$

Figure 24: Probabilistic Data Association Filtering.

To account for clutter in the measurements the PDA applies a clutter model. The poisson distribution is often used to model this. It is shown in equation 6 where P_{FA} represents the probability of a false detection and N represents the total number of resolution cells. P_{FA} is a sensor specific constant.

$$\mu(\phi) = e^{-\lambda}\frac{\lambda^{\phi}}{\phi!} \qquad where \quad \lambda = NP_{FA} \tag{6}$$

We also apply a model for misdetections by means of a Bernoulli random variable. This is applicable if the detection events are independent. It uses the detection probability P_D as a parameter to to model the detection event δ.

$$p(\delta) = \begin{cases} P_D & if\ \delta = 1 \\ 1 - P_D & if\ \delta = 0 \end{cases}$$

While it is not shown as a part of Figure 24, we also need to account for cases when there are no detections from the target. In this case the innovations and innovation covariance is given as in equation 2.4.3.

$$p(x_k|a_k, Z_{1:k}) \propto \begin{cases} p_{k|k-1}(x_k) & if\ a_k = 0 \\ f_z(z_k^{a^k}|x_k)p_{k|k-1}(x_k) & if\ a_k > 0 \end{cases}$$

2.4.4 PDA to JIPDA

The EKF, IMM and PDA filtering methods are easily applied under the assumption that all the measurements originate from the target or as clutter. Together they are great modelling and estimation tools, however they always assume that the target is existing and that estimation is needed. It does not have any mechanisms in place to handle track initialization or track loss. Given a validation gate around the track we can filter measurements that are likely to originate from the target, but we need a method in place to judge if a set of sequence of detections should initiate a new track. We also need a mechanism for terminating tracks when the target moves outside the surveillance region or dissolves. Assigning measurements to targets increase complexity, and in a real world scenario we need to be able to track multiple targets at once while taking into considerations clutter and a large set of measurements.
A way of initializing tracks is the M/N-logic. Here a tentative track is initialized if we receive two consecutive measurements in close vicinity to each other. The tentative track is then run through the PDA for N time steps, after which a decision is made. If we receive more then M measurements within those steps the track is initiated and if we received less than M measurements the preliminary track is terminated. The problem with such logic is that track initialization takes up N time steps. The integrated probabilistic data association (IPDA) algorithm is an extension to the PDA which in addition to state estimation also estimates the probability of target existence. This probability can in turn be used to initiate or terminate tracks. The IPDA consists of five steps:

1. Existence Prediction
2. State Prediction
3. Association Weights
4. State Update
5. Existence Update

Figure 25 shows the workflow in an IPDA algorithm. Here the $\mathscr{L}$ in the existence prediction corresponds to the following:

$$\mathscr{L}_k = 1 - P_D + \frac{P_D}{\lambda} \sum_{a_k=1}^{m_k} \mathcal{N}(z_k^{a_k}; \hat{z}_{k|k-1}, S_k)$$

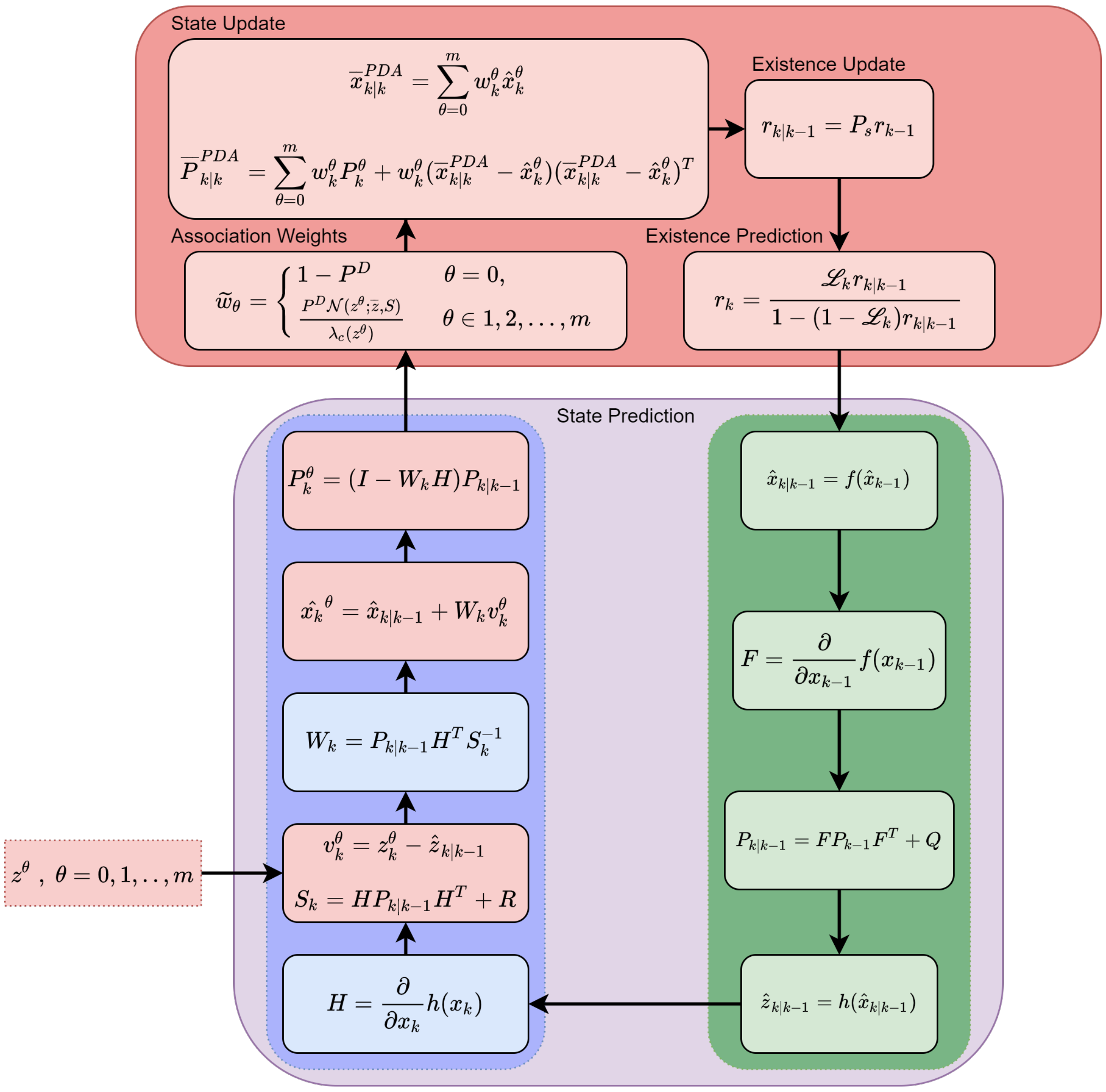
State Update
$\bar{x}^{PDA}_{k|k} = \sum_{\theta=0}^{m} w_k^\theta \hat{x}_k^\theta$
$\bar{P}^{PDA}_{k|k} = \sum_{\theta=0}^{m} w_k^\theta P_k^\theta + w_k^\theta(\bar{x}^{PDA}_{k|k} - \hat{x}_k^\theta)(\bar{x}^{PDA}_{k|k} - \hat{x}_k^\theta)^T$
Existence Update
$r_{k|k-1} = P_s r_{k-1}$
Association Weights
$\tilde{w}_\theta = \begin{cases} 1 - P^D & \theta = 0, \\ \frac{P^D \mathcal{N}(z^\theta;\bar{z},S)}{\lambda_c(z^\theta)} & \theta \in 1, 2, \ldots, m \end{cases}$
Existence Prediction
$r_k = \frac{\mathscr{L}_k r_{k|k-1}}{1 - (1 - \mathscr{L}_k) r_{k|k-1}}$
State Prediction
$P_k^\theta = (I - W_k H) P_{k|k-1}$
$\hat{x}_k^\theta = \hat{x}_{k|k-1} + W_k v_k^\theta$
$W_k = P_{k|k-1} H^T S_k^{-1}$
$v_k^\theta = z_k^\theta - \hat{z}_{k|k-1}$
$S_k = H P_{k|k-1} H^T + R$
$H = \frac{\partial}{\partial x_k} h(x_k)$
$z^\theta\ ,\ \theta = 0, 1, .., m$
$\hat{x}_{k|k-1} = f(\hat{x}_{k-1})$
$F = \frac{\partial}{\partial x_{k-1}} f(x_{k-1})$
$P_{k|k-1} = F P_{k-1} F^T + Q$
$\hat{z}_{k|k-1} = h(\hat{x}_{k|k-1})$

Figure 25: Integrated Probabilistic Data Association Filtering.

While the IPDA solves the problem of track initialization it does not solve the problem of multiple targets and assigning measurements. The Joint Probabilistic Data Association (JPDA) algorithm is the multi-target extension of the PDA which takes care of joint data association hypotheses. The IPDA in itself can be a sufficient tracker when the tracks and measurements are sufficiently far apart. A problem does however arise when the objects that are tracked are within close proximity to each other and measurements fall within the validation gates for several targets. This is where the JPDA comes into play. Data association hypotheses are made from a cluster of nearby tracks and measurements, and the probability for each hypothesis is calculated. The output from the filters weighted with the association probability are merged using mixture reduction into a single Gaussian for each track. The workflow of the JPDA can be seen in Figure 26.

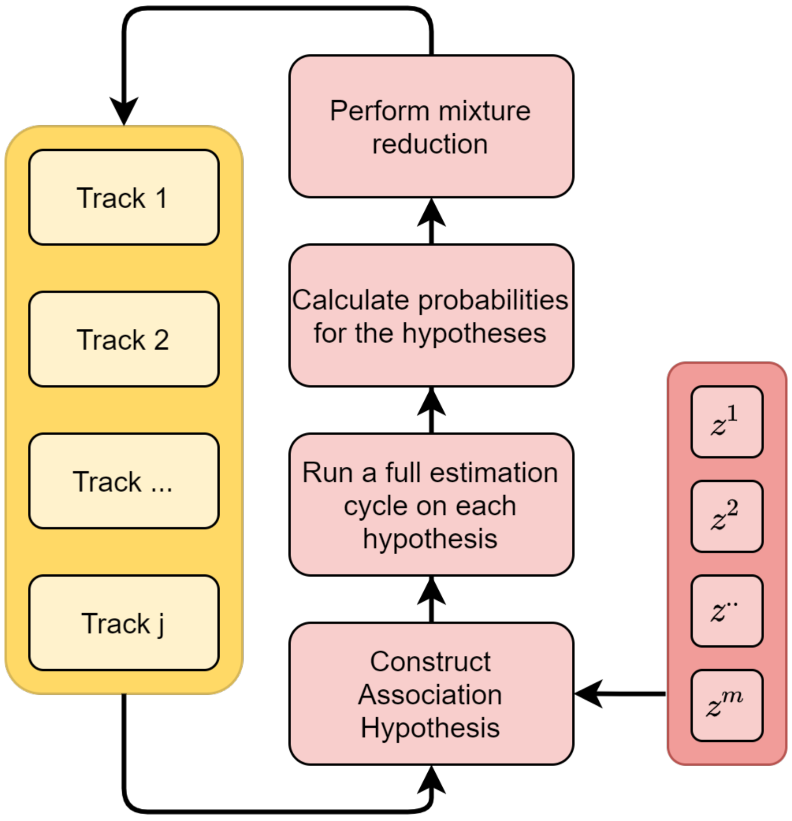
Perform mixture reduction
Calculate probabilities for the hypotheses
Run a full estimation cycle on each hypothesis
Construct Association Hypothesis
Track 1
Track 2
Track ...
Track j
z^1
z^2
$z^{..}$
z^m

Figure 26: Joint Probabilistic Data Association Filtering.

Constructing these association hypotheses and performing a full estimation cycle on each one of them is computationally demanding. Clustering of nearby tracks is one way of reducing the complexity of the problem. This can be done by a single linkage track clustering algorithm described in "Single-link and complete-link clustering [10]." Depending on the amount of measurements the amount of association hypotheses can still reach the hundreds. Using the auction algorithm described in "A distributed relaxation method for the assignment problem [4]" we can select only the few hypotheses with the most probability mass.

Merging the data association hypotheses in the JPDA and the estimation of probability of target existence from the IPDA we get the JIPDA. As can be seen in Figure 26 we use the IPDA when performing the full estimation cycle on each hypothesis.

2.4.5 Visibility state modelling

Being able to estimate the visibility of a target can make the tracker less susceptible to track-loss, a problem extensively described in "Hybrid state formulation and verification on maritime radar benchmark data [9]". For the IPDA two models for target existence and observability are often used, namely the markov chain one (MC1) and markov chain two (MC2). The MC1 assumes that a target exists and have a certain detection probability, while the MC2 is an extension of MC1 that assumes that a target exists and can be unobservable. In MC2 we model both the kinematic state, the detection probability and the visibility state in a hybrid state. While the kinematic state is continuous the visibility state v is a discrete state.

$$v = \begin{cases} 1 & \text{if the target is visible} \\ 0 & \text{otherwise} \end{cases}$$

For the birth intensity we assume an unknown target intensity instead of the constant birth intensity. We assume it to be stationary on the form

$$v_{k|k-1}(y) = b_{\Omega}(Hx)\mathcal{N}(H^{*}x; 0, P_v)Pr\{s_{k-1}\}Pr\{v_{k-1}\} = b_{\Omega}(Hx)\mathcal{N}(H^{*}x; 0, P_v)\mu^{0s}o^{0v}$$

The probabilities $Pr\{s_k\}$ and $Pr\{v_k\}$ can be organized in matrices representing a Markov chain. The parameter b_{Ω} quantifies the overall rate of the unknown target intensity. While the detection probability previously has been a constant value it is now dependant on the visibility state of the target.

$$P_D(v) = \begin{cases} P_D & \text{if v = 1} \\ 0 & \text{if v = 0} \end{cases}$$

The prior is given as

$$f^{t}_{k-1}(y) = f^{ts}_{k-1}(s)\mu_{k-1}o_{k-1}$$

The mode probabilities, visibility probabilities and predicted kinematic states are

$$\mu_{k|k-1} = \sum \pi^{\bar{s}s}\mu^{t\bar{s}}_{k-1}$$

$$\eta_{k|k-1} = \omega^{01}(1-\eta_{k-1}) + \omega^{11} n_{k-1}$$
$$f_{k|k-1}(x) = \int f_x(x|\overline{x}) f_{k-1}(\overline{x}) d\overline{x}$$

For the JIPDA-implementation with visibility modelling the measurement update is done similar to Figure 26. We do however now have four cases for the posterior, depending on whether the measurement corresponds to an empty track, a new target, a misdetection or a detection. For the case with a **new target** we have the following association weights, existence probabilities, updated model probabilities and state pdf

$$\omega_k^{tj} = \lambda + b\eta^0$$
$$r_k^{tj} = \frac{b\eta^0}{\lambda + b\eta^0}$$
$$\mu_k^{tsj} = \mu^{0s}$$
$$\eta_k^{tsj} = \eta^{0s}$$
$$f_k^{tsj}(x) = \mathcal{N}(x; \hat{x}_0^s, P_0^S)$$

In the case of a **misdetection** we have

$$\omega_k^{t0} = 1 - r_{k|k-1}^t + r_{k|k-1}^t(1 - \eta_{k|k-1}^t P_d)$$
$$r_k^{t0} = \frac{r_{k|k-1}^t (1 - \eta_{k|k-1}^t P_D}{1 - \eta_{k|k-1} + \eta_{k|k-1}(1 - \eta_{k|k-1}^t P_D)}$$
$$\mu_k^{ts0} = \mu_{k|k-1}^{ts0}$$
$$\eta_k^{t0} = \frac{(1-P_D)\eta_{k|k-1}^t}{1 - P_D \eta_{k|k-1}^t}$$
$$f_k^{ts0}(x) = f_{k|k-1}^{ts}(x)$$

And in the **detection** case these become

$$\omega_k^{tj} = P_D r_{k|k-1}^t \eta_{k|k-1}^t \sum \mu_{k|k-1}^{ts} l_k^{t\overline{s}j}$$
$$r_k^{tj} = 1$$
$$\mu_k^{tsj} = \mu_{k|k-1}^{ts0} l_k^{tsj} / \sum \mu_{k|k-1}^{t\overline{s}} l^{t\overline{s}j}$$
$$\eta_k^{tj} = 1$$
$$f_k^{tsj}(x) = f_z^s(z_k^j|x) f_{k|k-1}^{ts}(x) / l_k^{tsj}$$
$$where \qquad l_k^{tsj} = \int f_z^s(z_k^j|\overline{x}) f_{k|k-1}^{ts}(\overline{x}) d\overline{x}$$

2.4.6 Multi sensor tracking

The core of sensor fusion is reducing the uncertainty of data from separate sources by interpreting them together. This thesis is about fusing measurements in the Cartesian coordinate system from a LIDAR as well as detections in the form of bearing-only measurements from cameras. As mentioned in the introduction to this Section the methods discussed previously are focused on measurement level fusion. Converting the measurements using different measurement functions we continuously update our filters and decrease the uncertainty in our estimates.

As mentioned in subsection 2.2.1 the LIDAR points originates from range-bearing measurements but is converted into the Cartesian coordinate plane using equation 5. In ROS the standard is to output LIDAR points in Cartesian coordinates. Because of the transformation is already done, the measurement function for the LIDAR becomes

$$f_k(x_k) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_{x_k} \\ p_{y_k} \\ v_{x_k} \\ v_{y_k} \end{bmatrix}$$

For the camera we can calculate the bearing and elevation angle for a bounding box detection by knowing the field of view and the pixel resolution of a camera. As the tracking is done in the 2D plane the elevation angle is of no use and only the bearing angle is considered for tracking. This gives the following measurement function for a camera

$$f_k(x_k) = arctan\Big(\frac{p_{y_k}}{p_{x_k}}\Big)$$

It should be noted that the observability of a target using bearing only measurements is reduced if the target maneuvers more than the sensor platform. A system is defined as observable if all the states can be estimated directly from the measurements. For the bearing only tracking problem the states are only partially observable since the range state is observable only after the observer performs a manoeuvre.

In "Observability analysis of advanced guidance laws with bearing-only measurement [31]" it is shown that a constant velocity target is not observable for a stationary sensor. Given the measurement function for a bearing only tracker

$$M(t) = x(t)y(t)$$

where

$$x(t) = \begin{bmatrix} r_x(t_0) \\ r_y(t_0) \\ v_x(t_0) \\ v_y(t_0) \end{bmatrix}$$

$$M(t) = \begin{bmatrix} -sin\lambda(t) \\ cos\lambda(t) \\ -\Delta t sin\lambda(t) \\ \Delta t cos\lambda(t) \end{bmatrix}$$

$$y(t) = \int_{t_0}^{t} (t-\tau)[a_{m_y}(\tau)cos\lambda(t) - a_{m_x}(\tau)sin\lambda(t)]d\tau$$

And the observability matrix

$$A(t) = \begin{bmatrix} M(t) \\ \dot{M}(t) \\ \ddot{M}(t) \\ \dddot{M}(t) \end{bmatrix}$$

It shows that the observability matrix A(t) attains full rank if $\dot{\lambda}$ is non-zero which means movement of the sensor needs to be constantly accelerating to obtain full observability of a constant velocity target.

2.5 Tracking in autonomy

Being able to detect and estimate the behaviour of objects serves a vital role in building situational awareness. Correctly estimating and predicting the movement of other vessels at sea will give a solid basis for decision making when moving in order to avoid collisions.

2.5.1 Autonomy

Automation has been part of the human society for hundreds of years, with the industrial revolution and the transition from production using our hands to using machines perhaps being the biggest historic event. The complexity and amount of automated task has increased greatly over the last years, and with the development into newer and faster computers over the last few decades we are able to perform more complex and more computationally demanding tasks. Autonomy is a word often used when the automation reaches a certain level of complexity, such as automated control over a vehicle. The Norwegian Maritime Authority defines five levels of autonomy [43].

1. Decision support
2. Automated
3. Periodically unmanned/uncrewed
4. Unmanned/uncrewed
5. Fully autonomous

For a system to be autonomous it requires accurate information about the surroundings, and the level of autonomy that can be safely introduced highly depends on the overall situational awareness of an autonomous system. The higher degree of automation the more time is needed between human inputs to the system, up the fully autonomous stage where the vehicle should need no input from a human to perform its tasks. While humans need breaks and often behave differently to the same information, the goal with a fully autonomous system is to perform its tasks without any need for human interaction or monitoring. Knowing that the amount of accidents at sea happening as a consequence of erroneous human action is between 60 to 90% [12] an autonomous system can help save lives as well as giving a substantial economical gain.

2.5.2 Maritime rules of the sea

The Norwegian Maritime Authority sets the regulations in Norwegian waterways, these are based on international conventions. The regulations are described in "Regulations of 1 December 1975 No. 5 for preventing collisions at sea (Rules of the Road at Sea)" [44], more commonly described as COLREG. The rules apply to all vessels upon the high seas and in all waters connected that are navigable by sea-going vessels.

Rule 7 of these regulations applies to reducing the risk of any collision. Here it is stated;

> *(b) Proper use shall be made of radar equipment if fitted and operational, including long-range scanning to obtain early warning of risk of collision and radar plotting or equivalent systematic observation of detected objects.*
>
> *(c) Assumptions shall not be made on the basis of scanty information, especially scanty radar information.*

Rule 9 applies to movement in narrow channels.

> *(d) A vessel proceeding along the course of a narrow channel or fairway shall keep as near to the outer limit of the channel or fairway which lies on her starboard side as is safe and practicable.*

Rule 14 applies to a head-on situation:

> *(a) When two power-driven vessels are meeting on reciprocal or nearly reciprocal courses so as to involve risk of collision, each shall alter her course to starboard so that each shall pass on the port side of the other.*

Rule 18 defines the responsibilities between vessels

> *(a) A power-driven vessel underway shall keep out of the way of:*
> *(i) a vessel not under command*
> *(ii) a vessel restricted in her ability to manoeuvre*
> *(iii) a vessel engaged in fishing*
> *(iv) a sailing vessel*

In addition the Norwegian speed limits changed as of May 15th 2021 where you are no longer allowed to drive faster than 5 knots if one is less than 50 meters from people who are swimming or buoys. Earlier the same speed limit was in effect in areas close to shore where there were a lot of traffic and specific need to regulate the speed. This was either marked in sea charts or by signs in the areas [30].

3 Method

A qualititive analysis is conducted by measuring the results from the experiements towards relevant theory from Section 2. Data from different scenarios is collected to make a quantitive analysis and ensure a large enough sample size to come to a valid conclusion.

The data is collected in congested waters with limited room for safe navigation as this is the main operating environment for the Otter USV. It is collected as rosbags which can be played back later and used for training and testing. Because the data is collected as rosbags we can play it back when analyzing given conditions and avoid noise and minor differences when repeating the experiments. Shutting down different rosnodes like the camera detector or the LIDAR processing pipeline will be useful when analyzing sensor dropouts in the tracker, both registered and unregistered. Data is also collected in rain and in poor light conditions which helps test the robustness of the system when there is extra noise on the LIDAR and less useful information from the cameras.
Both target boats are equipped with GPS receivers and their position is logged locally. Even though the position of all three vessels present in the scenarios is logged locally they are all referenced to GPS time with a very low and quantified delay. This ensures precise positioning when playing back the data set. The accuracy of the tracking estimates is compared to the GPS positions and discussed. This is done over several instances of different scenarios to ensure validity and credibility in the results.

3.1 Test scenarios

As one of the goals in this project is to quantify the robustness of a multi-sensor tracker we conduct some tests of the tracking pipeline. Because of its size and modularity the Otter can be disassembled and transported to sheltered survey areas such as lakes, rivers and harbour areas. This is also where mid-range sensors like a LIDAR and cameras are the sensors of choice. In such congested waters with limited room for safe navigation it is important to obtain correct information about other vessels for use in path planning and collision avoidance. As such Trondheim harbour is a fitting testing area and will be used for data collection to be used in these tests. As seen in Figure 31 it is also filled with static targets that will serve as a challenge for the camera detector.

Three vessels are present in each scenario:

Vessel	**Role**
Otter 27 (Figure 27)	Ownship w/ Sensor hub
Otter 9 (Figure 42)	Target boat
Juggernaut (Figure 41)	Target boat

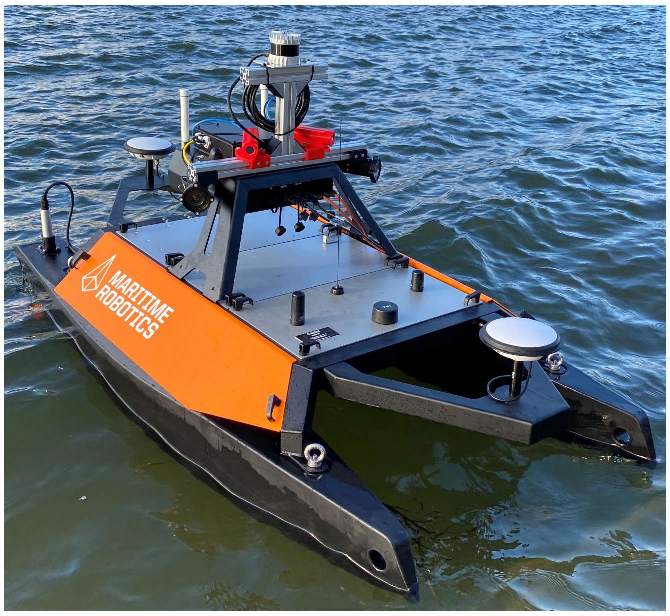

Figure 27: Otter 27

The Otter aswell as the sensor rig on Otter 27 is extensively explained in sections 4.1.1 and 4.1.2 while the two target boats are described in section 4.1.3.

