Fusion of Lidar and Camera for Collision Avoidance Purposes

Vegard Kamsvåg

December 2017

TTK4551 - Engineering Cybernetics, Specialization Project
Department of Engineering Cybernetics
Norwegian University of Science and Technology

Supervisor: Edmund Førland Brekke, NTNU
Co-supervisor: Geir Hamre, DNV GL

Problem Description

The goal of the specialization project is to prepare the processing pipeline leading to camera-lidar fusion. A comparison of detections of interesting objects in the camera data with interesting objects in the lidar data will be performed, and suitable detection models for camera and lidar will be analyzed .

Hvis tid

The following subtasks are proposed for the project:

1. Installation of Velodyne lidar together with camera, and recording of time-synchronized data from both sensors.
2. Calibration of the sensors, including specification of world-frame-to-sensor-frame measurement models for both sensors.
3. Implementation of a detector based on a convolutional neural network (CNN) for the detection of boats in camera data.
4. Implementation of a detector based on intensity of reflected signal strength for the lidar.
5. Analysis of the extent to which lidar detections correspond to camera detections and vice versa.

dedication (optional)

Abstract

Write your abstract here...

Preface

This project report presents the work for the mandatory course *TTK4551*, taken during the fall of 2017. The course counts for 7.5 credit points of the two-year master's programme in cybernetics and robotics.

Table of Contents

Abstract i

Preface ii

Table of Contents iv

Abbreviations v

Notation vi

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Review . . . 1
- 1.3 Report outline . . . 3

2 Background Theory 5
- 2.1 Sensor Fusion . . . 5
- 2.2 Computer Vision . . . 7
  - 2.2.1 Spatial Image Filtering . . . 8
  - 2.2.2 Convolutional Neural Nets for Object Recognition . . . 9
- 2.3 Robot Operating System . . . 15

3 Sensor Models 17
- 3.1 Image Formation . . . 17
  - 3.1.1 The Pinhole Camera Model . . . 17
  - 3.1.2 Intrinsic and Extrinsic Parameters . . . 18
- 3.2 Lidar . . . 22
  - 3.2.1 Velodyne VLP-16 Lidar Specifications . . . 22
  - 3.2.2 Lidar Sensor Model . . . 23
  - 3.2.3 Coordinate Transformation . . . 26
  - 3.2.4 The Lidar Point Cloud . . . 26

4 Method **27**
4.1 Sensors, Hardware and Processing Pipeline 27
4.2 Visual Detection based on Faster R-CNN 29
4.2.1 Faster R-CNN 29
4.2.2 Training and validation data 30
4.2.3 Implementation Aspects 30
4.3 ROS Implementation 31
4.3.1 ROS Nodes and Data Flow 31
4.3.2 Rosbag 32
4.3.3 MATLAB implementation 33
4.4 Camera Calibration 33
4.4.1 Time Synchronization 35

5 Experiments **37**
5.1 Performed Maneuvers 39

6 Results **43**
6.1 Post-processing 43
6.2 Metrics 46
6.3 Experimental Results 47
6.3.1 Experiment 1 47
6.3.2 Experiment 2 49
6.3.3 Experiment 3 50
6.3.4 Experiment 4 52
6.3.5 Experiment 5 53
6.3.6 Experiment 6 55
6.3.7 Experiment 7 56
6.3.8 Faster R-CNN Performance 58

7 Discussion **61**
7.0.1 Detection of Boats in Images 61

8 Conclusion **65**

A Appendix **67**
A.1 Geometric Camera Calibration 67
A.1.1 Perspective Projection between the Model Plane and Its Image . . 67
A.1.2 Solving the Calibration Problem 68
A.1.3 Radial Lens Distortion 70
A.1.4 Calibration Procedure 71

Abbreviations

ADC	=	Analog-to-Digital Converter
AIS	=	Automatic Identification System
ASV	=	Autonomous Surface Vehicle
CNN	=	Convolutional Neural Net
CCD	=	Charge Coupled Device
INS	=	Inertial Navigation System
GNSS	=	Global Navigation Satellite System
FOV	=	Field Of View
RPS	=	Rotations Per Second
ROS	=	Robot Operating System
HDMI	=	High Definition Multimedia Interface
OS	=	Operating System
USB	=	Universal Serial Bus
IP	=	Internet Protocol
TCP	=	Transmission Control Protocol
NMEA	=	National Marine Electronics Association
CPU	=	Central Processing Unit
GB	=	Gigabyte
RAM	=	Random Access Memory
HDD	=	Hard Disk Drive
FPGA	=	Field Programmable Gate Array