3.1.1 Scenario one - Overtaking

In the first scenario Otter 27 is driving towards and passing Otter 9. The Juggernaut starts behind Otter 27 before it speeds up and passes it. When the Juggernaut reaches the end of the path it stops and waits for Otter 27 to catch up, then it follows the Otter 27, driving behind it back to the starting point.

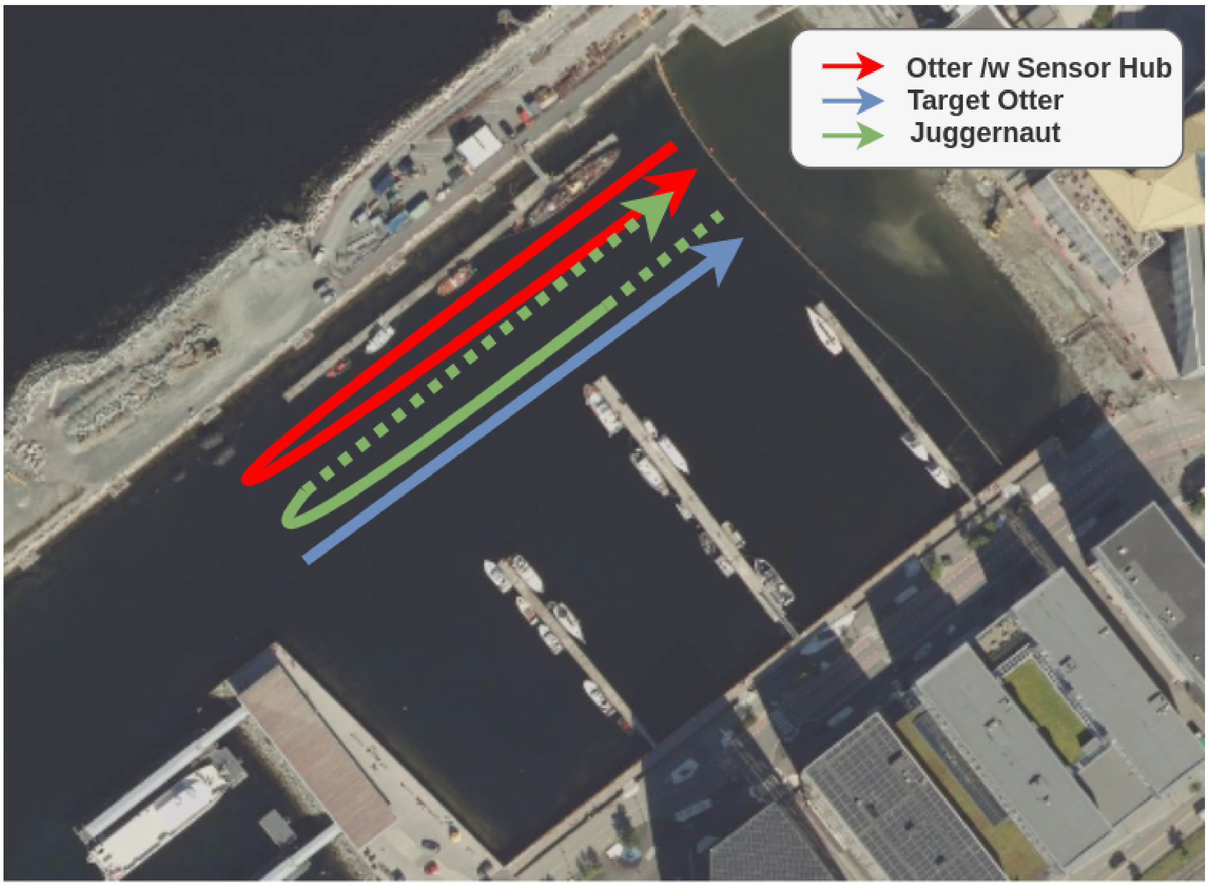

Figure 28: Movement of the boats in the first scenario.

The goal of this scenario is to test the multi target tracker with boats passing in close vicinity to own-ship in closed waters. Because the Juggernaut and target Otter spend a good amount of time behind the main Otter we get to test the multi sensor tracker when the targets are out of line of sight of the cameras and only within the LIDAR field of view.

3.1.2 Scenario two - Occlusion of target

In the second scenario the Otter 9 and the Juggernaut are both driving in opposite directions making a turn, passing on their starboard sides. The Juggernaut does the wider turn. Otter 9 makes a turn when Otter 9 is occluded by the Juggernaut and the two targets are aligned.

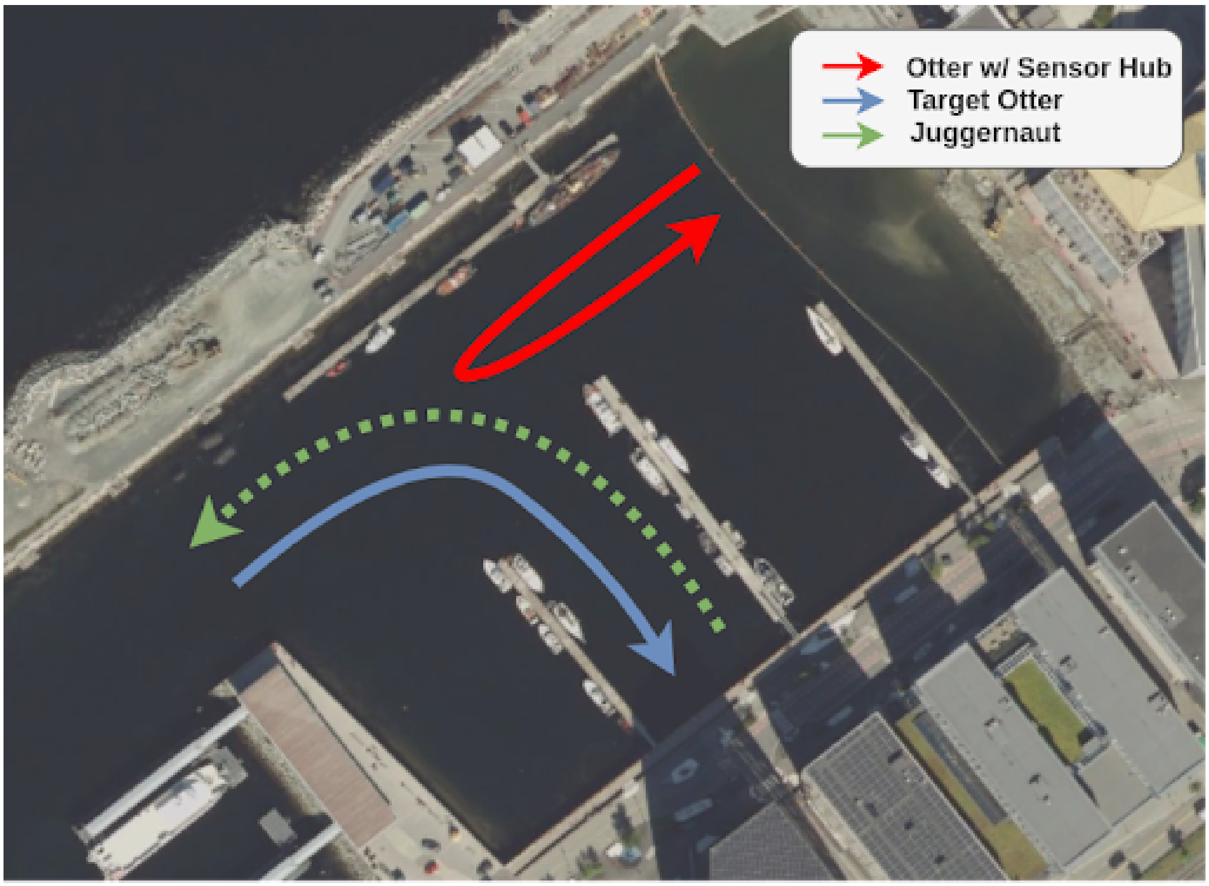

Figure 29: Movement of the boats in the second scenario.

The goal of this scenario is to test the multi target tracker when one of the targets, namely Otter 9 is occluded by another target, the Juggernaut. Because of the small size and relatively long distance between Otter 27 and Otter 9 the camera detector will be challenged on recognizing the target Otter. The two targets are also passing relatively close.

3.1.3 Scenario three - Passing of target

The third scenario is similar to the second scenario but with the roles swapped around. The main Otter and Juggernaut drive towards each other and pass while doing a turn while the target Otter drives back and forth in the main Otter's old path.

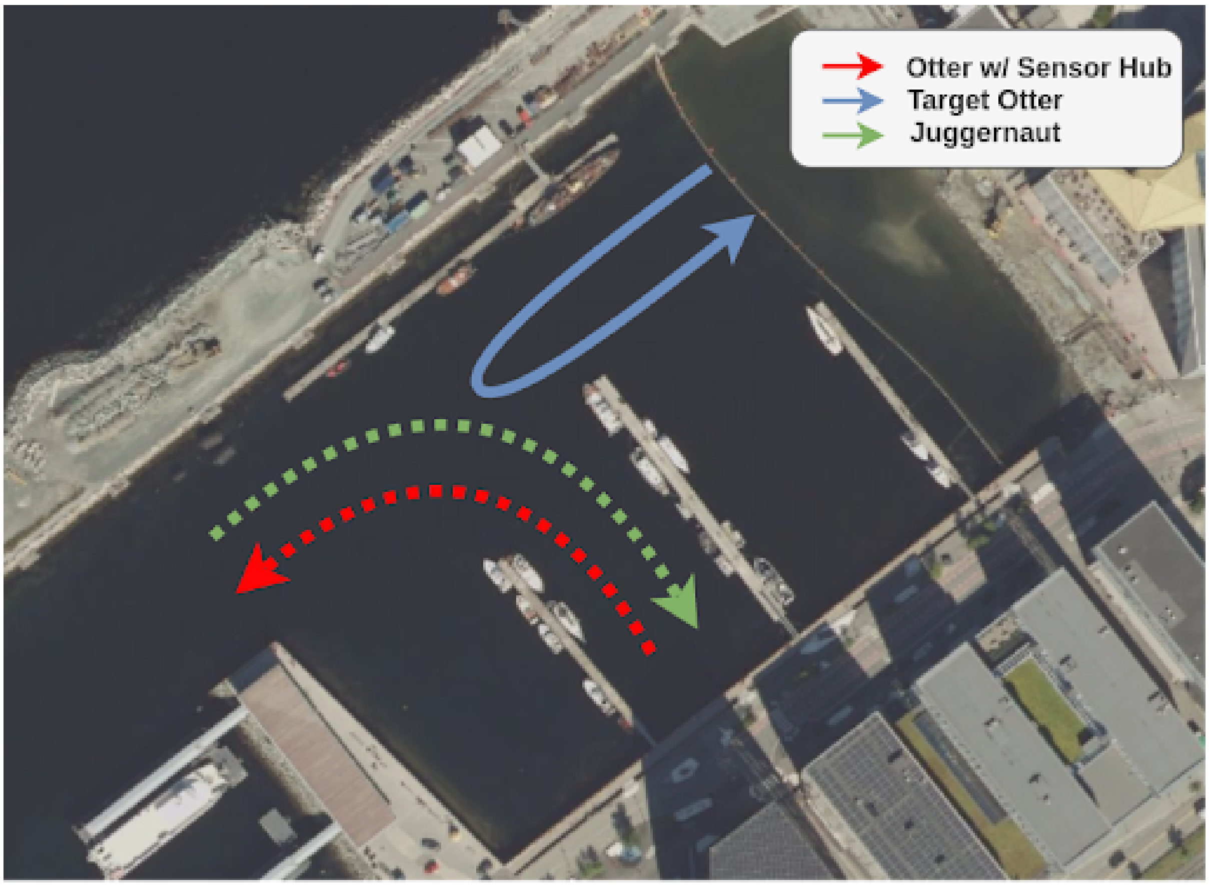

Figure 30: Movement of the boats in the third scenario.

The goal of this scenario is similar to scenario two but with Otter 9 initiating a turn while it is outside the cameras field of view. It is also a scenario where the Juggernaut moves from the port cameras field of view to the starboard cameras field of view, as well as a period where it is visible in both cameras. Since we will have two bearing angles referencing it's position at this point we should be able to triangulate the position given good calibration.

4 Experiment setup

The hardware used in data collection and the details of the software pipeline used for analysis will be expanded upon in this Section. The hardware setup used in the test scenarios are explained in Sections 4.1.1 and 4.1.2. The two target boats are outlined in section 4.1.3 along with the accuracy of their GPS positioning. The pipeline, including the image detector, LIDAR pipeline and tracker setup is explained in section 4.2

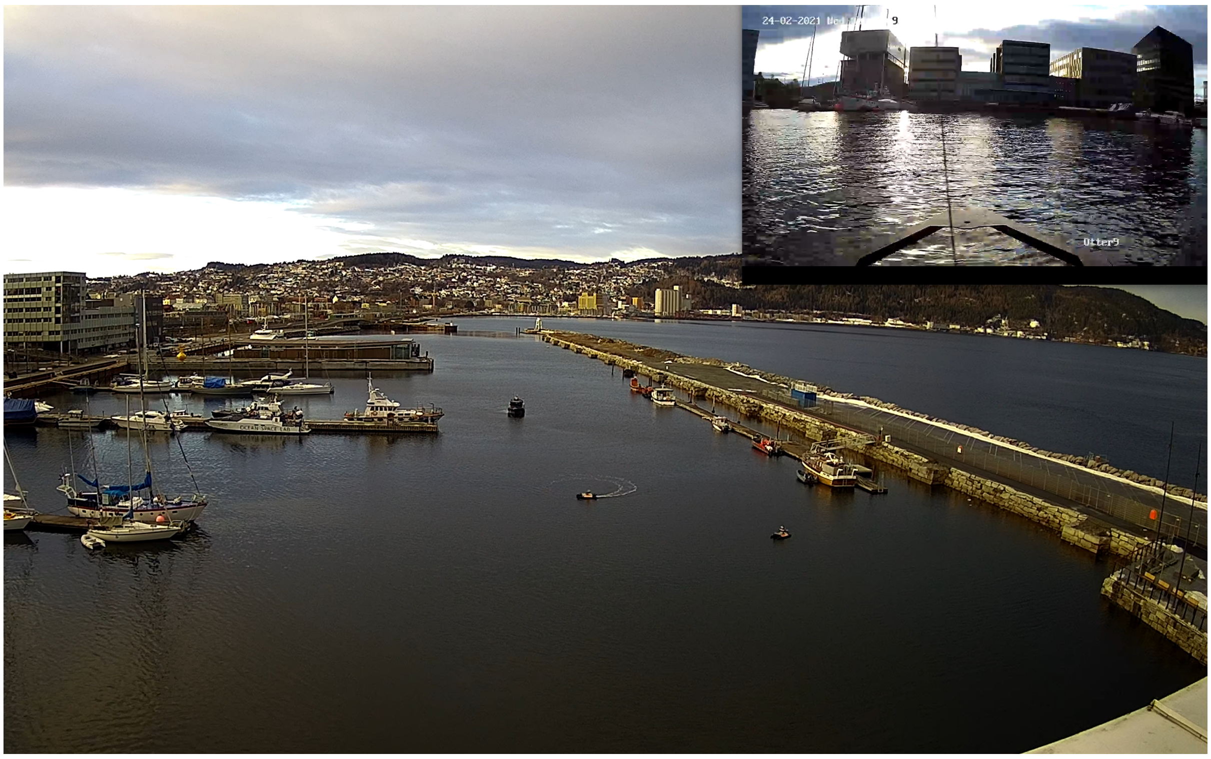

Figure 31: Overview of the testing area with the three boats used for data collection.

Figure 31 shows an overview image from the vehicle control station at Maritime Robotics. This was used when controlling the two Otters remotely. In the top right corner the camera on top of Otter 9 is visible. This is only used for situational awareness for the operator.

4.1 Hardware setup

4.1.1 Otter USV

The Otter (Figure 32) is an unmanned surface vehicle (USV) from Maritime Robotics and will serve as the main development platform for their autonomy project. The idea is to have a high grade of scaleability, as the end idea is to increase the autonomy on all of their products.

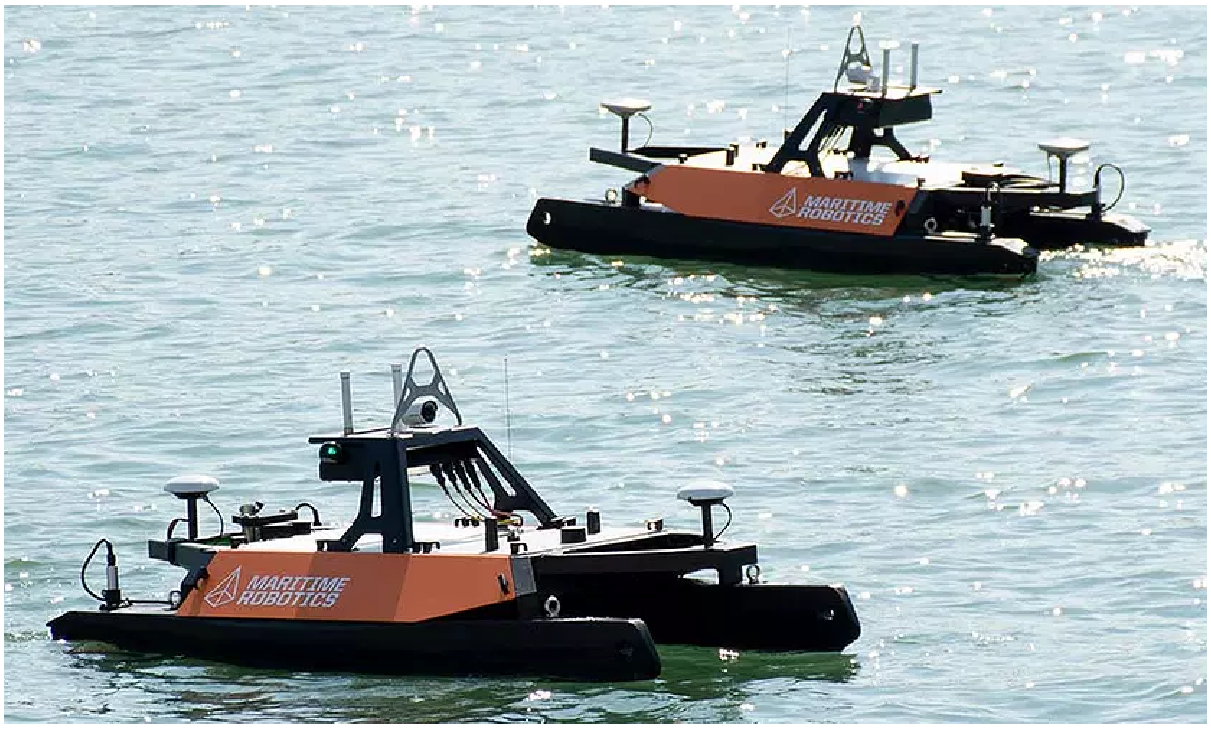

Figure 32: Otter USV

The Otter is the smallest USV delivered from Maritime Robotics and provides an easily deployable and modifiable testing platform for autonomy development, and is used for collecting data in the experiment. It is usually equipped with payload equipment used to perform tasks like environmental monitoring and seabed mapping. The Otter USV is 2 meter long and 1.2 meter wide, and because of its size and capabilities, it is often used in areas that cannot be reached by larger vessels. This increases the requirements for an autonomous system, being able to maneuver and make qualified decisions in smaller areas.

4.1.2 Sensor rig

Because of the Otters modularity, a separate targa is constructed for the purpose of autonomy development. This targa interfaces with all the sensors and contains the processing power needed for the Otter to develop situational awareness. The targa and the different sensors can be seen mounted on the Otter in Figure 33.

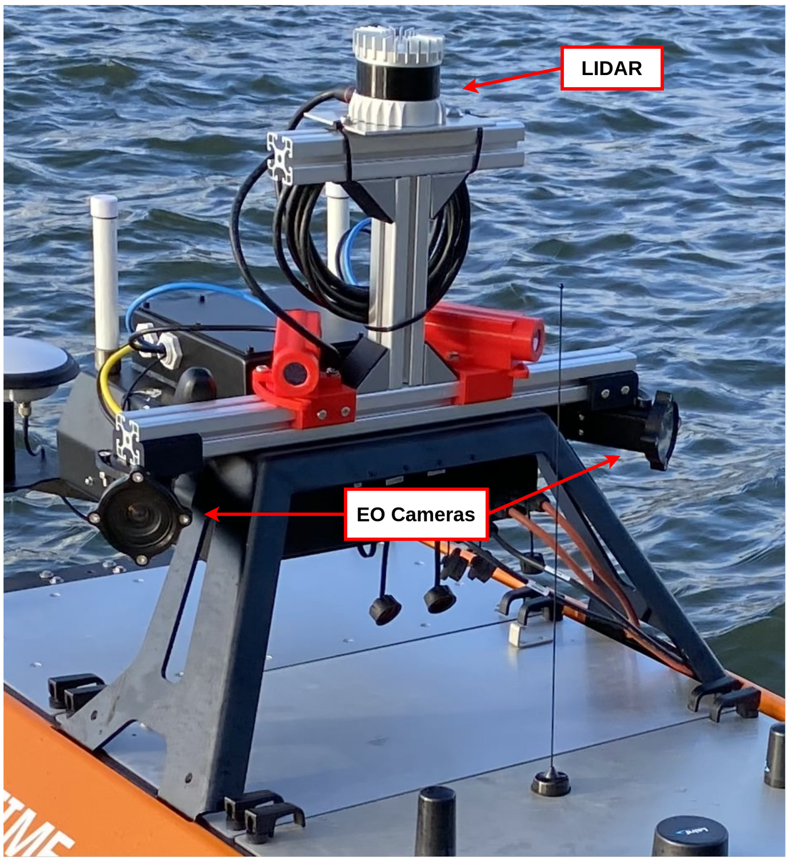

Figure 33: Otter targa for autonomy development

The Otter is usually equipped with sonars used for mapping that require high accuracy positioning systems. Most of these sonars are equipped with INS (Inertial Navigation System) with RTK-precision, and the output of these are transferred to the targa and used in the autonomy pipeline. For this project however, a Kongsberg mini-MRU [41] (Motion Reference Unit) is used to keep track of position, velocity, angular orientation and angular velocity. The MRU outputs time-stamped data and is synchronized with GPS time using PPS. It has an error in timing of less than 1 ms.

The sensor hub is equipped with two Dalsa Genie Nano C4040 EO-cameras (Figure 34.) They provide 4112x3008 resolution images and are equipped with Fujinon CF8ZA-1S lenses providing 85.7 degrees horizontal field of view (Figure 35). The camera mounts are constructed such that the horizontal angle can be modified, giving support for both mono and stereo vision, and the cameras are setup with hardware-triggering.

Figure 34: Genie Nano C4040

Figure 35: Fujinon Cf8ZA-1S

In this thesis the cameras are mounted for a dual stereo vision setup with around 170 degrees field of view in front of the Otter.