Notation

$\mathcal{R}_a^b$	=	Rotation matrix from coordinate systems a to b
$\overrightarrow{OP}$	=	Coordinate-free vector from point O to point P
$\mathbf{M}$	=	A matrix
$\mathbf{p}^a$	=	A coordinate vector, $\mathbf{p}$ decomposed in the a coordinate frame
t_{ab}^a	=	A translation vector from frame b to frame a, decomposed in frame a

Chapter 1

Introduction

1.1 Motivation

Autonomous surface vehicles (ASVs) operating in urban environments, e.g. ferries, will need slightly different exteroceptive sensors than ASVs operating in the open sea. A lidar with a range of 100 meters may be more appropriate than a maritime radar with range of several kilometers. Furthermore, the complexity of the environment means that the rich information from optical cameras will be more important. In order to build a coherent world image, which, e.g., collision avoidance decisions can be based on, the data from these sensors must be fused, together with data from interoceptive sensor systems such as an inertial navigation system (INS). Sensor fusion is the process of combining observations from different sensor to provide a robust and complete description of an environment or process of interest [1].

1.2 Review

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

- Stiller et al. [2] in 2000 proposed a multisensor concept with a variety of different sensor technologies with widely overlapping fields of view for an autonomous, unsupervised vehicle. They used stereo vision, laser scanners, radar, and short range radar, combined by sensor fusion into a joint obstacle map. The research was conducted as a part of the German project *Autonomes Fahren*.

- Mälich et al. [3] used low-level fusion of multibeam lidar and vision sensor into a detection and tracking framework. They used a cascaded AdaBoost (adaptive boosting [4]) detector based on haar-wavelet like features [5]. Lidar returns were used to generate regions of interest in the images.

- Aufrère et al. [6] at the NavLab group at Carnegie Mellon University proposed a high level fusion approach for object tracking using cameras and lidars for autonomous vehicles in cluttered urban environments. Their approach used a map-based fusion system, with a probability-based predictive model.

- Cho et al. [7] developed a multi-sensor system for moving object detection and tracking, building on the work of Aufrère et al. for the feature extraction for the lidar. Their approach fuses vision, lidar and radar. Detection in the vision module is represented as bounding boxes, and the data from the sensors were fused in an extended Kalman filter. The system detects and tracks pedestrians, bicyclists, and vehicles.

- Premebida et al. [8] demonstrated a perception system for pedestrian detection in urban scenarios using information from lidar and a single camera. Two sensor fusion architectures are described in their paper. A centralized architecture, where the fusion is done at the feature level, i.e. features from lidar and vision space combined in a single vector for posterior classification using a single classifier. The decentralized architecture employs two classifiers, one per sensor feature-space, fused by a trainable fusion method applied over the likelihoods provided by the component classifiers. They showed that the trainable fusion method lead to enhanced detection performance, and maintenance of false-alarms under tolerable values in comparison with single-based classifiers.

- Weigel et al. [9] demonstrated a vehicle tracking and lane detection multi-sensor system, using a lidar and a monocular camera. Detected vehicles are tracked and managed by a multi-object extended Kalman filter using the data from the lidar and the camera. The lidar was used to create appropriate regions of interest in the image plane, and subsequently the measurements in the image plane was incorporated into the Kalman filter.

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

- Wolf et al. [10] describe the perception and planning systems of an autonomous surface vehicle with the goal to detect and track other vessels at medium to long ranges. They employ a NASA JPL developed tightly integrated system termed CARACaS (Control Architecture for Robotic Agent Command and Sensing) that blends the sensing, planning and behaviour autonomy necessary for such missions. In their paper, they presents an autonomy system that detects and tracks vessels of a defined class while patrolling near fixed assets. The sensor suite includes a wide-baseline stereo vision system for close-up perception and navigation, and a 360 degree camera head for longer range detection, identification, and tracking. The perception system termed SAVAnT (Surface Autonomous Visual Analysis and Tracking) receives sensory input from 6 cameras, stabilized by INS pose, detects objects of interest and calculates absolute bearings for each contact. The applications discussed in the paper is of a military nature.