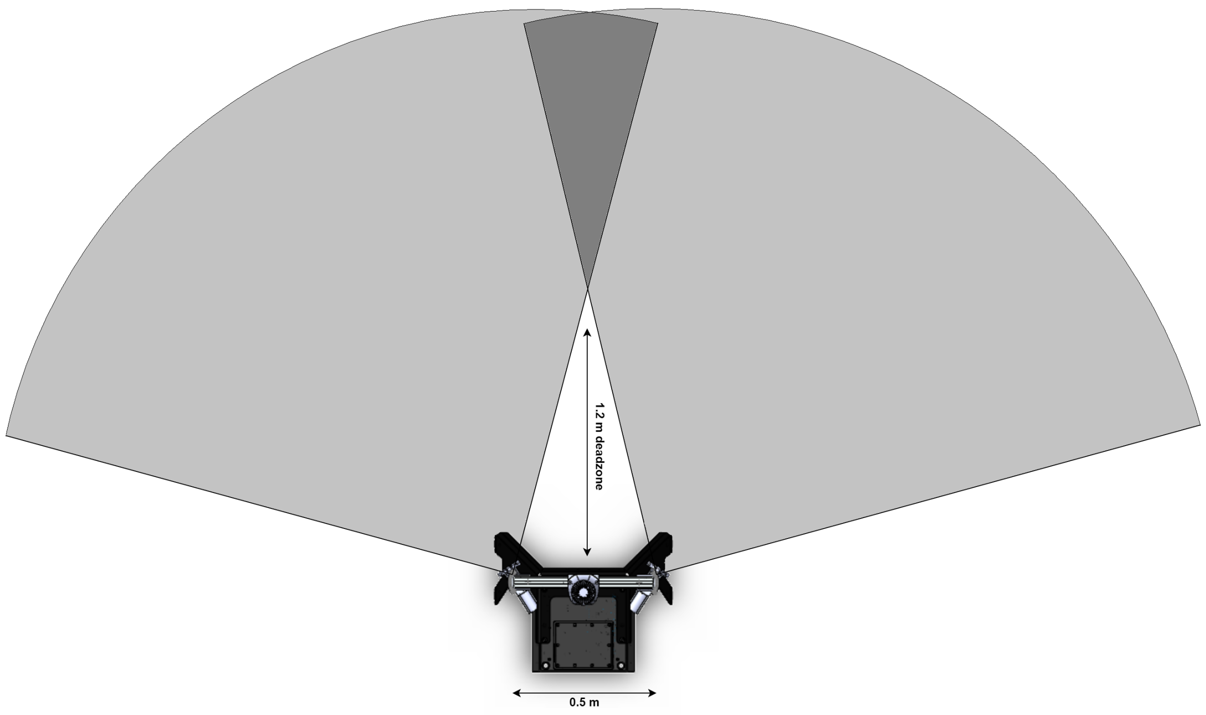

Figure 36: Field of view of EO cameras.

Data collection is done with an Ouster OS-1 128 LIDAR. This is a state of the art LIDAR with high range as well as high angular and vertical resolution and its specifications can be seen listed in Table 4.

	Ouster OS1-128
Vertical Resolution	128 Channels
Vertical Field of View	$\pm 22.5°$
Horizontal Resolution	512, 1024, 2048
Horizontal Field of View	360°
Range	120 m
Precision	$\pm 7 - 50$ mm
Rotation Rate	10 or 20 Hz
False Positive Rate P_{FA}	1 / 10.000
Extras	IMU, PPS Sync

Table 4: LIDAR Specifications.

The LIDAR is mounted on a rail at the top of the Otter as seen in Figure 33 to avoid anything blocking its 360 degree field of view.

Figure 37: Ouster OS-1.

The Ousters internal clock has a timestamp resolution of 1 microsecond and a data latency of less than 10 milliseconds. Because the LIDAR timestamps packets with its internal clock it is convenient to synchronize this using GNSS-PPS and use it as the reference time. For more information about these timestamping methods the reader is reffered to the project thesis [19].

The Sentiboard (Figure 38) deals with most of the low level time synchronization between the sensors on the rig. It has two SPI, two RS-232, three UART and one RS-422 I/O that all can be used to communicate with a range of different sensors [42]. Each I/O can be set up as an hardware interrupt when receiving data from a sensor. This data is re-packed and transmitted with an extra header containing a time stamp synchronized with GPS-time. Each I/O can also be set up for hardware-triggering and is able to provide a steady PPS signal. It can also synchronize all of the I/O across each other, so one can ensure that all outputs trigger sensors at the same time. The trigger signals can also be output at a different frequency relative to each other, while still maintaining synchronization.

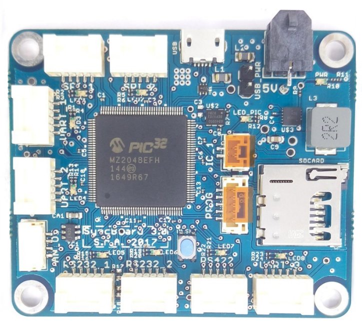

Figure 38: Sentiboard

The Sentiboard keeps track of time using a low drift oscillator with an accuracy of 10 PPM (parts per million.) This means that after a bit more than one and a half week (11.57 days) without any type of error correction, it can have a maximum inaccuracy of 10 seconds. Tests show that this synchronization board can relate sensor measurements to an absolute time reference with a clock drift of 1.9 microseconds per second RMS if it is connected to a GPS PPS-signal [1]. By using a low level system like the sentiboard for time synchronization we avoid delays due to other processes running on the local system. Figure 39 shows the difference in accuracy when timestamping using the sentiboards time of capture (ToC) instead of time of arrival (ToA.)

The current system has the NTNU Sentiboard synchronize with GPS time using PPS. The position, pose and twist of the vehicle is output from a Kongsberg Mini-MRU and sent through the Otters On-Board System (OBS) to the targa where it is logged in ROS. The timestamping of this INS data is done on the MRU itself which is synchronized with GPS time. The OBS is synchronized with GPS time via another u-blox receiver and set up as a NTP-server. The onboard computers are synchronized with this as NTP-clients. The cameras are hardware-trigged at a frequency of 5 Hz through the NTNU Sentiboard,

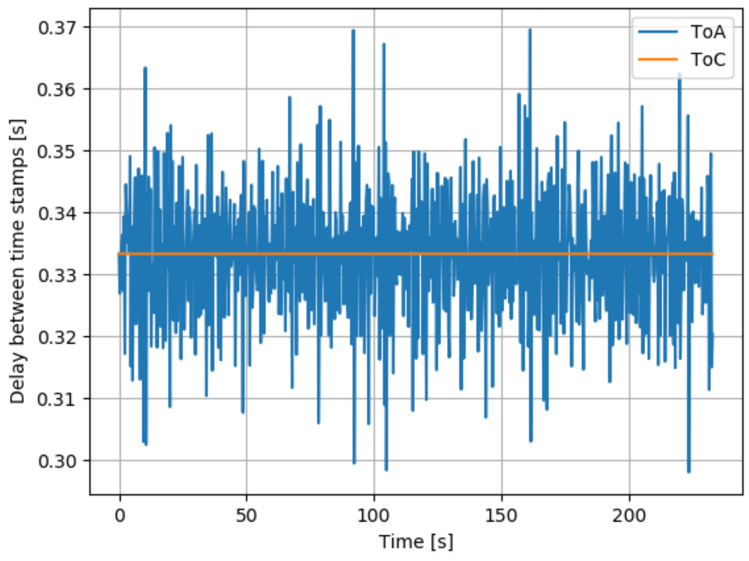

Figure 39: ToC vs. ToA timestamping precision.

and the time of the triggers are parsed in the sentiboard driver and broadcasted on a ROS-topic. This is merged with the camera images in the camera driver. The Ousters LIDAR data is timestamped locally on the sensor, with the internal clock of the sensors being synchronized with the GPS time messages from the OBS and a PPS-signal from the Sentiboard. The timing setup in this system is visualized in Figure 40.

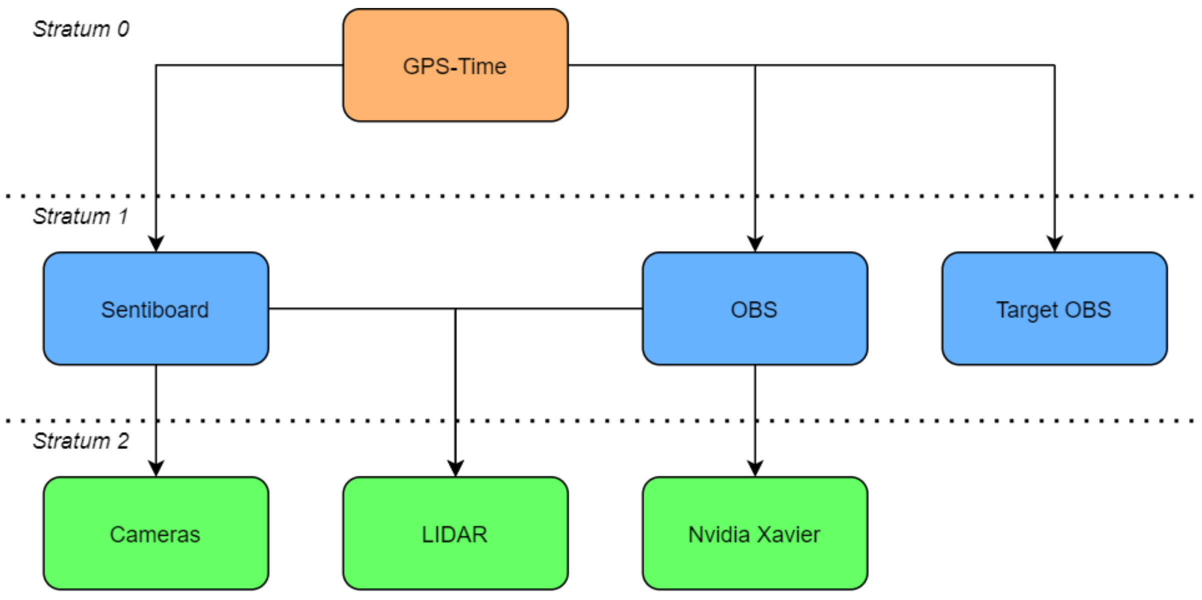

Figure 40: Time synchronization setup

4.1.3 Target boats

The first target boat is the Juggernaut. It is a Skarsvåg 880 built in 2019 and owned by Maritime Robotics. It is 8 meters long and 2.6 meters wide and modified so it can be controlled remotely. The position of the Juggernaut is received through a Simrad HS 60 and the stated accuracy from the GPS manufacturer is shown Table 5. This is documented as the worst case error and when the GPS performs a low grade of maneuvering the error is experienced to be a lot lower.

GPS Accuracy	± 1m
GPS Time Accuracy	30 ns
Update Rate	10 Hz

Table 5: Simrad HS60 GPS Specifications [25].

The Juggernaut, like most other boats, have a dark colored hull. Dark surfaces has low reflectivity in the spectral range of the LIDAR, making it a challenge to detect at longer ranges. Because of its size it should still be detected reliably when it is within the LIDAR working range.

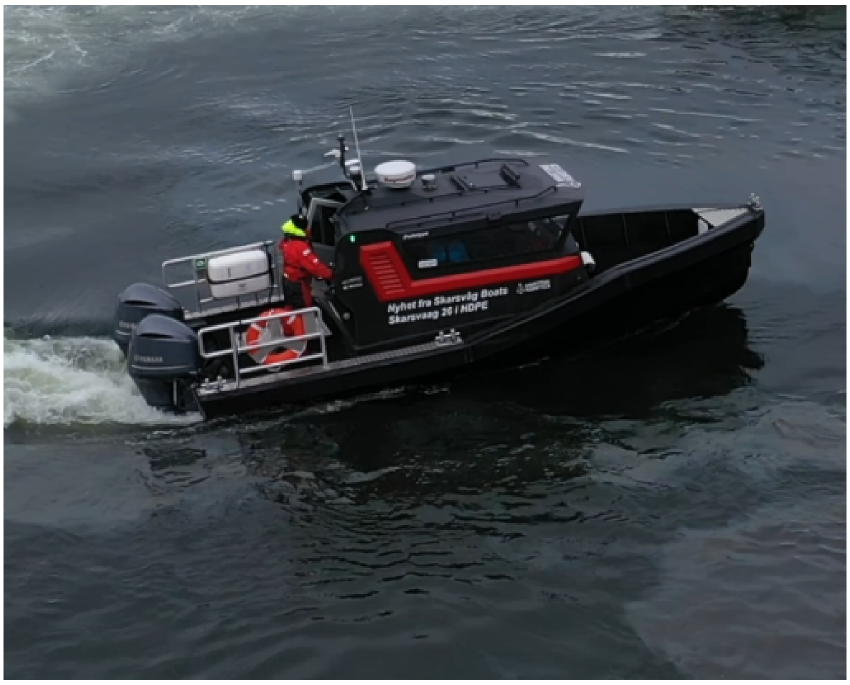

Figure 41: Juggernaut

The other target boat is an Otter USV. The specifications for the Otter is listed in section 4.1.1, and this one is set up with a ZED F9P Development kit for GPS positioning. Because of its relatively small size it should provide a detection challenge both for the LIDAR and the camera detector at mid to long range. The Otter is relatively low, and the high vertical resolution of the Ouster and the Velodyne is needed to detect it reliably. Like the Juggernaut most of it is colored in black, decreasing reflectivity making it hard to detect using a LIDAR.

GPS Accuracy	± 1m
GPS Time Accuracy	30 ns
Update Rate	20 Hz

Table 6: Zed F9P Specifications [46].

Just like the Juggernaut the GPS position of Otter 9 is logged in a rosbag. It is synchronized with GPS time giving a common time reference with the sensor rig.

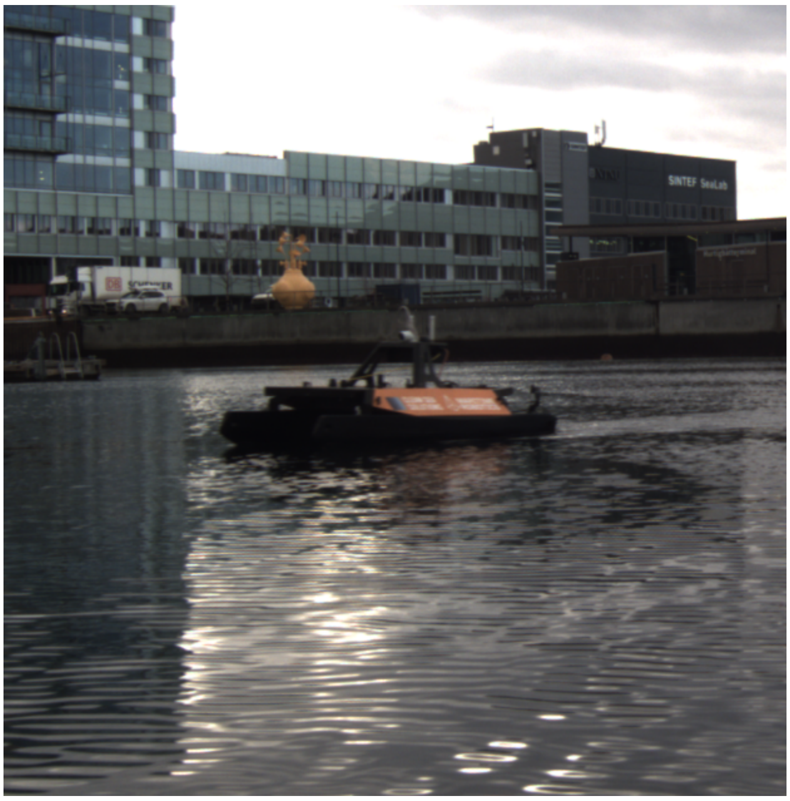

Figure 42: Otter 9

4.2 Pipeline implementation

This section aims to describe the work flow and some of the design choices made when creating the software pipeline shown in Figure 43. The pipeline is running in ROS and uses some modules and message types developed in the autosea project for visualization and publishing of tracker estimates. Even though the data from the scenarios are collected locally on the computers inside the Otter targa it is analyzed remotely to avoid having to take into consideration limited computational power.

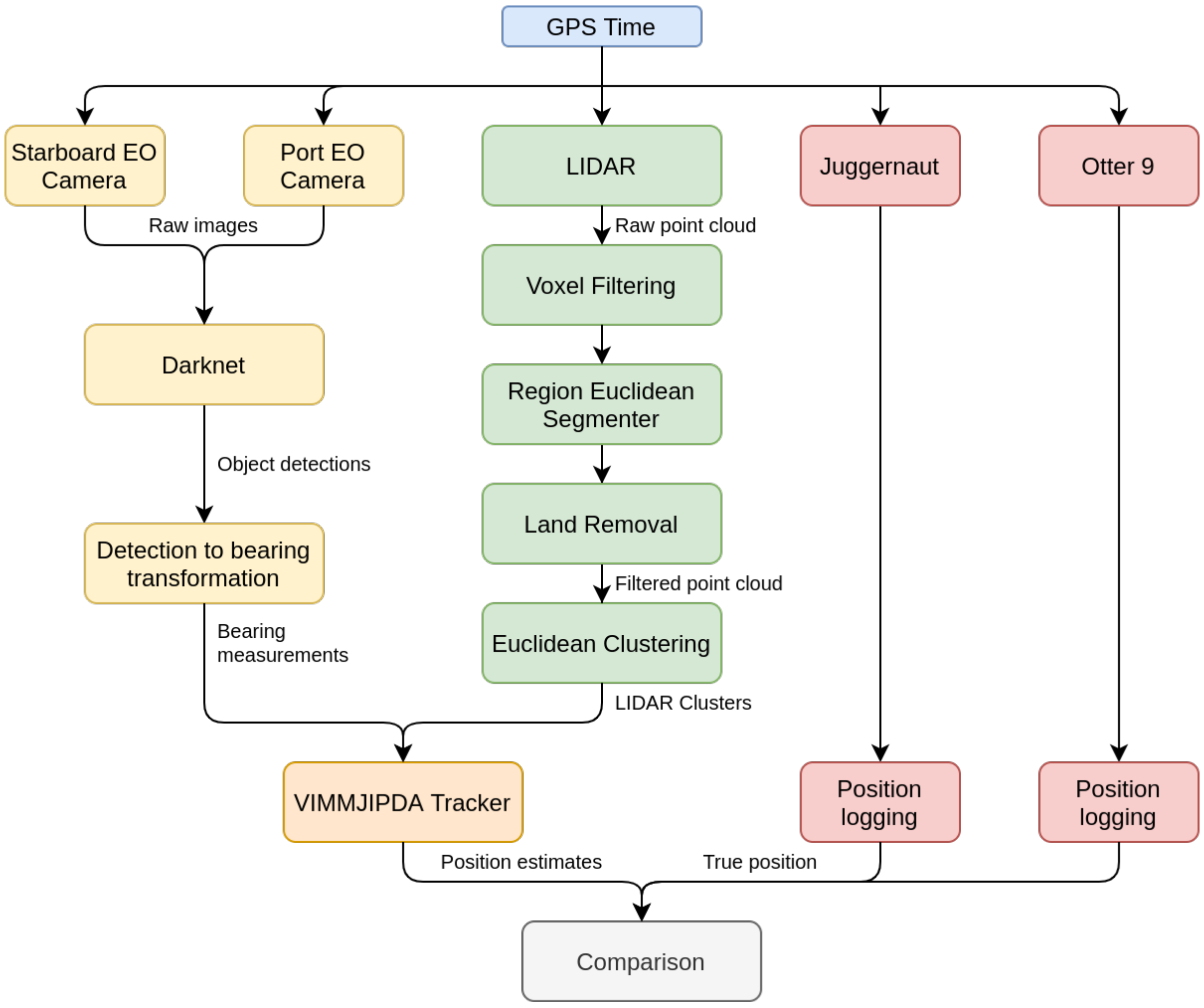

Figure 43: Pipeline in ROS

4.2.1 Camera pipeline

Using an object detector we want to obtain a bearing measurement from boats in the vicinity that can be used in the tracking pipeline. Because the main goal of this thesis is to test the robustness of the target tracker in congested waters we want reliable and consistent information from the detection pipelines. Having an object detector that can at least perform up to a standard, more general object detector should be the goal, and we want to achieve a mAP of at least 80% to avoid poor sensor data cluttering our analysis. Maritime Robotics has a modified darknet [38] object detector pipeline that is part of their situational awareness pipeline. This detector is used in this project.

Class	**Unique truths**	**TP**	**FP**	**Precision**	**Recall**	**AP**
Boat	1845	410	35	22.2%	92.13%	22.2%

Table 7: Performance of a pre-trained YOLO model on the data set.

Initial testing showed that a pretrained YOLO v3 network performed poorly on the data sets and the decision was made to train a new model from scratch. Due to the relatively unique shape and size of the Otter USV it is hard to detect using the pretrained model presented in "YOLOv3: An Incremental Improvement [39]". This model only contains boat as a general class and the performance on the dataset can be viewed in Table 7. Data from each scenario was collected in good and bad weather for three different LIDARs as datacollection was done while benchmarking a set of LIDAR's. This gave a total of six data sets for each scenario. Because the same vessels were present in each scenario, training on a few of these data sets in different light conditions and types of weather should be sufficient.

Scenario	**Weather**	**#Images pr. camera**
1	Cloudy	300
2	Sun	300
2	Rain	200
2	Dark	400
3	Sun	175
3	Rain	150
3	Dark	160

Table 8: Spread of annotated data from EO cameras.

In addition to the data from Table 8 around 1000 images from outside the data sets were annotated. These images were from different test runs when the performance of the system was being tested and validated, and although they did not pertain to any scenario they are collected within the same area and show many of the same objects. These numbers are for each individual camera and in total over 4000 images were annotated for the data set. The set was split into three pieces with a training, validation and a testing set. The script used to split this data set is added as an attachment to this report.

Type	**Size**	**Images**
Training set	85 %	3513
Validation set	10 %	413
Testing set	5 %	213

Table 9: The size of each dataset used for training, validation and testing.

The structure of the neural network that is trained is a YOLO v3 which is further described in [39]. Because of real-time limitations the images from the dalsa cameras are downscaled to 416x416 pixels. Increasing the image resolution and input resolution to the neural network causes a increase in memory usage of the graphics processing unit (GPU). This is already a limiting factor as the depth (amount of layers) of the network also affects memory usage. YOLO v3 is 117 layers deep and is considered to be a large neural network. Finding an optimal combination between network depth, resolution and processing time is out of the scope of this thesis. In a real system it would be hard to detect an Otter at long range using this resolution, however as the training data contains images of the Otter from the same angles as most data sets in different light conditions we expect to get good results using this network structure.

Class	**Unique truths**	**TP**	**FP**	**Recall**	**Precision**	**AP**
Motor vessel	1456	1419	105	97.46 %	93.11 %	97.46 %
Sailboat - Sail down	389	386	12	99.32 %	96.98 %	99.32 %
					mAP	96.98 %

Table 10: Performance of trained object detector on data set.

As can be seen in Table 10 the network performs very well on the training data. There are some false detections of motor vessels with a confidence threshold of 25%. Increasing the threshold to 85% we lose 30 true detections while the amount of false detections goes from 57 down to 48. As the neural network is already overfitted and most detections would rarely reach such a high confidence in a real scenario we do however choose to keep the threshold at 25%. The parameters used for training are shown in Table 11.

Parameters	**Value**
Batch	32
Subdivisions	4
Input resolution	416 x 416
Output classes	4
Learning rate	0.01
Momentum	0.9
Decay	0.0005
Training batches	20 000

Table 11: Parameters for training the neural network.

The Dalsa Genie Nano cameras are equipped with lenses that should have close to zero distortion. Because of the high resolution of such cameras zero distortion lenses are often a necessity as post processing of high resolution images require a lot of computational power. We assume zero distortion in the images and do not apply a distortion model to our bearing outputs. The calibration of the transforms to the cameras is however of high importance when fusing the measurements together with the LIDAR. If sensors in a multi sensor tracker are giving contradicting information it will degenerate the accuracy of our estimates and force us to use higher uncertainties in our models.

After data collection the transform to the port camera was found to be inaccurate when comparing the bearing measurements to the point cloud.

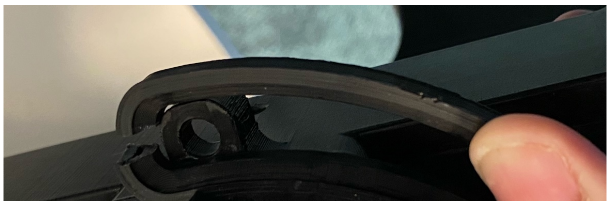

Figure 44: The broken camera mount.

Looking at Figure 44 we see that the 3D-printed camera mount had been damaged. This had caused the port camera to shift slightly to the port side which is not representative to the camera transform. Looking at Figure 45 we see that the antenna in front of the Otter is visible in the bottom image and that the rail in front of the Otter is more visible in the port camera than it was originally. Correcting this static transformation is done by shifting the transform in a negative yaw-direction until the bearing measurements overlap the LIDAR points generated by the targets. The script for changing the transforms is attached to this report.