- Elkins et al. [11] published a paper on the Autonomous Maritime Navigation project, with the stated goal of creating a set of sensors, hardware, and software that enables autonomy on unmanned surface vehicle (USV) platforms. The sensor suite includes cameras, radar, lidar, compass and GPS, integrated into a sensor fusion engine. Fusion algorithms are used to compile and correlate these data into a common tactical picture for the USV. In their paper, they showed the benefits of using a lidar for close-range detections over more long-range sensors such as radars. They also showed that the wavelength of the lasers in the lidar is such that the radiated energy does not reflect well of the water surface, i.e. most points returned are not water, making it well suited in the detection of obstacles.

1.3 Report outline

This report is divided into seven chapters. Following the introduction, background theory is presented in chapter 2. The geometric sensor models for the camera and the lidar, which is a prerequisite for sensor fusion, is presented in chapter 3. Chapter 4 presents some of the implementation details, the software and hardware used in the data collection and analysis, and the calibration procedure used for the camera. In chapter 5, the experiments performed in this project are presented, as well as the location used. In chapter 6, the results are presented and discussed. Chapter 7 presents the conclusion of the project, and suggests further work.

Chapter 2

Background Theory

In this chapter

introduser kapittel

2.1 Sensor Fusion

Robots and autonomous vehicles, be it a car or a ship, operate in unstructured environments, environments that are unpredictable. While robots working on a assembly line work in very structured, predictable environments, a vessel attempting to atonomously navigate a busy shipping lane has to rely on sensor inputs in order to make sense of its environment, and navigate safely. In sensing its environment, a vessel might use various sensors, such as radar(s), cameras, sonars or laser range finders. The different sensors typically give a incomplete and imperfect view of the surrounding world. Sensor data fusion is the process of combining such mutually complementary sensor information in such a way that a better understanding of the surroundings can be achieved [12]. The U.S. Joint Directors of Laboratories (JDL) Data Fusion Working Group developed a process model in 1985 for characterizing hierarchical levels of fusion processing, categorized fusion functions, and candidate algorithm approaches, and is the most widely used system for categorizing data fusion-related functions [13]. The model is illustrated in figure 2.1.

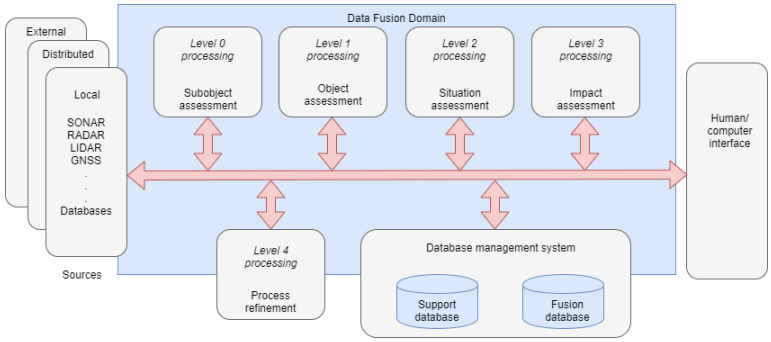


Figure 2.1: The Joint Directors of Laboratories data fusion model.

The JDL model defines the fusion levels as follows:

- Level 0: *Subobject assessment*. Preconditioning data to correct biases, perform spatial and temporal alignment, and standardize inputs.
- Level 1: *Object assessment*. Association of data (including products of prior fusion) to estimate an object or entitys position, kinematics, or attributes (including identity).
- Level 2: *Situation assessment*. Aggregation of objects/events to perform relational analysis and estimation of their relationships in the context of the operational environment.
- Level 3: *Impact assessment*. Projection of the current situation to perform event prediction, threat intent estimation, own force vulnerability, and consequence analysis.
- Level 4: *Process refinement*. Evaluation of the ongoing fusion process to provide user advisories and adaptive fusion control or to request additional sensor/source data.

The models different levels do not represent a process flow, but rather provides a structured and integrated view on the complete functional chain from distributed sensors, data bases, and human reports to the users and their option to act including various feedback loops at different levels. Although the model was developed with military applications in mind, it remains valid in civilian applications. Koch [12] identifies two characteristic features of sensor data fusion:

1. The available sensor data and context knowledge to be fused typically provide incomplete and imperfect pieces of information. The reasons for this are manifold, and are unavoidable in real-world applications. For dealing with these deficiencies, sophisticated mathematical methodologies and reasoning formalisms are applied.
2. Sensor data fusion is closely related to the practical design of surveillance and reconnaissance components for information systems. In implementing fundamental theoretical concepts, a systematic way of finding reasonable compromises between

> mathematical exactness and pragmatic realization issues, as well as suitable approximation methodologies are therefore inevitable. System aspects such as robustness and reliability even in case of unforeseeable nuisance phenomena, priority management, and graceful degradation are of particular importance in view of practicability.

This project aims to prepare the processing pipeline leading to fusion of measurements from a lidar with measurements from a camera. A lidar provides excellent range information, but has limits with regard to object recognition. A camera, on the other hand, is good for object recognition but has limits to the high resolution range information. The two sensors exhibit complementary properties, and combining (fusing) the measurements from both sensors can achieve more specific inferences about the surroundings than using a single, independent sensor. In order to associate measurements from the lidar and the camera to the external world, fusion with an INS is necessary, to be able to associate the (moving) coordinate frames of the sensors with a common world frame. Moreover, to be able to associate measurements from one sensor with the other, they need to be synchronized in time.

2.2 Computer Vision

Computer vision is a field of engineering and science concerned with extracting useful information from images. This has proved to be a challenging task, and it is still today an active field of research. Visual data is very complex, and the same object represented by two different images could be perceived very differently by a computer based on variations such as changes in illumination, partial occlusion of objects, changes in orientation, deformation and so on. Such variations are illustrated in figure 2.2. Despite these challenges, the recent advancements made within the field of deep learning, and deep convolutional neural nets in particular, has significantly improved the ability of computers to recognize objects in images. The ImageNet Large Scale Visual Recognition Challenge has been run annually since 2010, and is a benchmark in object category classification and detection on hundreds of object categories and millions of images [14]. The results of the top 5 classification errors from the challenges up until 2016 is shown in figure 2.3.

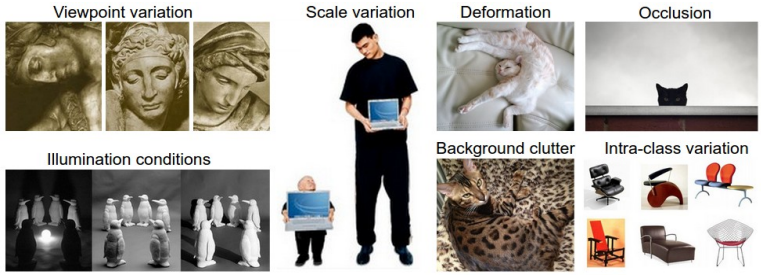


Figure 2.2: Challenges related to object classification in images.[1]

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

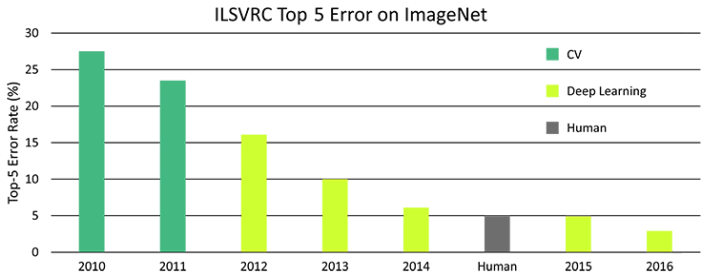


Figure 2.3: ImageNet top 5 errors over time.[2]

As seen in figure 2.3, deep learning algorithms dominate the field of object category classification and detection, even outperforming humans with state-of-the-art deep convolutional neural nets. This section will introduce some necessary basic image processing operations, and give a brief introduction to convolutional neural nets used for image object classification and detection.