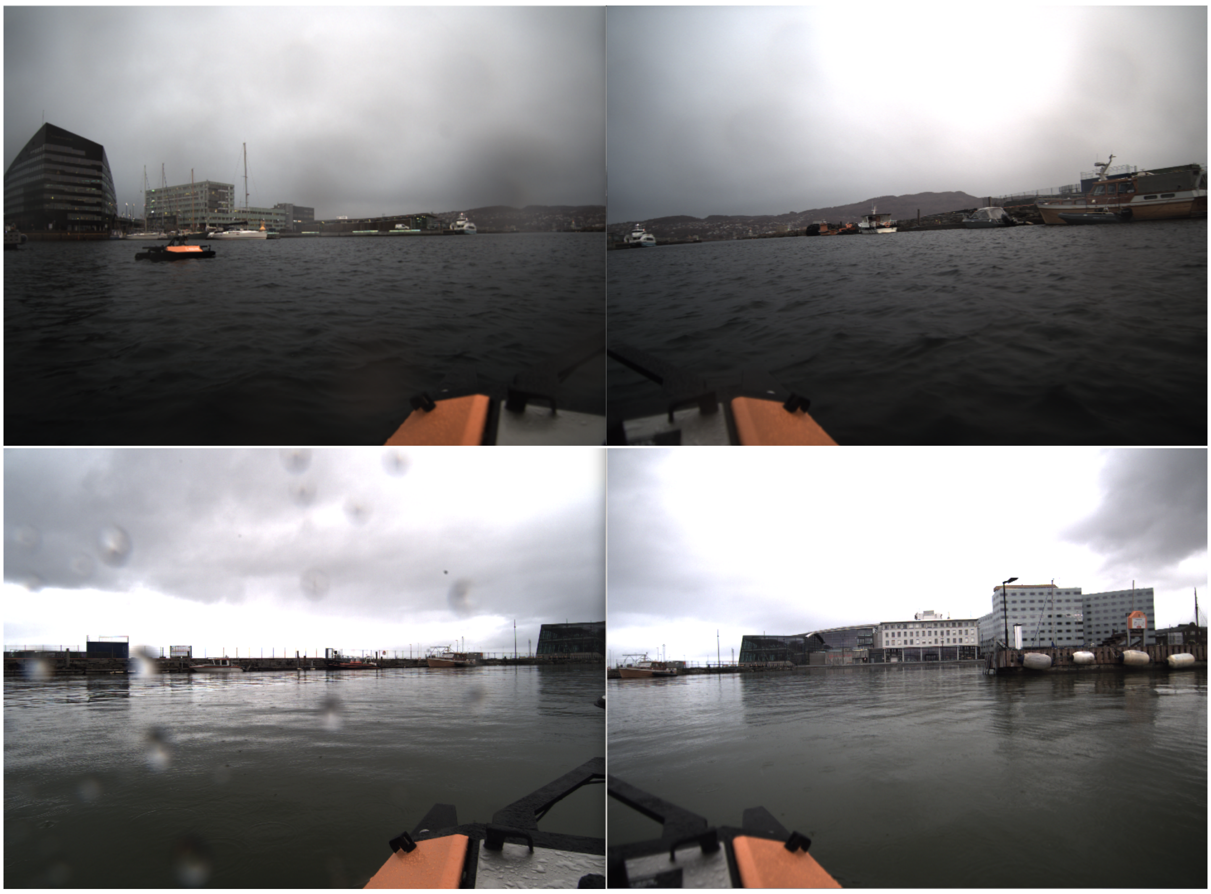

Figure 45: Camera images before (top) and after (bottom) the camera mount is broken.

The transformations from image detections to bearing measurements are displayed in Figure 46. This conversion is done in the local camera frame with the center of the image as the origin. As we know the vertical field of view of the camera as well as the resolution we apply a linear model to transform the pixel coordinates into a bearing angle. The bearing angle θ is given from the center of the bounding box. The elevation angle can be calculated in the same way, although as tracking is done in the Cartesian 2D plane we assume this to always be zero.

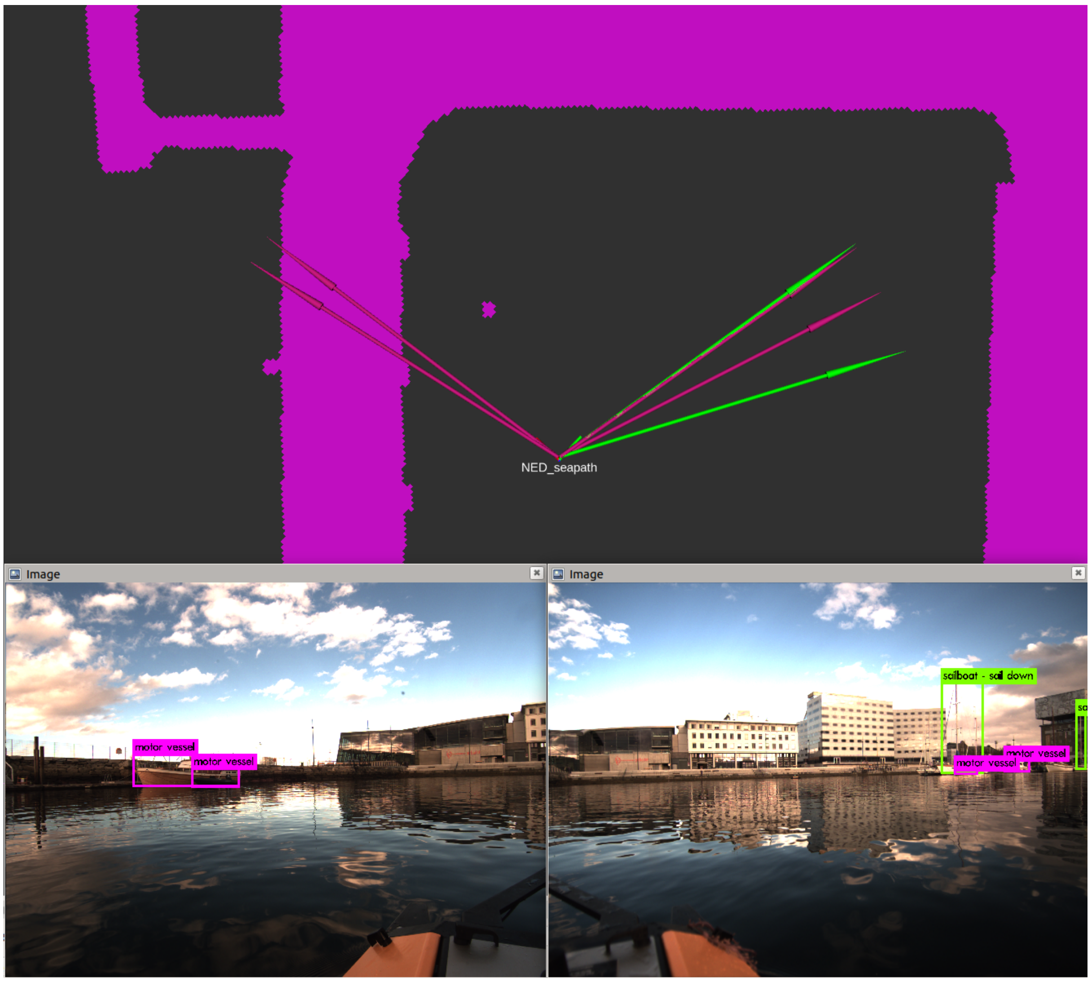

Figure 46: The output from the object detector transformed into bearing measurements.

$$\theta = x_{pixel} * \frac{\text{Field of view}}{\text{Image width}_{pixels}} - \frac{\text{Image width}_{pixels}}{2}$$

The sensor specific tracker parameters can be estimated using the results on the test set and the accuracy and time delay for the bearing measurements. Although the camera images are captured at 4112x3008 the neural network runs with an input of 416x416. The resolution of the bounding boxes follows the resolution of the network accuracy and as such the 85.7 degrees field of view of the cameras can be divided into an angular resolution of 0.2 degrees. After reviewing the output images from the neural network it is also clear that there is some noise in the placement of the bounding boxes. This adds a measurement uncertainty which we set to be around 2 degrees. This gives an overall angular measurement uncertainty of 2.2 degrees or 0.0385 radians.

The clutter density is estimated from the amount of false positives on the test set. From the results in Table 10 we see that the mean precision is 96.98 %. Likewise the probability of detection can be estimated by using the mean average precision in the same table at 96.86 %. This is however only the case when the objects are within the cameras field of view. As the cameras only cover around 170 degrees around ownship having such a high P_D would severly degenerate the quality of tracks outside the cameras field of view. P_D is hence set to be around half of this at 45%. For the camera measurements we estimate γ to be 3 by looking at how objects close to ownship move between image frames relative to the angular uncertainty in the camera measurement. The camera specific tracker parameters are summarized in Table 12. These values are just a starting point for tuning the tracker and will require modifications to achieve optimal performance.

Parameter	**Value**
σ_θ	0.0385 Radians
λ	10^{-5}
P_D	45 %
γ	3
r_{max}	110 m
r_{min}	10 m
$P_{initial}$	40 %

Table 12: Initial camera specific tracker parameters.

4.2.2 LIDAR pipeline

The raw data from the Ouster OS-1 at the start of scenario one is shown in Figure 47. At this point there are a few hundred thousands data points within the cloud that needs to be processed. A voxel grid filter is applied to down-sample the raw cloud. The parameters for the voxel grid filtering method are shown in Table 13.

Parameter	Value [cm]
Leaf size (x)	10
Leaf size (y)	10
Leaf size (z)	30

Table 13: Parameters for voxel grid filtering.

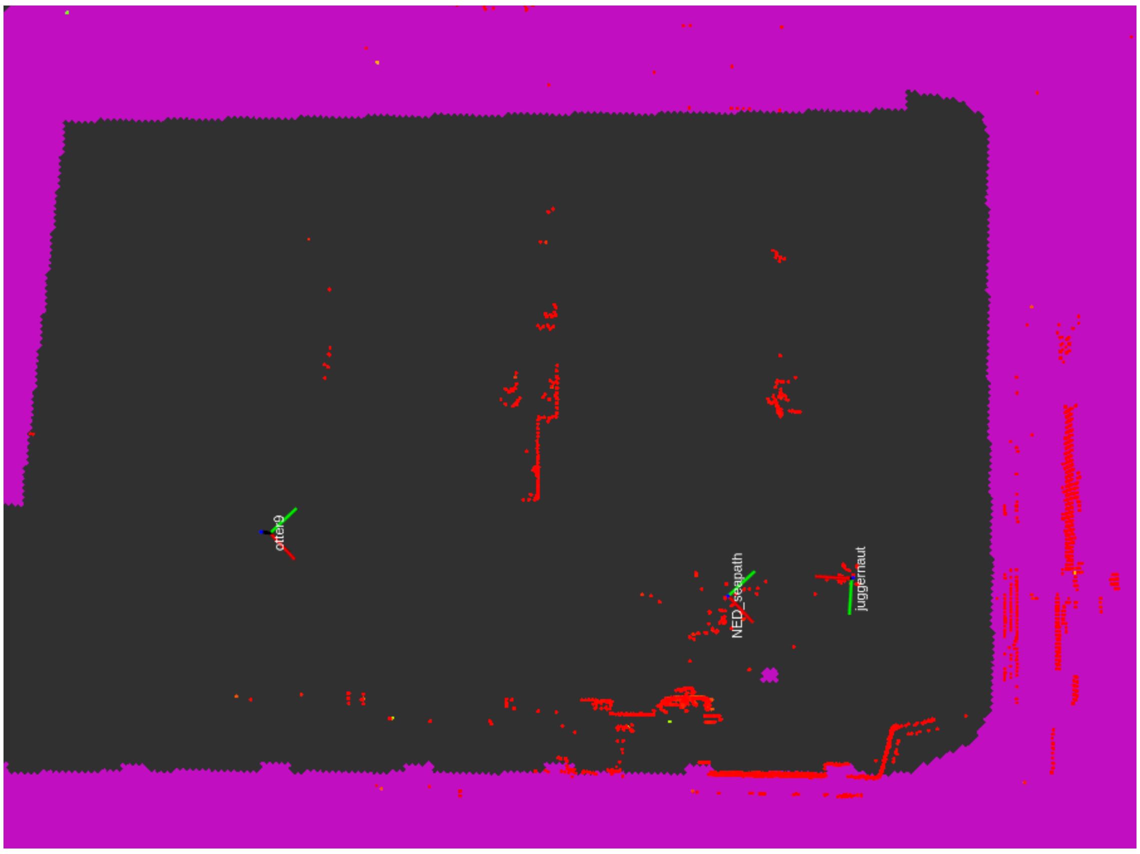

Figure 47: The raw point cloud output by the LIDAR.

The three coordinate systems in Figure 47 represent the Juggernaut, Otter 27 and Otter 9. The middle one is Otter 27 with the sensor hub and from the image one can see a lot of clutter around it. This is due to a lot of kelp floating around in the harbour because of nearby construction work. There were some attempts at applying the land filtering methods described in Section 2.2.2, although with poor results as there were not enough ground points to accurately model the ground plane. Instead the algorithm chose data points from nearby targets which gave a plane that was not aligned with the water surface and instead filtered out objects of interest. Instead of ground plane estimation a Euclidean segmenter from LidarPerception [18] is applied. The parameters can be seen in Table 14 and the resulting point cloud can be seen in Figure 48.

Parameter	**Value**
Region delta tolerance	0.1
Minimum cluster size	20
Maximum cluster size	30 000

Table 14: Parameters for the Euclidean cluster segmenter.

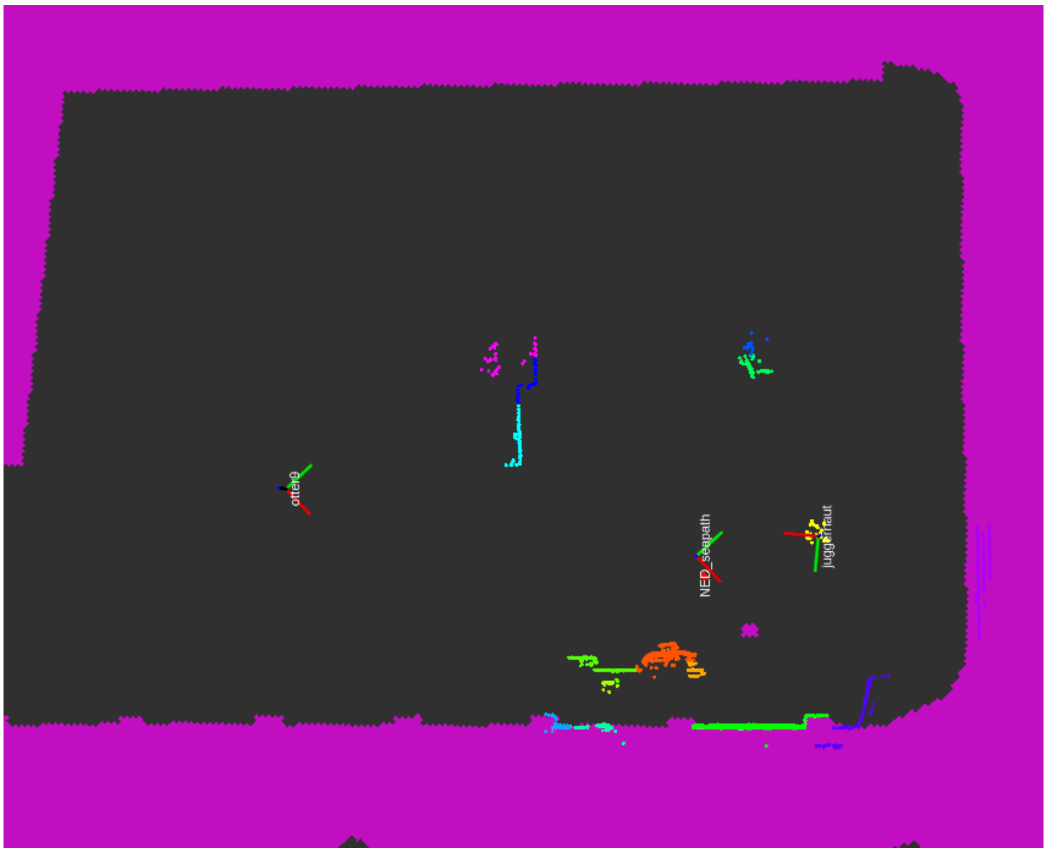

Figure 48: The output from the voxel grid filtering and segmenter.

The Euclidean cluster segmenter reduces the amount of data points by a good margin, however there are still too many points to input to the tracker. There are also points that are guaranteed to not pertain to any objects of interest. Looking at the right side of Figure 48 we observe a lot of points on land that are of no interest for the tracking application. In autosea [8] a module was developed that filtered out RADAR points on land based on a grid map. Because of the operating range of the RADAR this was made for targets a lot further away then our scenario, and as such the resolution of the grid was relatively low with a resolution of 10x10 meters. For the LIDAR points this was modified slightly, increasing the grid resolution to 10 cm and removing all points that are within two meters of the land cells in the grid map. After this land filtering we apply Euclidean clustering with parameters in Table 15 to reduce the data points further. The output after applying these two methods can be seen in Figure 49.

Parameter	**Value**
Cluster threshold	8.4

Table 15: Parameters for the Euclidean clustering.

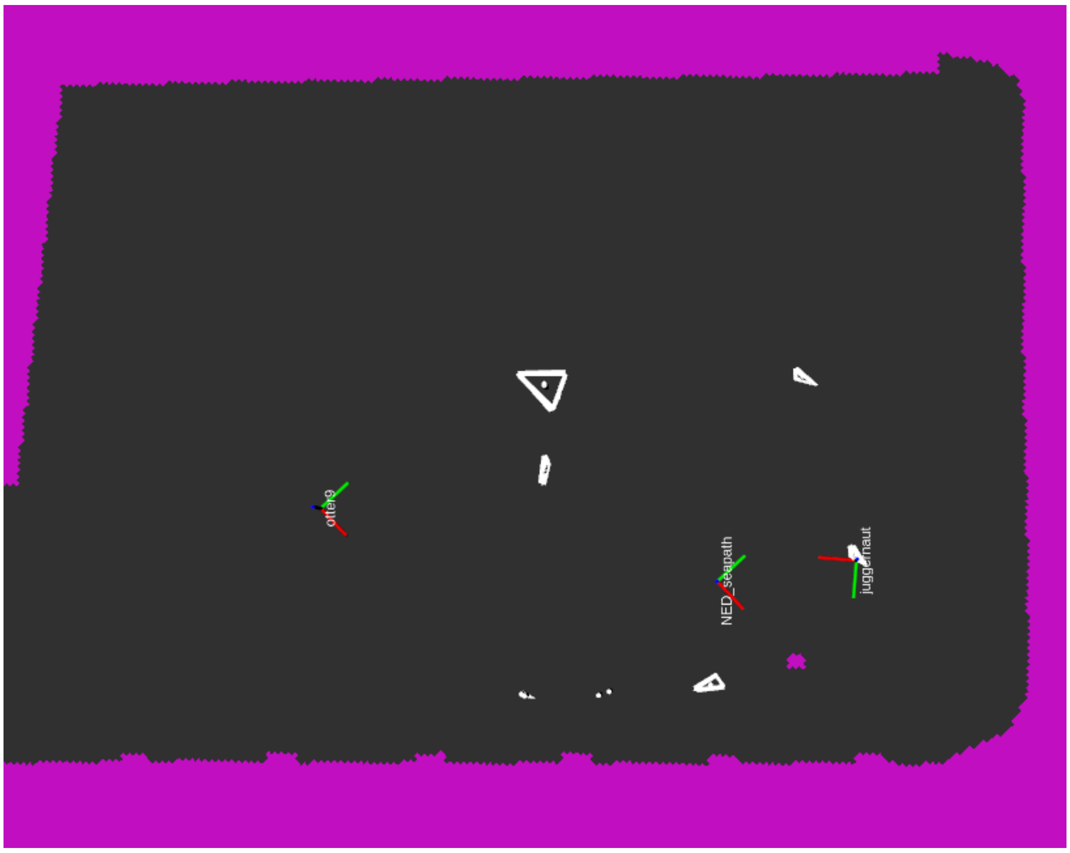

Figure 49: The output from Euclidean clustering and land filtering.

Like the cameras the LIDAR also has some sensor specific parameters that can be estimated. The clustering method does not output the center of an object as our single measurement after clustering as the sensor is only viewing each object from one side. As such the range covariance will vary with the size of the object and the stability of our clustering method. We choose a high enough value that should be able to cover all the targets in our scenarios with $\sigma_r = 1$. The same goes for the bearing covariance of the LIDAR measurements and we choose a value of 10 degrees or 0.03 radians to be able to cover instability in the clustering method. Because of the filtering we don't expect many false detections, and as such the clutter density is set to $\lambda = 10^{-6}$. Because few LIDAR data points can be interpreted as clutter and the LIDAR clustering method is stable we set the probability of detection P_D to be 98 %. The size of the validation gate is set to $\gamma = 3$.

Parameter	**Value**
σ_r	1
σ_θ	0.03 rad
λ	10^{-6}
P_D	98 %
γ	3

Table 16: Initial LIDAR specific tracker parameters.

4.2.3 Tracking pipeline

The original tracker presented in Audun Hems article [9] is tested on AIS measurements and measurements originating from RADAR with limited surrounding targets. The test scenarios from the article have also been performed far away from land with little clutter from objects that are of no interest to the tracker. As such a lot of effort has been put into filtering the LIDAR data and reducing the amount of data points originating from the LIDAR. The tracker works in a global NED frame with its origin placed at the Maritime Robotics office building. In a NED frame the X-axis has positive direction north, Y-axis positive direction east and the Z-axis positive direction down. The sensor frames are defined with X axis pointing forward, Y axis pointing starboard and Z-axis pointing up. A positive rotation in the sensor frame describes a negative rotation in the NED frame.

All sensor data originate from a local sensor frame which is transformed to the NED-frame using a static transformation to the sensor and the INS position of ownship at the time of data capture. Using the local sensor frame pertaining to each individual measurement we are able to shift the origin of the measurements which will give increased performance of the multiple bearing measurement sources. With the two camera measurements originating from different origins the uncertainty of targets within both images should be decreased greatly. Having two bearing measurements intersect the target our Kalman filter should be able to triangulate the position if the measurement uncertainty is small enough. If all bearing measurements originated from the same origin it would instead add a lot of inaccuracy to the tracker and would not make full use of multiple sources of bearing only measurements.

For the prediction step we choose three process models for our IMM-filter. With the high uncertainty in the bearing only measurements we use two CV models, one with high and one with low process noise. We also use one CT rate model with a relatively high noise as the Otter and Juggernaut can have a high turn rate. The noise parameters are gathered in Table 17.

Model	**Process noise**
CV_{low}	0.4
CV_{high}	2.3
CT	0.5

Table 17: Noise in the process models

We set the initial mode probabilities for new tracks to be high for the two models with high uncertainty. This is done to prevent incorrect convergence when using bearing only measurements. The probabilities are presented in Table 18.

Model	**Initial probability**
CV_{low}	20 %
CV_{high}	60 %
CT	20 %

Table 18: Initial probability for the process models

Because of the high update rate of the LIDAR the following mode transition matrix is chosen and shows to give good results. In the research paper presenting the VIMMJIPDA tracker [22] the transition probabilities for each mode is set to $\pi_{00} = 0.9, \pi_{11} = 0.9$ and $\pi_{22} = 0.99$. These probabilities give good results on AIS measurements which enter the tracker every 2 to 10 seconds and RADAR measurements with a sample interval of 2.5 seconds. The LIDAR point cloud is given at a much higher rate of 10 Hz and as such the probabilities below give good results. Higher probabilities give large and unwanted fluctuations in the mode probabilities.

$$\pi = \begin{bmatrix} 0.99 & 0.005 & 0.005 \\ 0.005 & 0.99 & 0.005 \\ 0.00005 & 0.00005 & 0.9999 \end{bmatrix}$$

For the same reason the survival probability is set to be relatively high at 99.9%. The tracks are set to be terminated when the existence probability reaches 30% and the transition from a preliminary to a confirmed track is set to happen when the existence probability reaches 80%. Lower survival probabilities with the high update rate from the LIDAR cause tracks to be terminated in less than a second when there are periods without measurements from the target. The tracker also contains a parameter for maximum idle steps which causes the track to be terminated after a certain amount of updates without measurements. For the original implementation this was set to 10 measurements. For this implementation it is set to 200 update steps to avoid early termination of tracks. With the images from the two cameras coming at a rate of 2 Hz and the LIDAR update rate of 10 Hz this allows tracks to survive for 14 seconds without measurements. The maximum velocity for the tracked targets is given as 5 m/s as this is the maximum speed allowed inside the harbour (Section 2.5.2.) The initial velocity covariance is given as 4 m/s. This is relatively high for the scenarios but higher covariance is shown to give better convergence of the tracks when using bearing measurements.

The tracker ROS node is made out of one tracker class that contains different callback functions, one for each type of sensors. The different callbacks unpack and convert the different measurements into types that are supported by the underlying VIMMJIPDA framework. Because the VIMMJIPDA originally supports cartesian and polar (range-bearing) measurements some modifications are done to support bearing only measurements. The underlying framework also applies to single sensor measurements and have general parameters for clutter density, detection probability and other sensor specific parameters. This is extended where the tracker now has several sets of sensor specific parameters that are used during the sensor specific callbacks. The VIMMJIPDA tracker framework contains a manager class that manages the data flow. The manager class populates the underlying tracker with the position of the sensor in the NED-frame before the measurement update step is called. This causes a possible race condition when the callbacks from different sensors are done in parallel at a high rate. If two measurements come too close to each other the position of the sensor is changed before the first measurement is finished processing and we get an incorrect state estimate. The tracker class is protected by a mutex to avoid this race condition. A mutex is a programmable lock that needs to be released before someone else can use the resource it is protecting. This gives good results but it does not allow for the tracker to process more than one measurement at once. Changing the structure of the tracker to avoid this mutex is suggested as future work. Our state vector is given as

$$x = \begin{bmatrix} p_x \\ p_y \\ v_x \\ v_y \\ \theta \end{bmatrix} \tag{7}$$

The non linear measurement model is given with

$$\hat{\theta} = atan2\left(\frac{\hat{p}_y - p_{y,o}}{\hat{p}_x - p_{x,o}}\right) \tag{8}$$

where the subscript o denotes the state of the own ship. The gating of measurements is done with the equation

$$\left\| \frac{(\theta - \hat{\theta})^2}{S} \right\| < \gamma \tag{9}$$

Given bearing only measurements the tracker attempts to associate each measurement to each confirmed and preliminary tracks. Each track state is converted into an bearing angle relative to the sensor origin and compared with the incoming measurements using Equation 9. If a measurement is associated with the target we perform the update step for the track estimate using the bearing only measurement model and the extended kalman filter equations shown in Section 2.4.1. Because we have a non-linear measurement equation we linearise our measurement model and the innovation covariance is calculated using the Jacobian H around the state estimate $\hat{x}$

$$H = \begin{bmatrix} \frac{p_y}{p_x^2 + p_y^2} & \frac{-p_x}{p_x^2 + p_y^2} & 0 & 0 & 0 \end{bmatrix} \tag{10}$$

If the measurement is not associated to any existing track state we instead initiate a new, preliminary track. Certain assumptions are used with the object detectors and scenarios at hand. One such assumption is limiting the range of the camera detections to be between a minimum range $r_{min} = 10$ and a maximum range $r_{max} = 100$. We estimated the mean position of the initiated preliminary track to be at the center of this range interval. The new tracks are initialized using the method from [2] with initial mean and covariance given by

$$\overline{r} = \frac{r_{max} - r_{min}}{2} \tag{11}$$

$$x = \begin{bmatrix} \overline{r}\sin\theta \\ \overline{r}\cos\theta \end{bmatrix} \tag{12}$$

$$P = \begin{bmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{yx} & \sigma_y^2 \end{bmatrix} \tag{13}$$

$$\begin{aligned} \sigma_x^2 &= \overline{r}\sigma_\theta^2 \cos^2\theta + \sigma_r^2 \sin^2(\theta) \\ \sigma_y^2 &= \overline{r}\sigma_\theta^2 \sin^2\theta + \sigma_r^2 \cos^2\theta \\ \sigma_{xy} &= \sigma_{yx} = \sigma_r^2 - \overline{r}^2\sigma_\theta^2 \sin(\theta)\cos(\theta) \end{aligned} \tag{14}$$

In Figure 50 we see a newly initiated track based on bearing measurements from the equations above. Note that the track is shifted slightly as it is only initiated after receiving a few successive measurements that have caused a high enough probability for a confirmed track to be initiated.

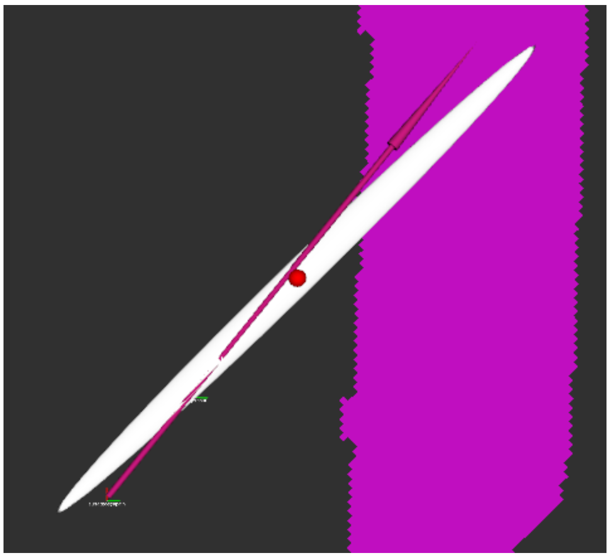

Figure 50: Newly initiated track based on bearing measurements

The final tracker specific parameters are summarized in Table 19.

Parameter	**Value**
Maximum Velocity	5 m\/s
Initial velocity covariance	3 m\/s
Survival Probability	99 %
Confirmation Threshold	80 %
Termination Treshold	30 %
Visibility Probability	90 %
Maximum idle steps	200 Steps
Birth Intensity	10^{-3}
$P_{CV,low}$	20 %
$P_{CV,high}$	60 %
P_{CT}	20 %
$\sigma_{CV,low}$	0.4
$\sigma_{CV,high}$	2.3
σ_{CT}	0.5
π_{11}	99%
π_{22}	99%
π_{33}	99.99%

Table 19: Final tracker specific parameters.

5 Results and discussion

5.1 Sensor processing delays

Fusing together measurements from a LIDAR and a camera object detector poses some timing difficulties. Processing the images in the object detector poses a large time delay that is not existent in the LIDAR pipeline. Figure 51 shows the delays in the LIDAR pipeline. The total delay represents the delay from the image is timestamped in the driver until the tracker estimates are published. The cloud processing delay is the time from the LIDAR data is time stamped until it is received in the tracker node. The tracker delay is the time from LIDAR data is received in the tracker until the estimates are published.

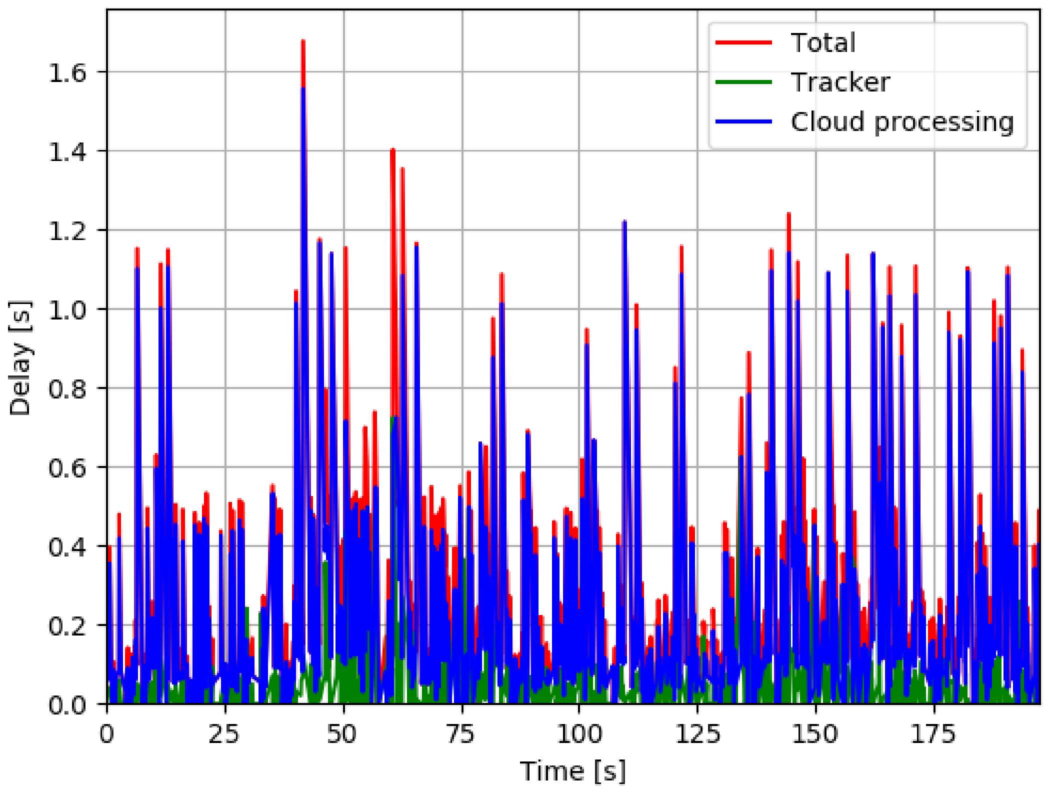

Figure 51: Processing delays in the LIDAR pipeline

In Figure 52 the delays in the image processing pipeline are visualized. The mean delay between LIDAR data capture and publishing of the tracker estimate is 215 milliseconds while the object detection mean delay is 560 milliseconds.

Both pipelines have significant noise and variations in the processing time. This is partly due to the inaccuracies of ROS time stamping presented in Figure 39. A consequence

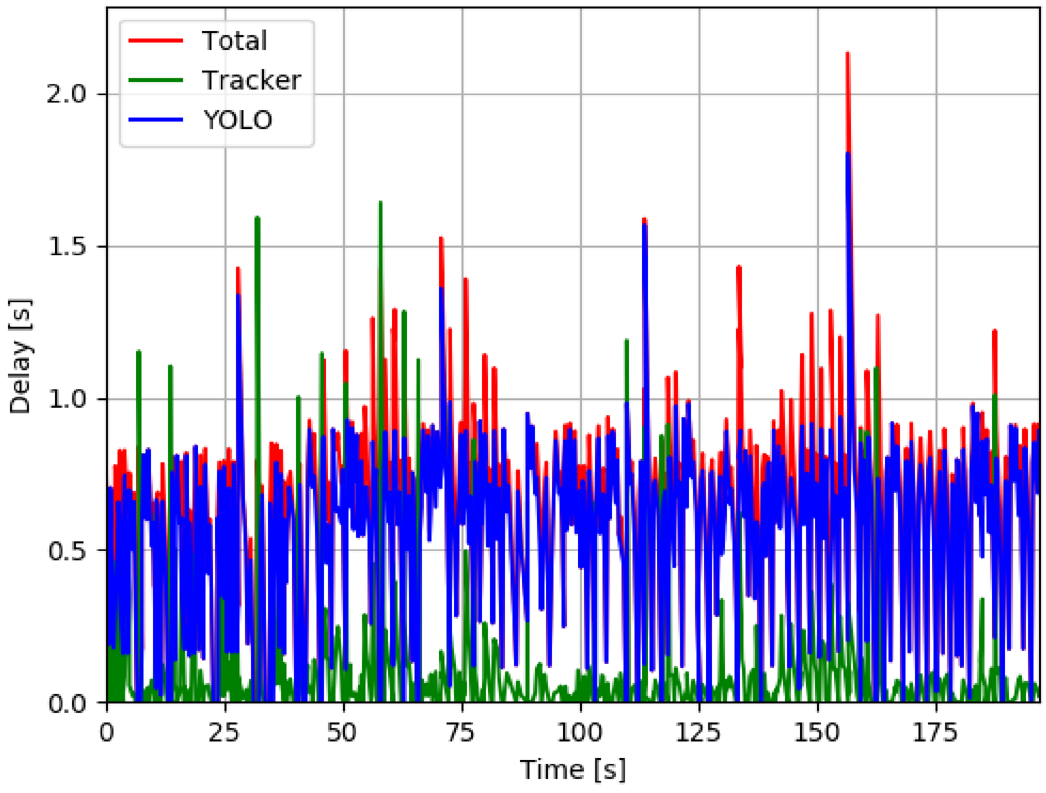

Figure 52: Processing delays in the object detection pipeline

of the difference in processing time is that messages are received asynchronously in the tracker, often with a negative time delay. When applying a discrete time Kalman filter proper processing of this late data is needed to obtain an optimal state estimate. This problem is thoroughly discussed in [23] where a negative time-update technique is presented, however applying such a technique is out of scope of this thesis. The following results will have a lower than optimal performance as a consequence of this delay.

5.2 JIPDA association delays

Initially the computationally demanding task of processing both the LIDAR and camera measurements in the tracker caused a propagating large time delay. The large amount of measurements received in the tracker caused the processing time of the tracker to go beyond the time between measurements received from the LIDAR processing pipeline. As such the overall delay in the pipeline kept propagating as shown in Figure 53. The delay kept increasing until the maximum message queue size was reached and LIDAR messages were dropped.

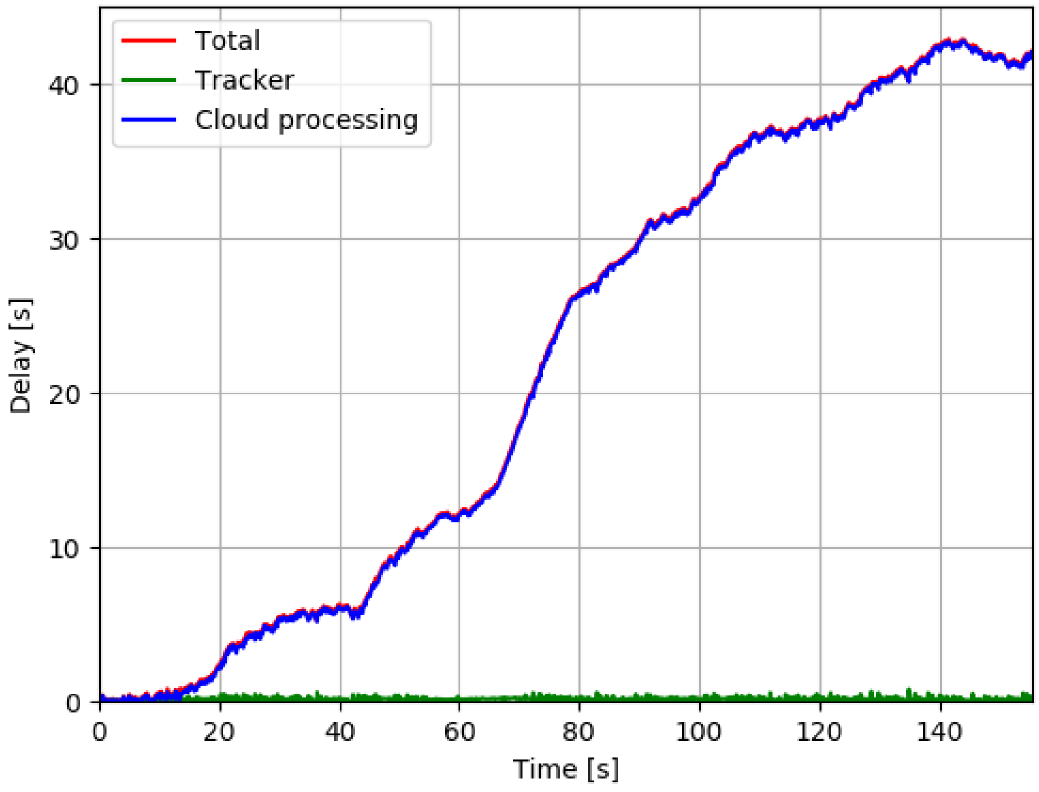

Figure 53: Delays without land filtering of the LIDAR points

As mentioned in Section 4.2.2 the large amount of data points from a LIDAR relative to AIS measurements or an RADAR in open water caused a lot of data points to be input to the tracker. Several layers of filtering of the LIDAR data has been applied to avoid such a propagating delay. In the harbour area a lot of the LIDAR points were on or around land like what is shown in Figure 54, and the amount of measurements was reduced by half after removing points that are of no interest with respect to the situational awareness pipeline.

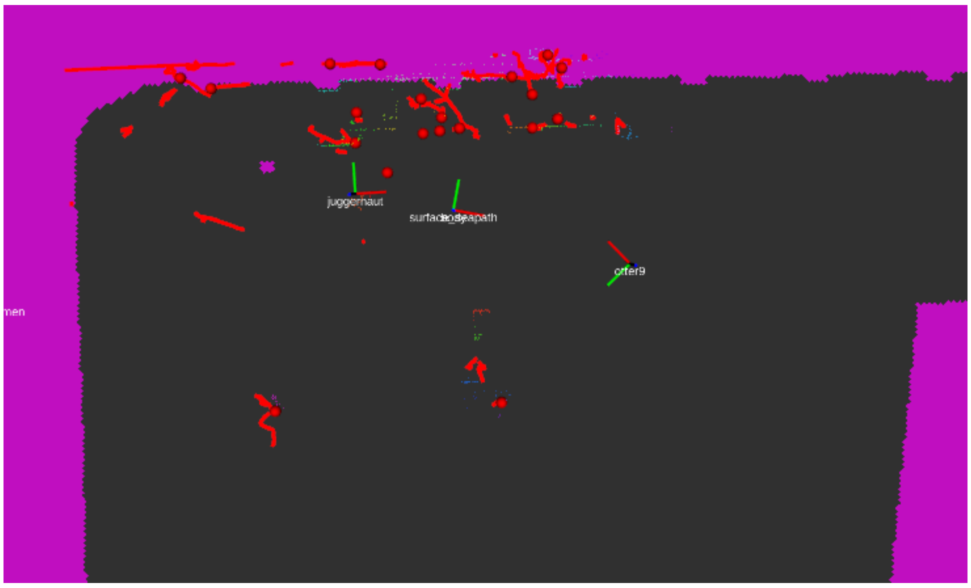

Figure 54: Tracks initiated around measurements on land

The time delay relative to the amount of association hypotheses is displayed in Figure 55.

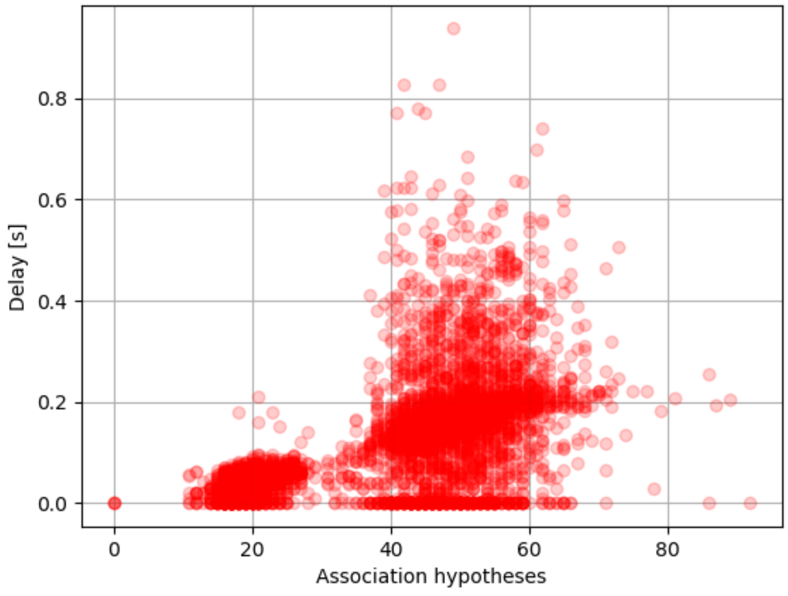

Figure 55: Time delay in tracker relative to association hypotheses

5.3 Scenario one - Overtaking

The relative range between the Juggernaut and ownship is shown in Figure 56. The goal of scenario one is to see how the tracker behaves when the target leaves and enters the field of view from the camera while staying within range of the LIDAR. We get to see how the bearing measurements from the cameras behaves on an already initiated track. Around 60 seconds into the data set is when the Juggernaut passes ownship and when the range between the two vessels is at its shortest. Ownship reaches the end of its path at 90 seconds and makes a turn while the Juggernaut starts following again at 125 seconds into the scenario.

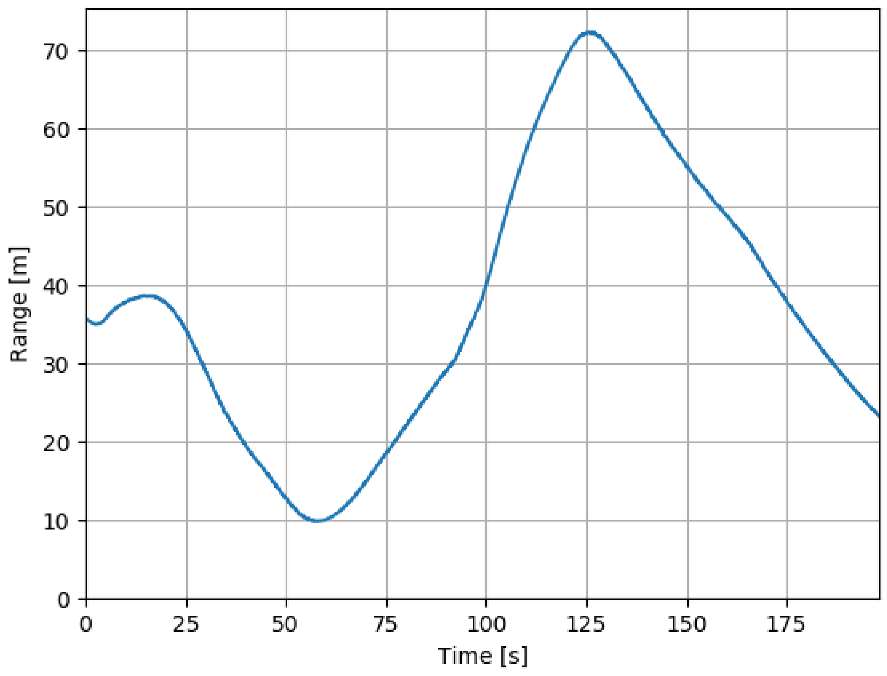

Figure 56: Range from ownship to Juggernaut in scenario one

In Figure 57 we see the estimated position of the Juggernaut using only the LIDAR and in Figure 58 we see the estimated position using both the LIDAR and the two cameras. Since the track is already initiated when the target enters the camera field of view we don't experience any earlier track initiation and since the target is out of field of view when it leaves the LIDAR range we don't expect any longer track length when introducing the bearing only measurements. The start of the track is at the top right of the two figures at coordinates $(-20, -60)$ and the track ends at $(-90, -90)$.

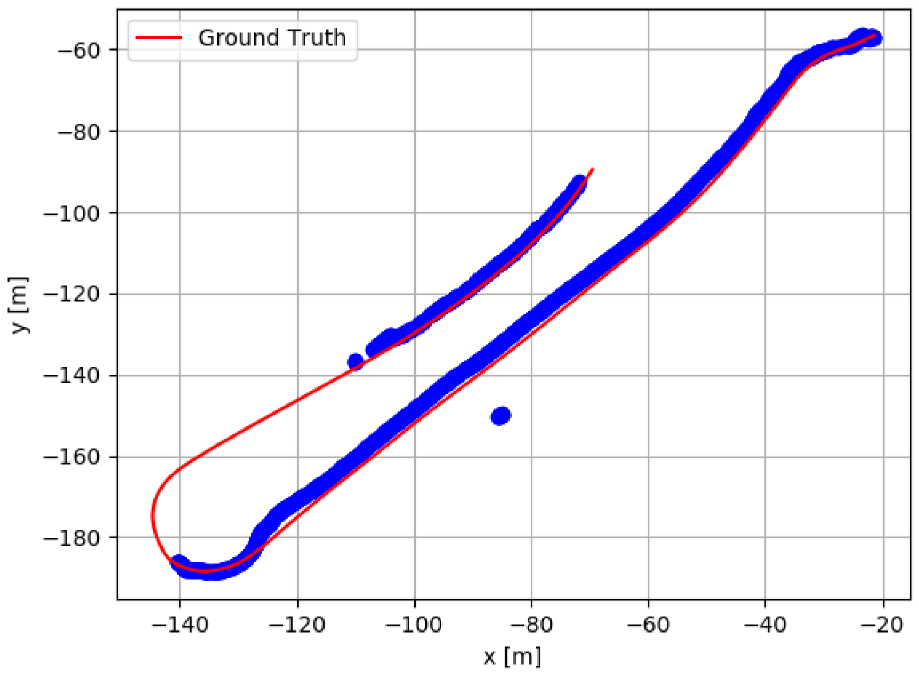

Figure 57: Estimated and ground truth position of Juggernaut using only LIDAR

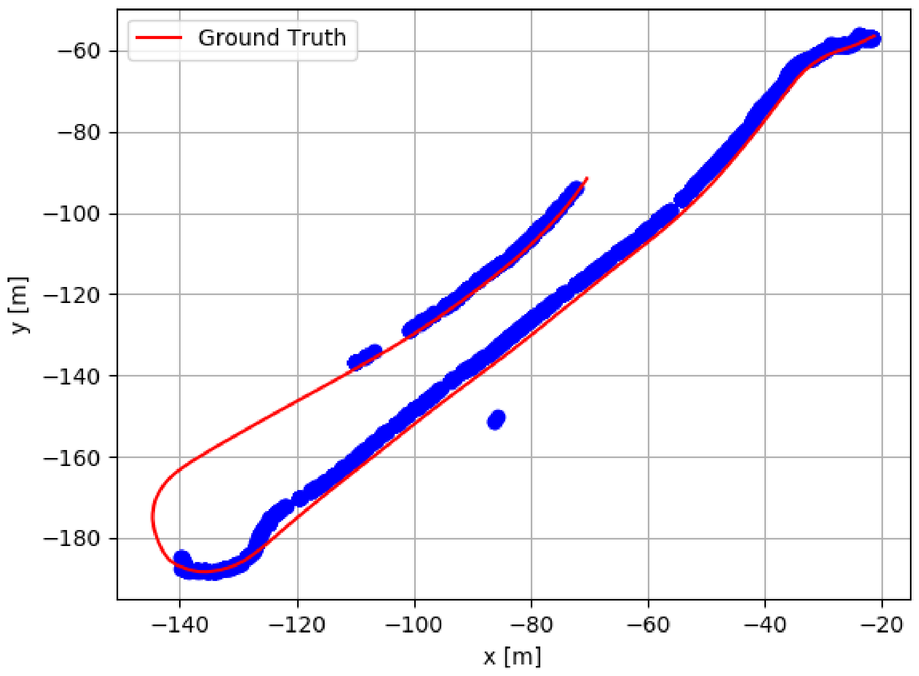

Figure 58: Estimated and ground truth position of Juggernaut using LIDAR and camera

The biggest difference between the two scenarios is a seemingly more noisy track when introducing the bearing only measurements. There is also track loss after re-initialisation. Looking at Figure 59 we see that the existence probability fluctuates a lot when adding the measurements from the camera detector. Because the bearing only measurements are at the center of the bounding box this might not always be representative of the LIDAR cluster. When the Juggernaut travels close to ownship the LIDAR detects part of the wake behind the boat. This is clustered together with the boat which causes a shift to the LIDAR detection. The bounding box seemingly also fluctuates a bit between images depending on the accuracy of the object detector and the annotations. Increasing the bearing covariance of the detections to $\sigma_\theta = 0.06$ gives better results which are presented in Section 5.4. This does however increase the size of the measurement gate and give larger clusters and more association hypotheses for the JIPDA tracker.