2.2.1 Spatial Image Filtering

The term spatial in spatial image filtering refer to the image plane itself, and spatial image processing involves direct manipulation of the pixels themselves in the image, in contrast to other methods operating in a transform domain such as the frequency plane. A spatial filter consists of a *kernel* (also called a mask, neighborhood, template or window) (typically rectangular), and a *predefined operation* that is performed on the pixels in the image encompassed by the neighborhood [15]. Filtering the image creates a new pixel with coordinates equal to the center of the neighborhood, with value given by the predefined operation performed on the pixels in the neighborhood. Spatial filtering of an image of size $M \times N$ with a filter of size $m \times n$ is given by

$$g(x,y) = w(x,y) * f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x-s, y-t) \tag{2.1}$$

where $a = \frac{m-1}{2}$ and $b = \frac{n-1}{2}$, $w(s,t)$ is the filter kernel, with its center at $w(0,0)$ aligned with the pixel at location (x,y). $g(x,y)$ is the response of the filter at location (x,y). $*$ denotes the convolution operator. x and y are varied such that each pixel in the image is visited. In other words, spatial filtering of an image with a kernel of size $m \times n$ corresponds to convolving the image with the kernel. The filter and the corresponding image pixels for a given (x,y) are illustrated in figure 2.4. The filter kernel used in this illustration is 3×3 pixels.

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

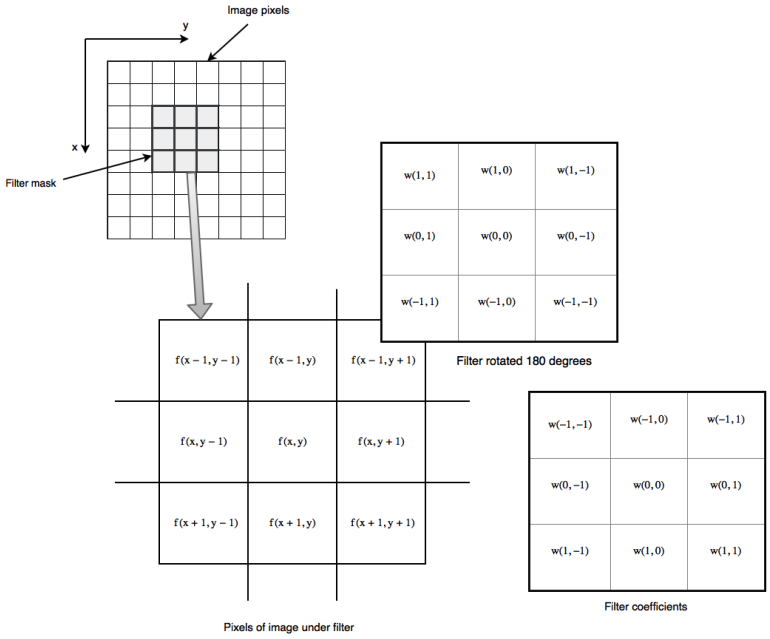


Figure 2.4

Convolution between the image and the filter kernel corresponds to correlating the image and the filter kernel rotated 180 degrees.

2.2.2 Convolutional Neural Nets for Object Recognition

A convolutional neural net, abbreviated CNN, is a neural network designed for object recognition in images. Since their introduction by LeCun *et al.* [16] in the late 1980's, convolutional neural nets have shown excellent performance at tasks such as hand-written digit classification and face detection. In 2012, Krizhevsky *et al.* [17] won the ImageNet 2012 classification benchmark with their deep convolutional neural network, dubbed AlexNet, achieving an top-5 test error rate of 15.3%, compared with the 2nd place result of 26.2%. Since then, the ImageNet classification benchmark has been dominated by deep convolutional nets. The dramatic improvement in performance can be attributed to several factors [18]:

1. The availability of very large datasets, with millions of labeled examples.

2. The adaptation of powerful GPU processing implementations, making training of very large models tractable.

3. Better model regularization strategies, such as dropout, to prevent overfitting [19].

In a classifier using classical computer vision techniques, features are extracted from an image, for example using a histogram of oriented gradients, and the extracted features are