The detection probability P_D is a constant and in the current implementation of the tracker assumes a 360 degree field of view of the sensor. The fluctuations in existence probability when including the bearing measurements are also because of the update steps for the bearing measurements when the targets are outside of the cameras field of view. Between 80 and 90 seconds into the scenario we can see that the probability fluctuates less as the Juggernaut is within the field of view of both cameras.

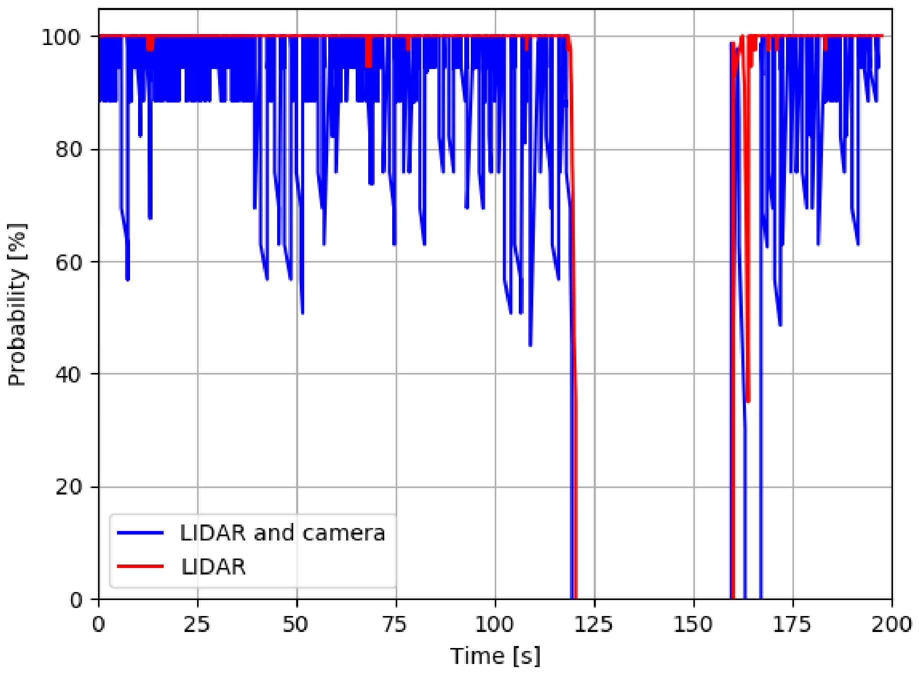

Figure 59: Existence probability of Juggernaut

The fluctuations in existence probability are reflected in the estimation error shown in Figure 60 where the position estimate fluctuates a lot when introducing the bearing measurements. The error is at its highest just after re-initiating the track when the Juggernaut is passing ownship.

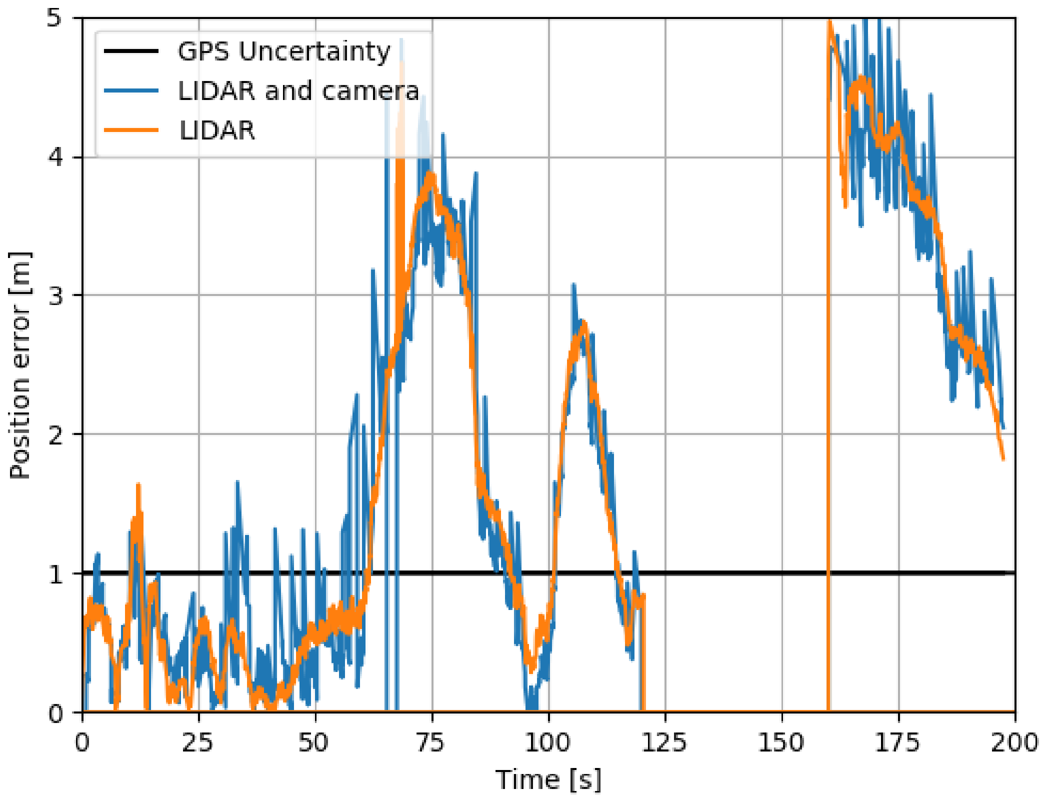

Figure 60: Estimation error of Juggernaut (with GPS position as ground truth)

Similar to the Juggernaut we get some extra noise when adding the bearing measurements to the tracker. As seen in Figure 75 we also get earlier track initiation of the Otter when using the camera detections in addition to the LIDAR. Because of the smaller size of the Otter compared to the Juggernaut we get a lot less data points when using the LIDAR. The clustering method described in Section 4.2.2 has a minimum cluster size of 20 data points and even when using the ousters full horizontal resolution of 2048 points the Otter can only be detected within 30 meters from ownship. The initial estimate has a lot higher uncertainty than when only using the LIDAR, however this is expected when the track is initiated using bearing only measurements as described in Section 4.2.1. The track does converge more quickly and we get a lower error when the Otter is detected by the LIDAR.

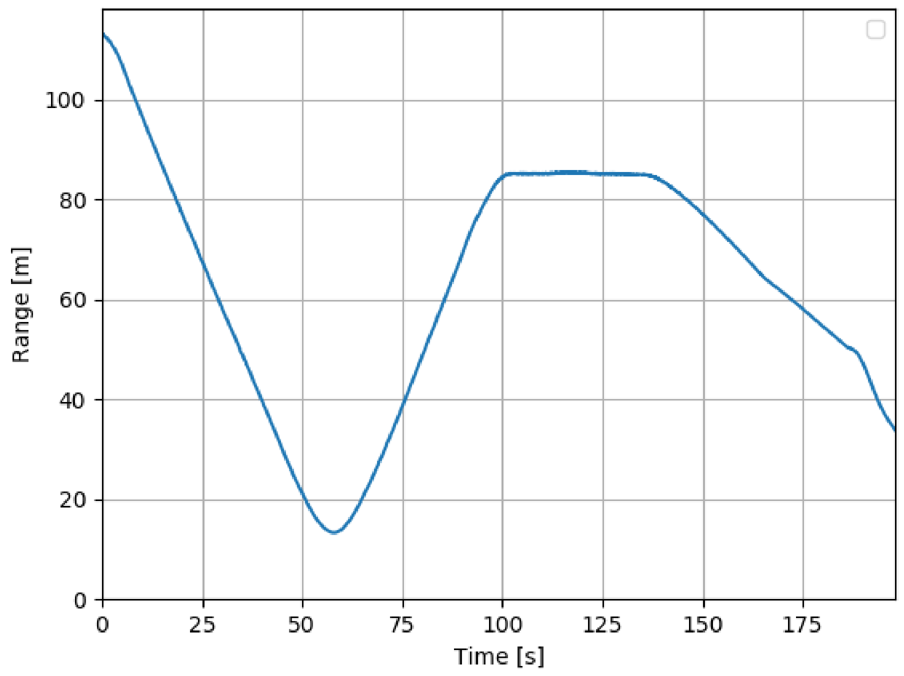

Figure 61: Range to Otter 9 in scenario one

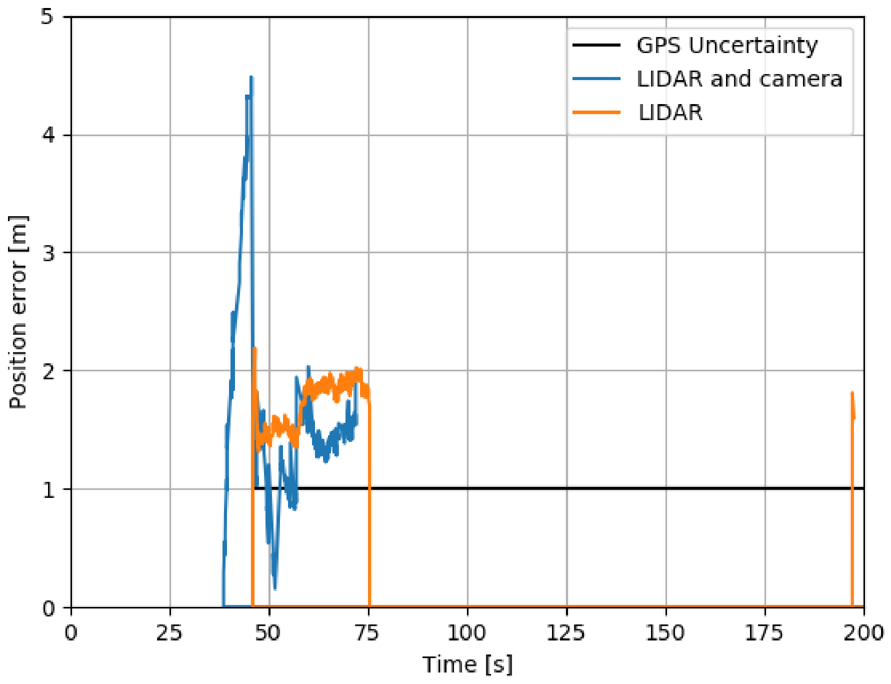

Figure 62: Estimation error of Otter 9 (with GPS position as ground truth)

Looking at the mode probabilities for the Otter (Figures 63 and 64) we see that the initial probability when using bearing only measurements is shifted towards the CT model. This stabilizes more when the Otter gets detected by the LIDAR, however the probabilities remain slightly shifted compared to only using LIDAR measurements.

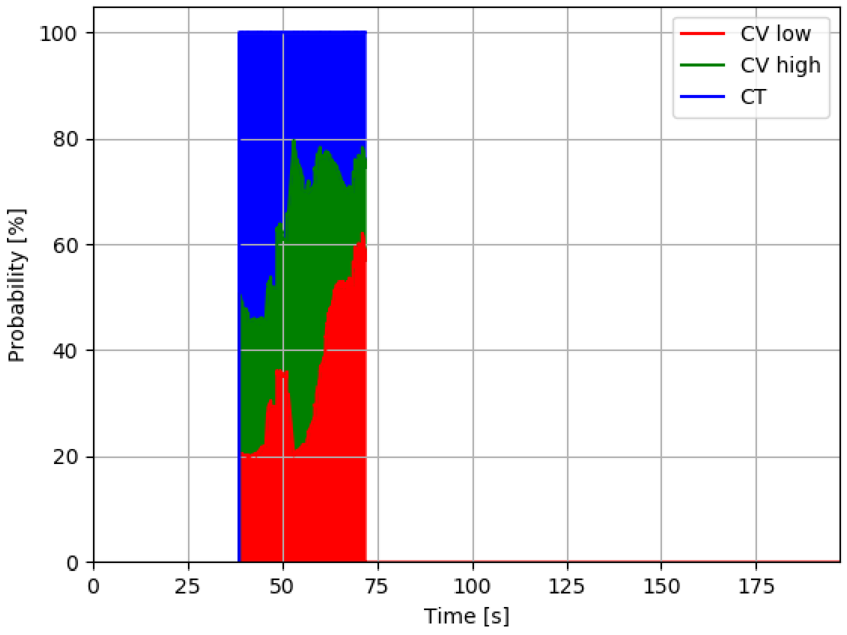

Figure 63: Mode probabilities for Otter 9 using LIDAR and camera

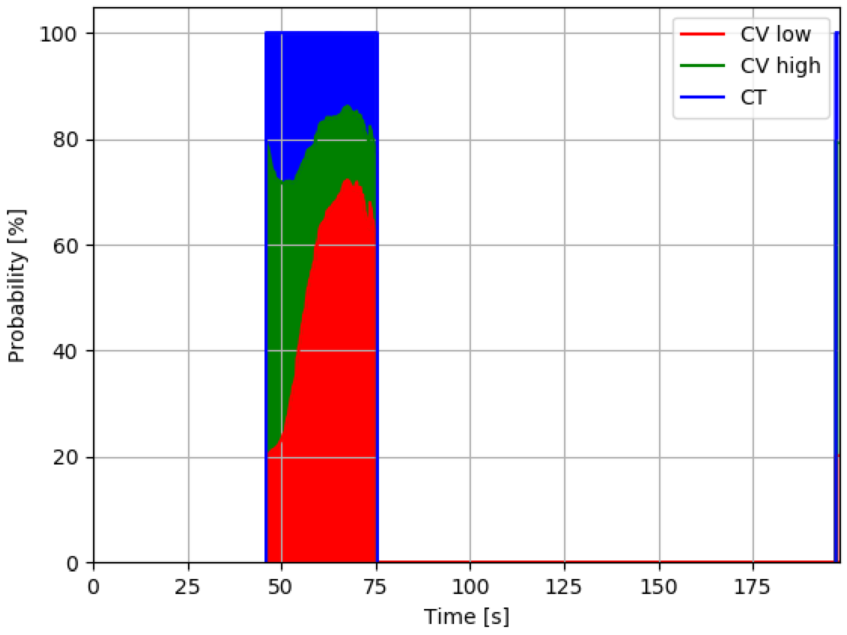

Figure 64: Mode probabilities for Otter 9 using only LIDAR

5.4 Scenario two - Occlusion of target

The goal of scenario two is to see how the tracker behaves when the Otter becomes occluded by the Juggernaut. The pass happens from 60 to 65 seconds into the scenario when the Juggernaut is around 20 meters away from ownship and the Otter is a bit over 30 meters away. In Figures 66 and 67 the Juggernaut starts in the top left corner at $(-160, -65)$, makes a turn at 75 seconds into the scenario and end its path at $(-128, -165)$.

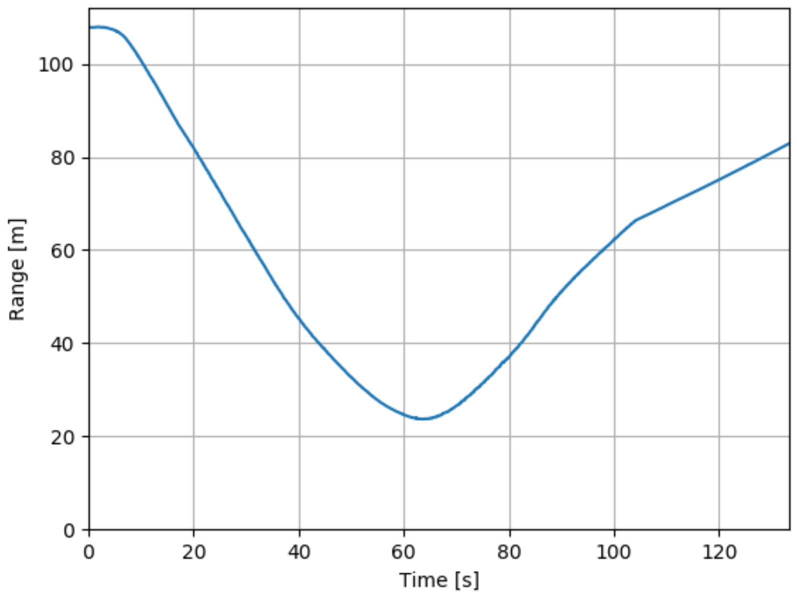

Figure 65: Range from ownship to Juggernaut in scenario two

As seen on the next page the addition of the camera detector does not greatly improve the position estimate of the Juggernaut. The precision with the multi sensor tracker is however higher compared to scenario one after increasing the noise covariance for the bearing measurements. The measurements does become a lot more scattered after ownship has made its turn and the target is no longer within the field of view of the cameras.

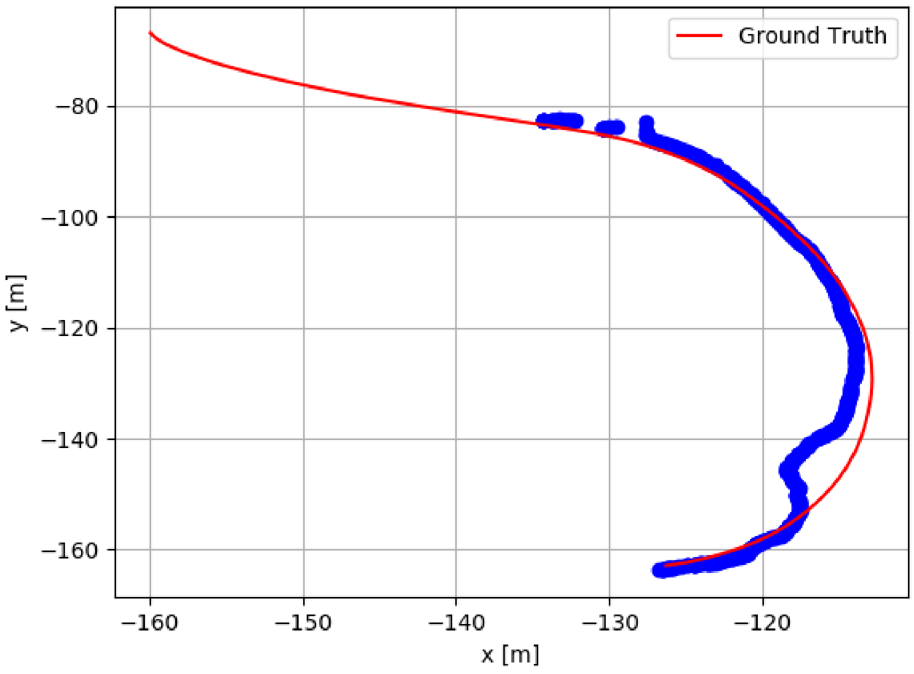

Figure 66: Estimated and ground truth position of Juggernaut using only LIDAR

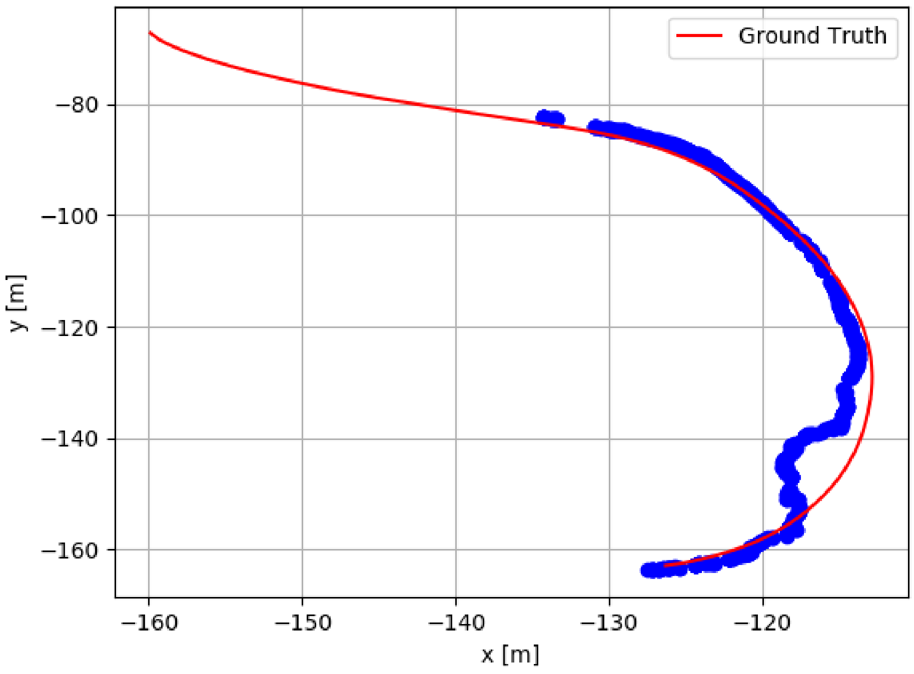

Figure 67: Estimated and ground truth position of Juggernaut using LIDAR and camera

Changing the bearing covariance also decreases the noise in the existence probabilities. Until the Juggernaut makes a turn at around 75 seconds there is a high probability of existence. The existence probability is however degenerated when the Juggernaut is no longer in the cameras field of view.

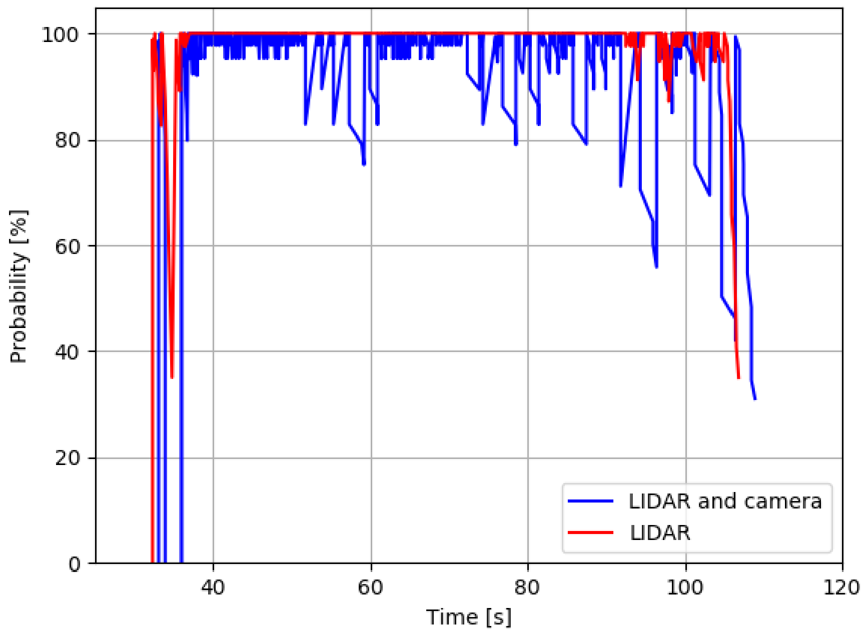

Figure 68: Existence probability of Juggernaut

Unfortunately the same problem occurs in scenario two with respect to the Otter. The small size makes it hard to detect for the LIDAR and the camera detection are not enough to initiate a track on their own. Due to the increased measurement noise (and hence a larger validation gate) it instead takes slightly longer to initiate the track when the Otter emerges behind the Juggernaut. Similar as the Juggernaut we also get an increased estimation error when ownship makes a turn and the Otter is no longer within the cameras field of view. The error is however still within the accuracy of the GPS.

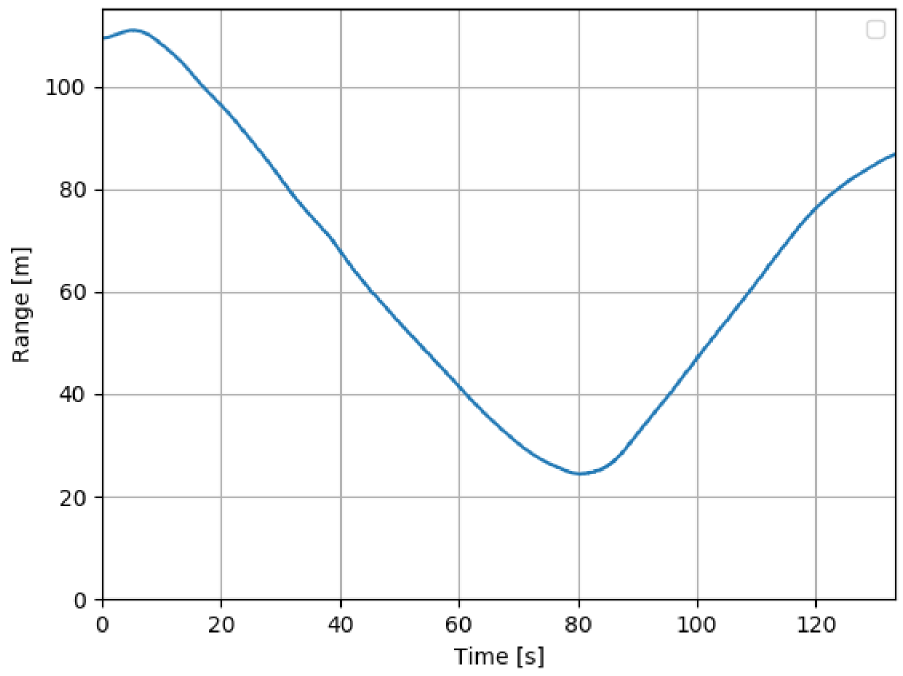

Figure 69: Range to Otter 9 in scenario two

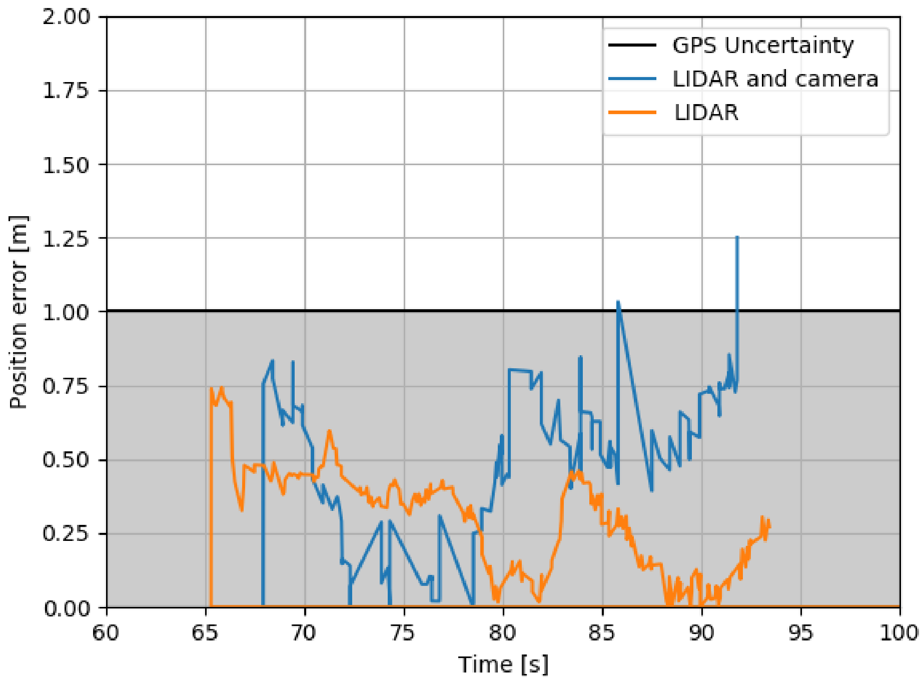

Figure 70: Estimation error of Juggernaut (with GPS position as ground truth)

The problems with detecting the Otter is shown in Figure 71 displaying the raw point cloud when the Otter is passing the Juggernaut. The coordinate system closest to the camera is ownship and the one furthest away is the target Otter. While the Juggernaut and surrounding boats are detected by a lot of points the Otter is only detected as a small line. This is filtered out as noise due to the minimum cluster size in the LIDAR filtering. Attempts were made to decrease the cluster size further to detect the Otter, although this in turn caused a lot of noise to be input to the tracker. This required a larger clutter density which in the end did not give cause any earlier detection of the Otter.

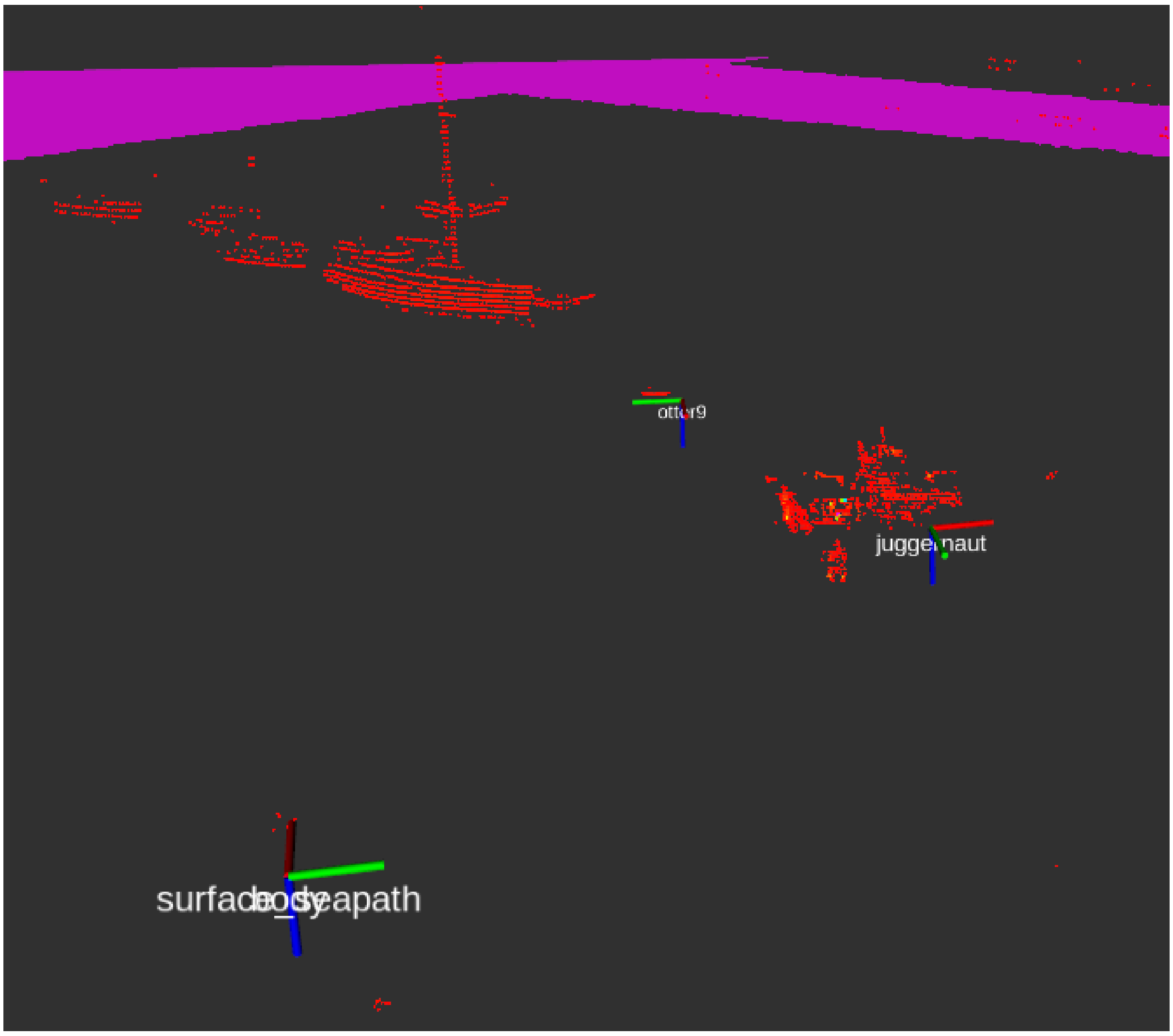

Figure 71: Raw point cloud showing the two target boats

5.5 Scenario three - Passing of target

In scenario three the goal is to test the tracker when the Juggernaut is located in front of ownship and within field of view of both the cameras. This happens while the Juggernaut is relatively far away from the Otter and at the edge of the LIDAR working range. The Juggernaut starts at $-138, -162$ in the bottom left in Figures 74 and 73. It maneuvers to $(-142, -79)$ and passes ownship between 55 and 60 seconds into the scenario.

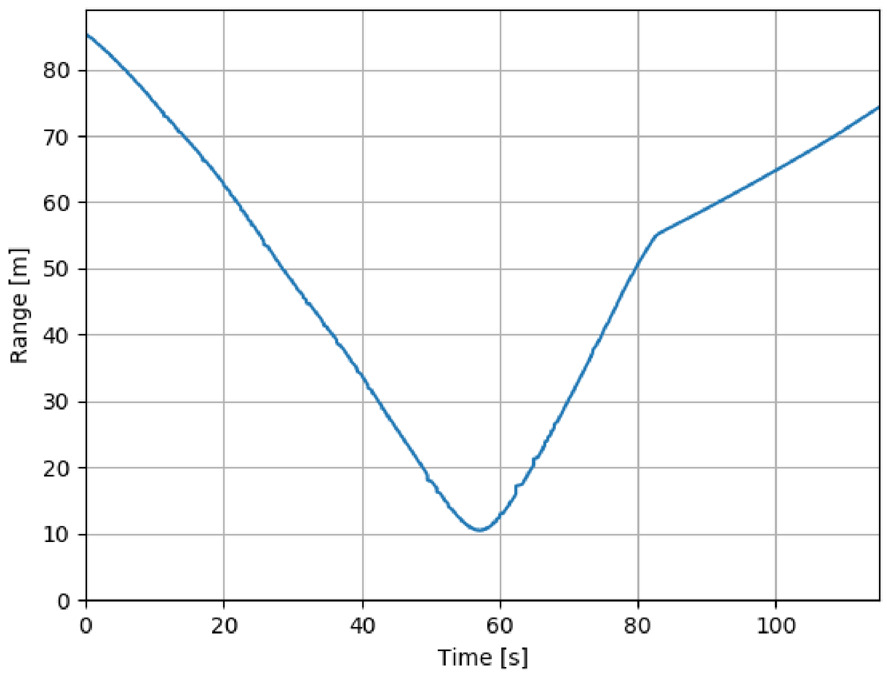

Figure 72: Range from ownship to Juggernaut in scenario three

In this scenario we are able to maintain a track of the Juggernaut a lot easier when fusing the measurements from the cameras and the LIDAR. In Figure 73 we see that the tracker receives enough LIDAR-measurements to initiate a track a few meters into the scenario and in Figure 74 we see that the camera measurements are able to maintain the track.

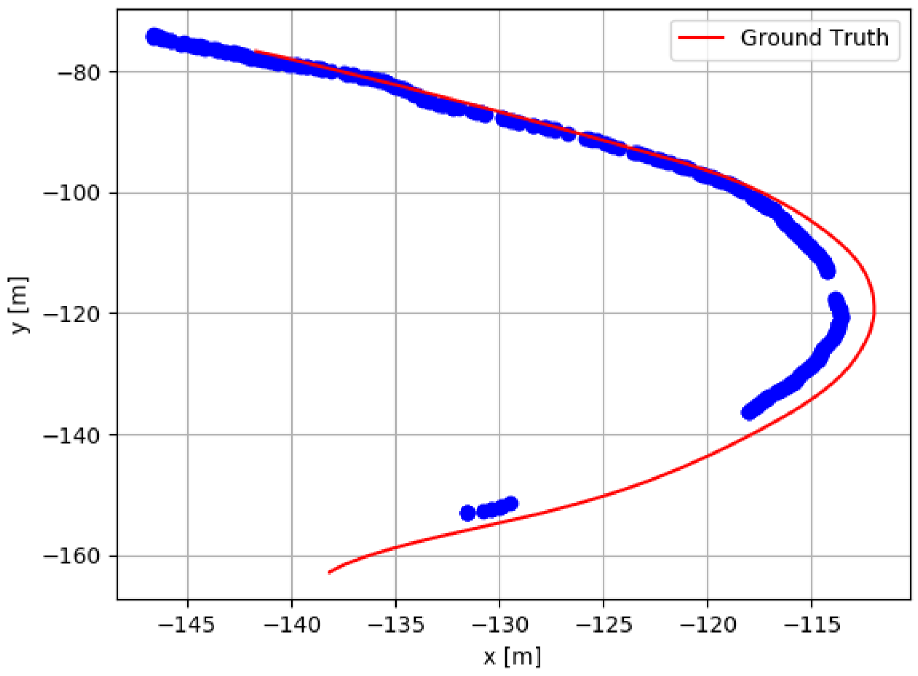

Figure 73: Estimated and ground truth position of Juggernaut using LIDAR

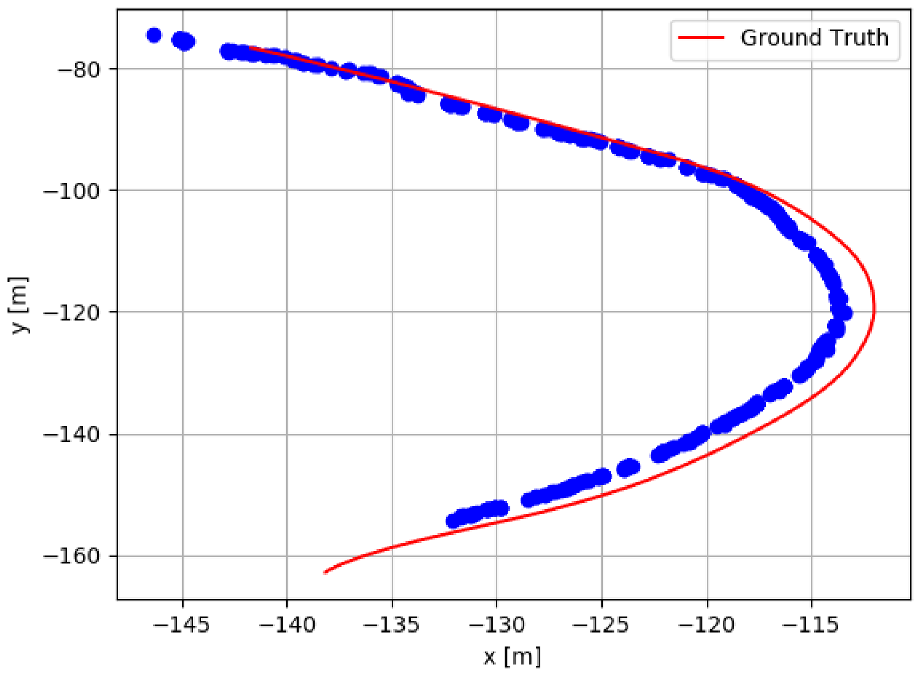

Figure 74: Estimated and ground truth position of Juggernaut using LIDAR and camera

In figure Figure 75 we see that the initial probability is shifted towards the constant turn rate model when the track is initiated and maintained by the camera measurements. We also see that the mode probabilities are shifted more towards the high process noise constant velocity model compared to the one with low process noise. Around 40 seconds into the scenario we get more updates from the LIDAR which gives a higher probability for the target to follow the CV model with low process noise.

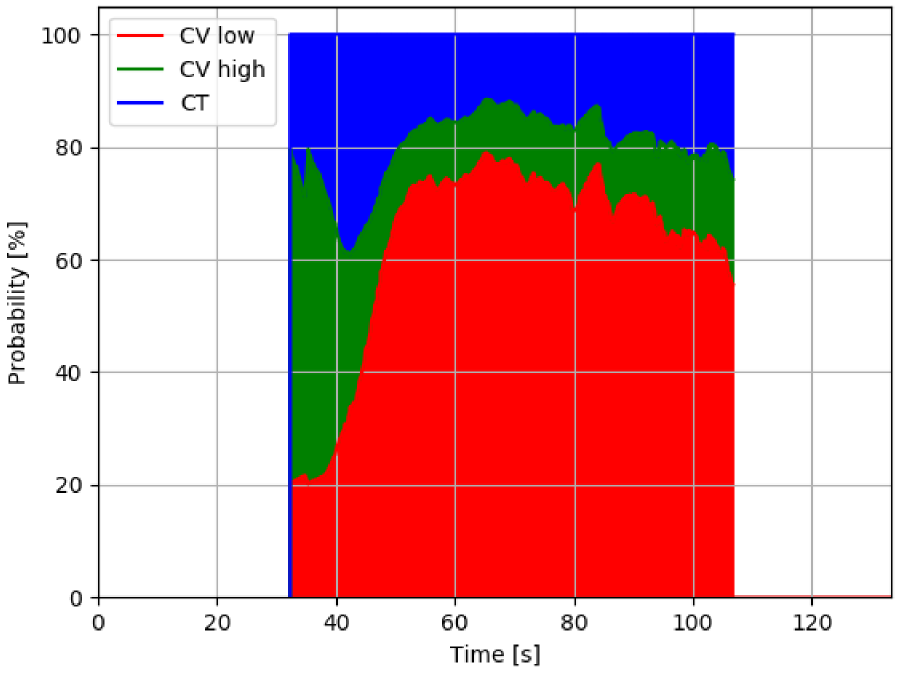

Figure 75: Mode probabilities of the Juggernaut track using LIDAR and camera

In Figures 76 and 77 the Otter starts from the left at $(-62, -112)$ and moves left to $(-93, -111)$. Unlike the Juggernaut the track of the Otter is not improved when fusing the camera measurements with the LIDAR. It takes longer for the track to initialise and it dies out earlier than with only the LIDAR. Because the Otter is always outside of the cameras field of view the lack of measurements from the cameras rather give a decrease in existence probability. This is similar to what is seen in the other scenarios when the target is outside the cameras field of view.

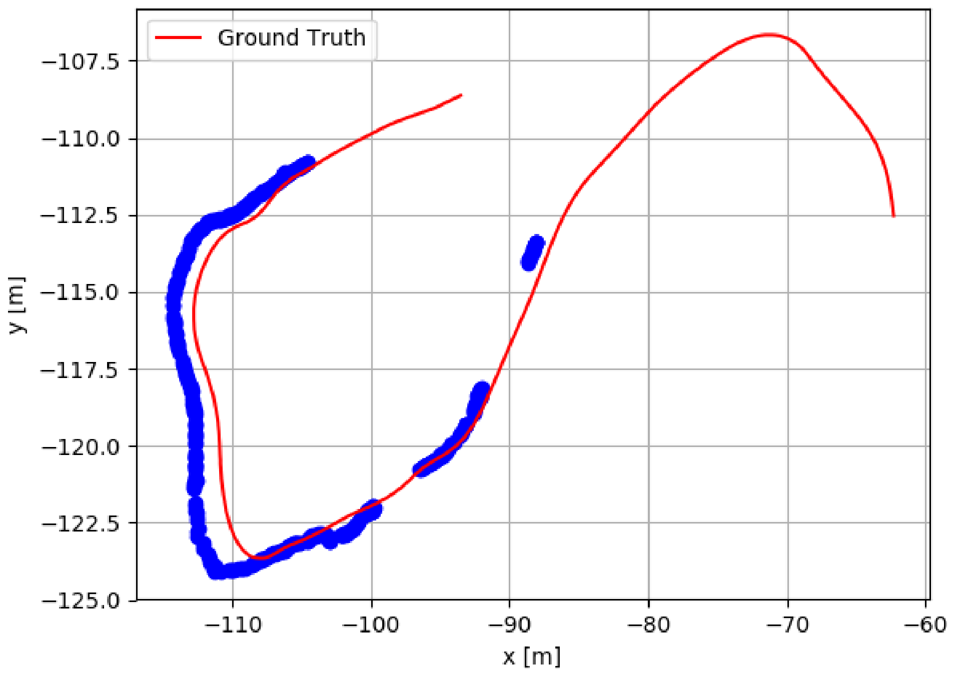

Figure 76: Estimated and ground truth position of Otter 9 using LIDAR

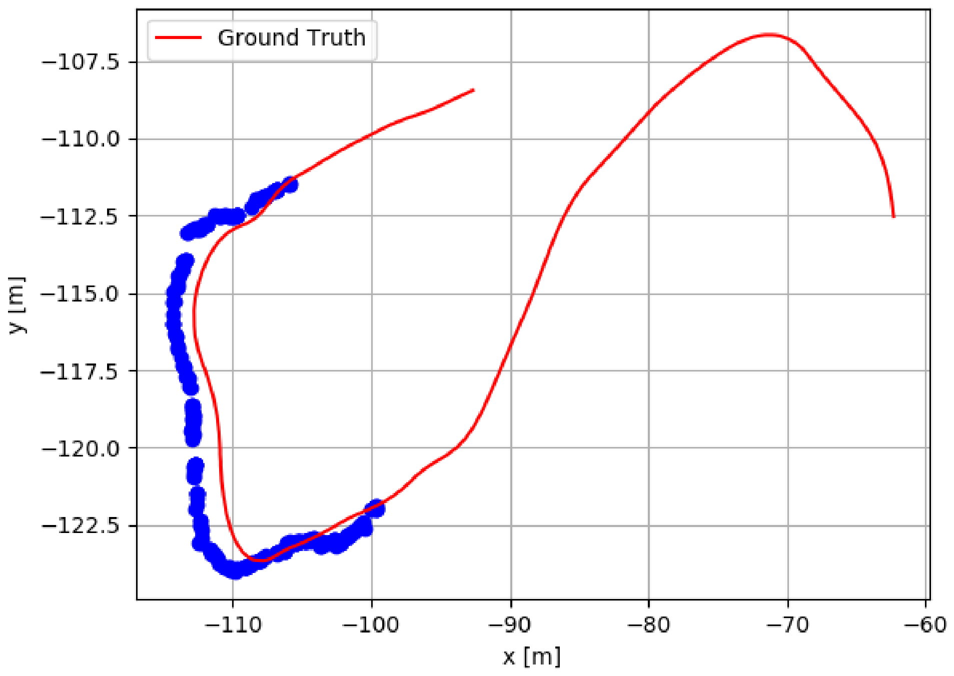

Figure 77: Estimated and ground truth position of Otter 9 using LIDAR and camera

5.6 Loss of LIDAR measurements

To simulate a loss of LIDAR data the measurements past a certain timestamp in the data set is removed. In scenario two the LIDAR data is removed when the two targets are passing. At this point the track of the Juggernaut is well initiated with a high existence probability. From Figure 78 we see that the camera is able to maintain a reliable track for five seconds after the LIDAR measurements are cut at 48 seconds. The probability does not decrease linearly which shows us that the bearing measurements are being processed.

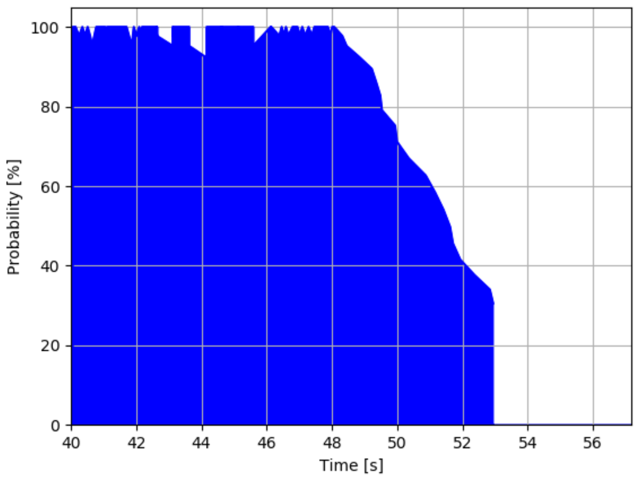

Figure 78: Existence probability after LIDAR dropout

Similar to the other scenarios we see that the probabilities when relying on bearing measurements are shifted towards the CT model.

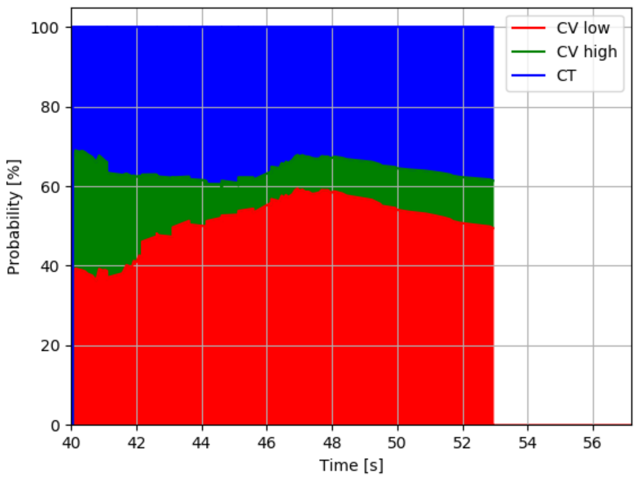

Figure 79: Mode probabilities during sensor dropout

In Figures 80 and 81 we see a large amount of detections around the target at this point in time. Several of these measurements are gated to the target and instead of maintaining the track the covariance grows larger and the track is terminated due to the large uncertainty.

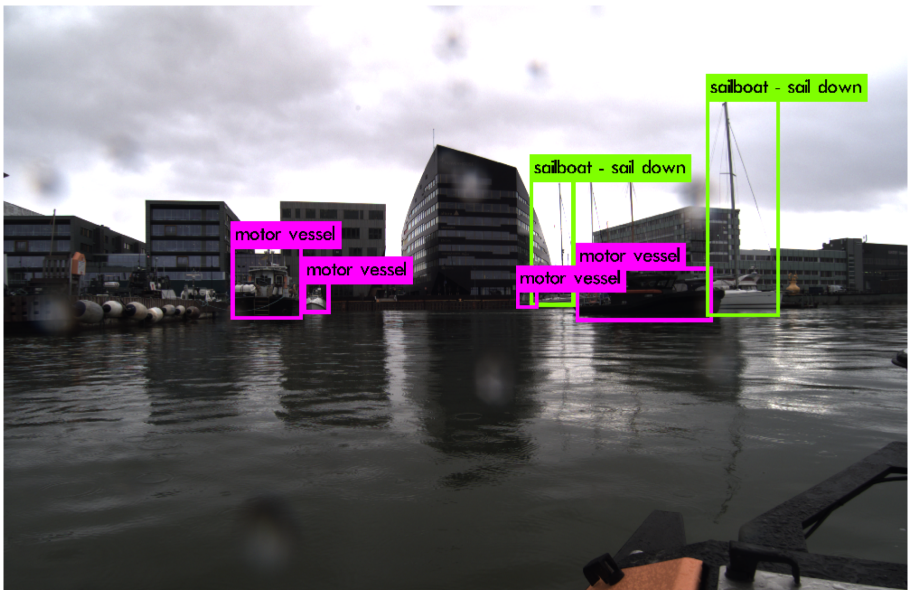

Figure 80: Camera detections a few seconds after LIDAR dropout

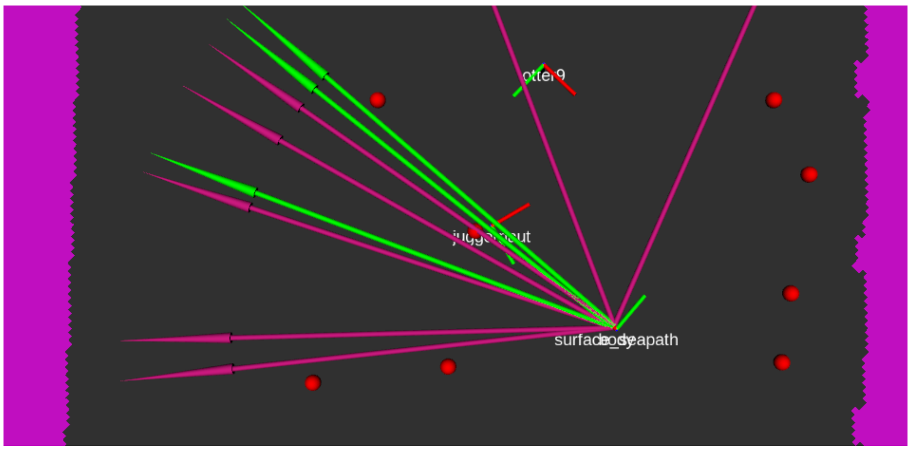

Figure 81: Camera detections a few seconds after LIDAR dropout

Sensor dropout was also simulated in the first scenario. This time the LIDAR data drops out at 78 seconds, a few seconds after the Juggernaut has passed ownship. The tracker is able to maintain the track for nine seconds before it is terminated as shown in Figure 82. This is longer than in the previous scenario, however we would expect a well initiated track to be maintained for longer with the bearing measurements. Looking at Figure 83 we see that there are three measurements gated with the track estimate. Similar to the previous scenario they add uncertainty to the estimate and cause track divergence, however this takes longer as there are less measurements within the validation gate compared to Figure 81.

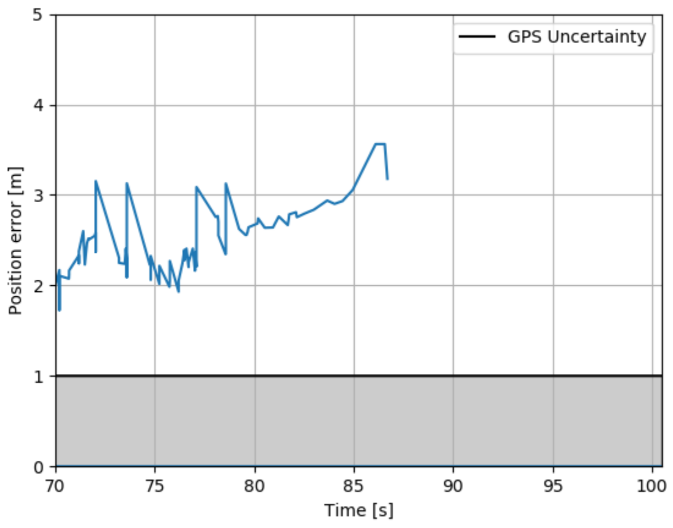

Figure 82: Position estimate error compared to GPS position

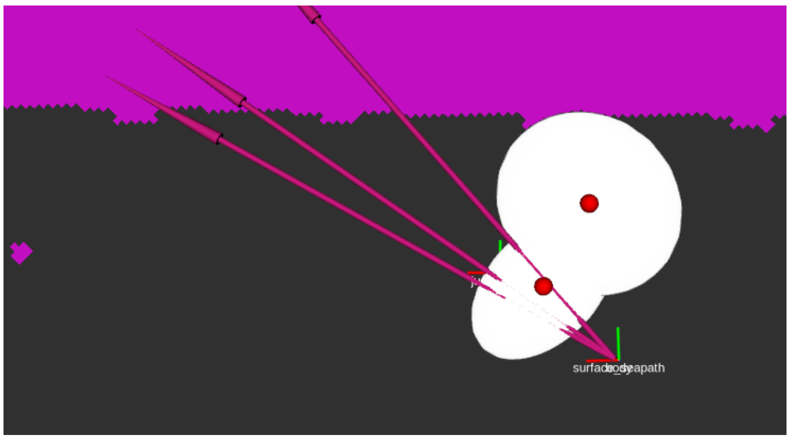

Figure 83: Position estimate error compared to GPS position

6 Conclusion and future work

6.1 Conclusion

Data from three different test scenarios has been collected. In these scenarios two targets are equipped with GPS receivers and maneuver in patterns that provide different challenges for the tracker. To avoid poor sensor data cluttering our analysis, extra data sets have been collected in the same areas for training the object detector.

Reliable sensor data has been obtained for both the LIDAR and the camera and bearing measurements are extracted from the detection of a neural network. For the camera images an object detector has been trained on over 4000 annotated images to obtain a mAP of over 96 %. Processing techniques have been implemented to reduce the amount of points and to cluster the LIDAR data into objects of interest to the tracking pipeline. The tracker has been restructured into a ROS-node and implemented into the situational awareness pipeline. It is extended to contain sensor specific parameters and modified to support bearing only measurements from the object detector.

The GPS positions of the targets are compared with the estimates and give good results. When introducing the bearing only measurements we get good estimates when the targets are within the cameras field of view, however the performance is degenerated when the targets are outside of the cameras surveillance range because of the tracker assuming a 360 degree field of view from its sensors. The camera helps to initiate and maintain a track when the targets are within the cameras field of view. In addition LIDAR dropouts have been simulated and the camera is able to maintain the track for a few seconds on its own, but the gating method should be improved to further increase the performance.

6.2 Future work

The current implementation of the bearing only measurements can be improved by a more dynamic gating method, as the current implementation does not take into account the velocity covariance or the range to the target. A well known problem with a bearing only tracking is incorrect convergence of the covariance ellipses. Adding a range parameterised Kalman filter as shown in "Improved bearings-only target tracking with iterated Gaussian mixture measurements [49]" is shown to give more precise convergence in "Recursive Bayesian estimation: bearings-only applications [27]". Adding methods for handling asynchronous messages from the sensors will increase the performance of the tracker due to the different processing delays for the different sensors [20]. Changing the tracker structure to avoid the mutex and allow processing several sets of measurements in parallel will increase the efficiently of the tracker.

YOLO uses a convolutional layer to learn and predict the most likely locations of bounding boxes in an image. Adding another layer to the network which proposes possible locations using LIDAR measurements and the location of existing tracks could help increase the accuracy of a neural network. Using INS data and the rotation of own-ship relative to existing targets could be used to propose bounding boxes and ensure new measurements of existing targets. While the object detector used in this thesis has a very high accuracy it is not reasonable to assume such performance of an object detector in a real world scenario. This object detector is over-fitted to the targets pertaining to the scenarios. Extending an object detector with an extra layer for bounding box prediction using INS data would increase the reliability and stability of the bearing only measurements once a track has been initiated.

Currently the output from the object detector is a bearing measurement relative to the center of the bounding box. Extending this to include the horizontal edges of the bounding box would give more information about a targets size and possibly also the distance to an object. Adding some bounding parameters like the maximum width of different classes from the neural network could help gauge distance. Adding detection classes for different types of boats could be used to limit the size of the objects which could be used to triangulate range. Different detection classes for different types of boats can also be used to attribute dynamical properties to the tracks with limits to speed and acceleration.

In Figures 48 and 49 we can see that the docks in the center of the map are interpreted by the Euclidean segmenting method as a detection which is sent into the target tracking pipeline. Extending the ground segmentation to include maps created dynamically by a SLAM algorithm would reduce the amount of false inputs to the tracker. Sea charts are often changed, new docks are added (like those in Trondheim harbour) and in a congested environment we get a lot of false tracks from nearby docks and land areas. As shown in Figure 53 the processing time increases exponentially from the amount of association hypotheses.

References

[1] Sigurd Albrektsen and Tor Johansen. User-configurable timing and navigation for uavs. Sensors, 18:2468, July 2018.

[2] M. Arulampalam, Branko Risti, N. Gordon, and T. Mansell. Bearings-only tracking of manoeuvring targets using particle filters. EURASIP Journal on Advances in Signal Processing, 2004, Nov 2004.

[3] R. Aufrere, J. Gowdy, C. Mertz, C. Thorpe, C. Wang, and T. Yata. Perception for collision avoidance and autonomous driving. Mechatronics, 13:1149–1161, 2003.

[4] Dimitri Bertsekas. The auction algorithm: A distributed relaxation method for the assignment problem. Annals of Operations Research, 14:105–123, Dec 1988.

[5] H. A. P. Blom and Y. Bar-Shalom. The interacting multiple model algorithm for systems with markovian switching coefficients. IEEE Transactions on Automatic Control, vol. 33, no. 8, page 780–783, 1988.

[6] E Brekke. Fundamentals of Sensor Fusion - Target tracking, navigation and SLAM. NTNU, Jan 2020. [Online; accessed Feb 5, 2021] `https://folk.ntnu.no/edmundfo/msc2019-2020/sf13chapters.pdf`.

[7] E Brekke, Hem A.G., and Tokle L.-C. N. The vimmjipda: Hybrid state formulation and verification on maritime radar benchmark data. Global OCEANS 2020 Online Proceedings, 2020.

[8] E Brekke, E Wilthil, Bjørn-Olav Eriksen, D Kufoalor, Øystein Helgesen, I Hagen, Morten Breivik, and Tor Johansen. The autosea project: Developing closed-loop target tracking and collision avoidance systems. Journal of Physics: Conference Series, 1357:012020, Oct 2019.

[9] Edmund Førland Brekke, Audun Gullikstad Hem, and Lars-Christian Ness Tokle. The vimmjipda: Hybrid state formulation and verification on maritime radar benchmark data. In Global Oceans 2020: Singapore – U.S. Gulf Coast, pages 1–5, 2020.

[10] Cambridge University Press. Single-link and complete-link clustering. `https://nlp.stanford.edu/IR-book/html/htmledition/single-link-and-complete-link-clustering-1.html`, April 2009. [Online; accessed June 13, 2021].

[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. IEEE International Conference on Robotics and Automation (ICRA), pages 1836–1843, May 2014.

[12] Jiri de Vos, Robert G. Hekkenberg, and Osiris A. Valdez Banda. The impact of autonomous ships on safety at sea – a statistical analysis. Reliability Engineering & System Safety, 210:107558, 2021.

[13] dhruv's space. Ml basics #4: Replace negatives with zeros! `https://dhruvs.space/posts/ml-basics-issue-4`, Sept 2019. [Online; accessed May 4, 2021].

[14] Christoph Domke and Quentin Potts. Lidars for self-driving vehicles: a technological arms race. `https://www.automotiveworld.com/articles/lidars-for-self-driving-vehicles-a-technological-arms-race/`, Sept 2020. [Online; accessed March 28, 2021].

[15] Katherine Ellis, Suneeta Godbole, Simon Marshall, Gert Lanckriet, John Staudenmayer, and Jacqueline Kerr. Identifying active travel behaviors in challenging environments using gps, accelerometers, and machine learning algorithms. Frontiers in public health, 2:36, Apr 2014.

[16] enginBozkurt. Lidar obstacle detection. `https://github.com/enginBozkurt/LidarObstacleDetection`, Sept 2019. [Online; accessed Feb 6, 2021].

[17] Sondos Fadl and Noura Semary. Robust copy-move forgery revealing in digital images using polar coordinate system. Neurocomputing, June 2017.

[18] Gary Chan and Joshua Whitley. Lidar perception. `https://github.com/LidarPerception/common_lib`, 2019. [Online; accessed June 13, 2021].

[19] Didrik Grove. Data synchronization in maritime target tracking. Technical report, Norwegian University of Science and Technology, Jan 2021. Project thesis.

[20] Thomas Hanselmann and Mark Morelande. Multiple target tracking with asynchronous bearings-only-measurements. In 2007 10th International Conference on Information Fusion, pages 1–8, 2007.

[21] Richard Hartley and Andrew Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, 2 edition, 2004.

[22] Audun Gullikstand Hem. Maritime multi-target tracking with radar and asynchronous transponder measurements. Master's thesis, Norwegian University of Science and Technology, Jan 2021. Unpublished.

[23] Richard Hilton and David Martin. Tracking with time-delayed data in multisensor systems. page 56, August 1993.

[24] Heather Hinkel, John J. Zipay, Matthew Strube, and Scott Cryan. Technology development of automated rendezvous and docking/capture sensors and docking mechanism for the asteroid redirect crewed mission. In 2016 IEEE Aerospace Conference, 2016.

[25] NAVICO Inc. Hs60 gps compass user guide. `http://busse-yachtshop.de/pdf/simrad-hs60-manual.pdf`, 2014. [Online; accessed May 1, 2021].

[26] Jet New. Gaussian mixture models with tensorflow probability. `https://medium.com/analytics-vidhya/gaussian-mixture-models-with-tensorflow-probability-125315891c22`, June 2020. [Online; accessed June 13, 2021].

[27] R. Karlsson and F. Gustafsson. Recursive bayesian estimation: bearings-only applications. IEE Proceedings - Radar Sonar and Navigation, page 305–313], 2005.

[28] Jelena Kocić, Nenad Jovičić, and Vujo Drndarević. Sensors and sensor fusion in autonomous vehicles. In 2018 26th Telecommunications Forum (TELFOR), pages 420–425, 2018.

[29] Vegard Kvamsgård. Fusion between camera and lidar for autonomous surface vehicles. Master's thesis, Norwegian University of Science and Technology, July 2018.

[30] Jan Nic. Langfeldt. Boatman's drivers test. `https://www.seileren.no/wp-content/uploads/2014/08/English-baatforer-text.pdf`, Aug 2014. [Online; accessed June 14, 2021].

[31] Chang-Hun Lee. Observability analysis of advanced guidance laws with bearing-only measurement. IFAC Proceedings Volumes, 43:136–141, Sept 2010.

[32] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander Berg. Ssd: Single shot multibox detector. In SSD: Single Shot MultiBox Detector, volume 9905, pages 21–37, Oct 2016.

[33] M. Mahlisch, R. Schweiger, W. Ritter, and K. Dietmayer. Sensorfusion using spatiotemporal aligned video and lidar for improved vehicle detection. IEEE Intelligent Vehicles Symposium, page 424–429, 2006.

[34] MathWorks. What is camera calibration? `https://www.mathworks.com/help/vision/ug/camera-calibration.html`, 2020. [Online; accessed May 15, 2021].

[35] OpenStax College. Polar coordinates. `https://courses.lumenlearning.com/precalctwo/chapter/polar-coordinates/`, 2021. [Online; accessed Mar 26, 2021].

[36] Point Cloud Library. Euclidean cluster extraction. `https://pcl.readthedocs.io/en/latest/cluster_extraction.html`, 2021. [Online; accessed May 27, 2021].

[37] C. Premebida, O. Ludwig, , and U. Nunes. Lidar and vision-based pedestrian detection system. Journal of Field Robotics, 26:696–711, 2009.

[38] Joseph Redmon. Darknet: Open source neural networks in c. `http://pjreddie.com/darknet/`, 2013-2016. [Online; accessed June 13, 2021].

[39] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 779–788, 2016.

[40] AutoPilot Review. Elon musk on cameras vs lidar for self driving and autonomous cars. `https://www.youtube.com/watch?v=HM23sjhtk4Q`, 2019. [Online; accessed Feb 27, 2021].

[41] Kongsberg Seatex. minimru - the compact reference unit. `https://www.kongsberg.com/globalassets/maritime/km-products/product-documents/datasheet_minimru.pdf`, Dec 2020. [Online; accessed Feb 12, 2021].

[42] SentiSystems. Sentiboard documentation. `https://gitlab.senti.no/senti/senti-doc/`, June 2021. [Online; accessed Feb 25, 2021].

[43] Sjøfartsdirektoratet. Føringer i forbindelse med bygging eller installering av automatisert funksjonalitet, med hensikt å kunne utføre ubemannet eller delvis ubemannet drift. `https://www.sdir.no/contentassets/2b487e1b63cb47d39735953ed492888d/rsv-12-2020.pdf?t=1619765992454`, Aug 2020. [Online; accessed June 14, 2021].

[44] Norwegian Maritime Authority (Sjøfartsdirektoratet). Regulations of 1 december 1975 no. 5 for preventing collisions at sea (rules of the road at sea), Dec 1975.

[45] Knut Turøy. Sensor Fusion of Camera-LIDAR for ReVolt. Master's thesis, Norwegian University of Science and Technology, Dec 2019.

[46] u blox. Zed-f9p datasheet. `https://www.u-blox.com/en/docs/UBX-17051259`, June 2020. [Online; accessed Feb 25, 2021].

[47] H. Weigel, P. Lindner, and G. Wanielik. Vehicle tracking with lane assignment by camera and lidar sensor fusion. IEEE Intelligent Vehicles Symposium, pages 513–520, June 2009.

[48] WolfWings. Barrel distortion visual example. `https://en.wikipedia.org/wiki/File:Barrel_distortion.svg`, 2008. [Online; accessed May 27, 2021].

[49] Qian Zhang and Taek Lyul Song. Improved bearings-only target tracking with iterated gaussian mixture measurements. IET Radar Sonar Navig, pages 294–303, 2017.

Appendix

A: Script for splitting dataset following the YOLO standard

```
import glob
import random
import os
import shutil
import argparse

parser = argparse.ArgumentParser(description = "Split and shuffle dataset into train/test/validation sets.")
parser.add_argument("train", type=float, help="portion of the original set to be used as a training set (as a decimal number.)")
parser.add_argument("val", type=float, help="portion of the original set to be used as a validation set (as a decimal number.)")
parser.add_argument("path", type=str, help="path to the folder containing the original dataset. This folder needs to contain both .txt and .png files correspoding to the YOLO 1.1 annotation standard.")

args = parser.parse_args()

PATH = args.path
r_train = args.train
r_val = args.val

print("Splitting dataset in " + str(PATH) + ". " + str(r_train*100) + "% Training, " + str(r_val*100) + "% Validation and " + str(100-(r_train+r_val)*100) + "% Test.")

if r_train + r_val > 1:
    raise Exception("Training and validation portion set to be greater than 1. The combined size of train and val cannot exeed 100% of the original dataset.")

train_folder = PATH + 'train/'
valid_folder = PATH + 'valid/'
test_folder = PATH + 'test/'

# Making sure there are no existing train/valid/test folders. This is to avoid duplicate files and files not listed in the train/validation/test.txt-files.
if (os.path.exists(train_folder)):
```

```python
        raise Exception("Train folder already exists. Remove all existing train/val/test folders/files before you continue.")
    if (os.path.exists(valid_folder)):
        raise Exception("Validation folder already exists. Remove all existing train/val/test folders/files before you continue.")
    if (os.path.exists(test_folder)):
        raise Exception("Test folder already exists. Remove all existing train/val/test folders/files before you continue.")

    # Get all paths to your images files and text files
    img_paths = glob.glob(PATH+'*.png')
    txt_paths = glob.glob(PATH+'*.txt')

    # Check for duplicate files (both txt and png)
    for img in img_paths:
        found = False
        for txt in txt_paths:
            if txt[:-4] == img[:-4]:
                found = True
                continue
        if not found:
            raise Exception(str(img) + " does not have a corresponding text file.")

    for txt in txt_paths:
        found = False
        for img in img_paths:
            if img[:-4] == txt[:-4]:
                found = True
                continue
        if not found:
            raise Exception(str(txt) + " does not have a corresponding image file.")

    # Calculate number of files for training, validation
    data_size = len(img_paths)

    train_size = int(data_size * r_train)
    val_size = int(data_size * r_val)
    test_size = data_size - train_size - val_size
    print("Training Images: " + str(train_size) + ", Validation Images: " + str(val_size) + ", Test Images: " + str(test_size-2))

    print("Shuffling the data (with their corresponding text files.)")

    \# Combine txt and img files
    img_list = list()
    txt_list = list()

    for img in img_paths:
        for txt in txt_paths:
            if txt[:-4] == img[:-4]:
                img_list.append(img)
                txt_list.append(txt)
                continue

    file_duo = zip(img_list, txt_list)

    \# Shuffle
    img_txt = list(file_duo)
    random.seed(43)
    random.shuffle(img_txt)
    img_paths, txt_paths = zip(*img_txt)

    \# Check that all txt and img files in the list matches
    for img, txt in zip(img_paths, txt_paths):
        if img[:-4] != txt[:-4]:
            raise Exception("Image: " + str(img) + " and text: " + str(txt)
                + " are not similar.")
        if img[-4:] != ".png":
            raise Exception(str(img) + " is not a PNG-file.")
        if txt[-4:] != ".txt":
            raise Exception(str(txt) + " is not a txt.-file.")

    \# Now split the list into training, validation and test sets
    train_img_paths = img_paths[:train_size]
    train_txt_paths = txt_paths[:train_size]

    valid_img_paths = img_paths[train_size:val_size+train_size]
    valid_txt_paths = txt_paths[train_size:val_size+train_size]

    test_img_paths = img_paths[val_size+train_size:]
    test_txt_paths = txt_paths[val_size+train_size:]

    os.mkdir(train_folder)
    os.mkdir(valid_folder)
    os.mkdir(test_folder)

    def move(paths, folder):
        for p in paths:
            shutil.move(p, folder)

    # Move images and text files to train, valid and test-folders.
    move(train_img_paths, train_folder)
    move(train_txt_paths, train_folder)
    move(valid_img_paths, valid_folder)
    move(valid_txt_paths, valid_folder)
    move(test_img_paths, test_folder)
    move(test_txt_paths, test_folder)

    print("Migration completed.")

    train_file = PATH + 'train.txt'
    valid_file = PATH + 'val.txt'
    test_file = PATH + 'test.txt'

    train_img_paths = glob.glob(train_folder + '*.png')
    test_img_paths = glob.glob(test_folder + '*.png')
    val_img_paths = glob.glob(valid_folder + '*.png')

    def writeListContents(inputlist, filepath):
        \# To avoid writing to already existing files (which might contain
           non-existing paths.)  Trust the script to do the job.  `O.o)/`
        if (os.path.exists(filepath)):
            raise Exception(str(filepath) + ' already exists. Remove all
                existing train/val/test folders/files before you continue.')
        f = open(filepath, "w")
        for i in range(len(inputlist)):
            f.write(inputlist[i] + "\n")
        f.close()

    print("Creating train.txt, valid.txt and test.txt files. Filling them
        with the corresponding image paths.")
    writeListContents(train_img_paths, train_file)
    writeListContents(val_img_paths, valid_file)
    writeListContents(test_img_paths, test_file)
```

B: Script for changing a ROS transform

```python
import rosbag
from copy import deepcopy
import tf

bagInName = 'otter27-ouster-scen1-512-cloudy-2021-03-02-12-54-01.bag'
bagIn = rosbag.Bag(bagInName)
bagOutName = 'otter27-ouster-scen1-512-cloudy-2021-03-02-12-54-01-fixed.bag'
bagOut = rosbag.Bag(bagOutName,'w')
with bagOut as outbag:
    for topic, msg, t in bagIn.read_messages():
        if topic == '/tf':
            new_msg = deepcopy(msg)
            for i,m in enumerate(msg.transforms): # go through each frame->frame tf within the msg.transforms
                if m.header.frame_id == "targa_base":
                    if m.child_frame_id == "eocam_port":
                    rotation = tf.transformations.quaternion_from_euler(0, 0, 0.50)
                    m.transform.rotation.x = rotation[0]
                    m.transform.rotation.y = rotation[1]
                    m.transform.rotation.z = rotation[2]
                    m.transform.rotation.w = rotation[3]
                    new_msg.transforms[i] = m


            outbag.write(topic, new_msg, t)
        else:
            outbag.write(topic, msg, t)

bagIn.close()
bagOut.close()
```

Multi-sensor multi-target tracking using LIDAR and camera in a harbor environment

NTNU
Norwegian University of
Science and Technology

MARITIME
ROBOTICS