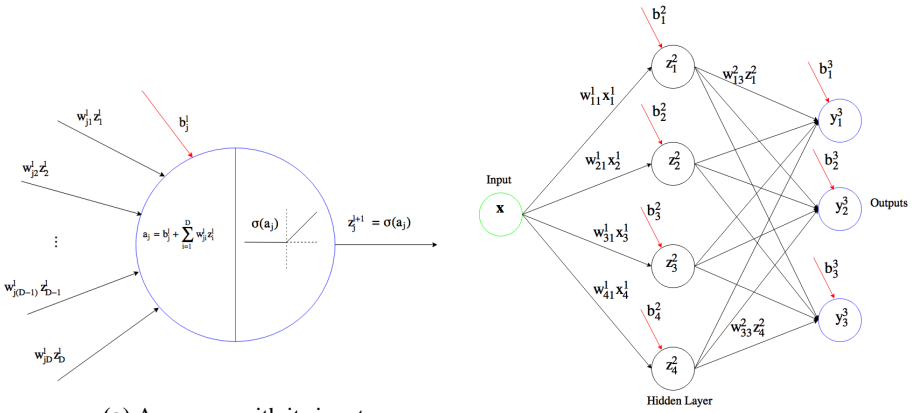
subsequently used to train a classifier such as a support-vector machine. In a convolutional neural network, on the other hand, the features are hidden, and feature extraction and classification is performed in a single pipeline. Features in a CNN are generated via one or several layers of filter convolutions, which generates a set of abstract sub-images from the input image. For the user, a CNN classifier appears as a ”black box” where the internal workings of the feature extraction and classification are largely hidden. In order to understand how a convolutional neural net works, some basic concepts need to be introduced. The following sections first introduce the neuron and the concept of a neural network followed by a short discussion on how they are trained, then a brief introduction to convolutional neural nets will be given. The derivations and notation is largely based on that given in [4] for the conventional neural network, while the convolutional neural network part is based on [20].

Neural Networks

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

$$y(\mathbf{x}, \mathbf{w}) = w_0 + \sum_{j=1}^{n-1} w_j \phi_j(\mathbf{x}) \tag{2.2}$$

where $\phi_j(\mathbf{x})$ are known as the nonlinear *basis functions*. A neural network utilized for regression and classification, on the other hand, can be thought of as a combination of basis functions in parametric form, where the parameters of the basis functions are adapted during training. The term *neural network* has its origins in attempts to find mathematical representations of information processing in biological systems. The basic building block of a neural network is the *neuron*, or *perceptron*, illustrated in figure 2.5a.



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

(b) A simple neural network.

Figure 2.5: A neuron, alongside a simple neural network with a single hidden layer.

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

$$a_j = b_j^1 + \sum_{i=1}^{D} w_{ji}^1 x_i \tag{2.3}$$

where $j = 1, \ldots, M$, b_j^1 is a bias, the parameters w_{ji}^1 are the weights, and the superscript 1 indicates that the corresponding parameters are the parameters of the first layer in the network. The result of this weighted sum of the inputs, a_j is known as the activation. Each a_j is then transformed via a nonlinear, differentiable activation function, $z_j\sigma(a_j)$, which is then input to the next layer in the network. Figure 2.5a illustrates this process for a single neuron. The quantities z_j are referred to as *hidden units*, and the corresponding layers are referred to as *hidden layers*. The nonlinear functions $\sigma(\cdot)$ are generally chosen to be sigmoidal functions such as the logistic sigmoid function

$$\sigma(a) = \frac{1}{1 + \exp(-a)} \tag{2.4}$$

or the hyperbolic tangent function ($\tanh(a)$). The outputs z_j are again linearly combined to give output unit activations as

$$a_k = b_k^2 + \sum_{j=1}^{M} w_{kj}^2 z_j \tag{2.5}$$

where $k = 1, \ldots, M$ and K is the total number of outputs. This corresponds to the second layer of the network. Lastly, the output unit activations are transformed using an appropriate activation function to give a set of network outputs y_k. A simple network with a single hidden layer is illustrated in figure 2.5b. The choice for the output activation function is largely determined by the nature of the data and the assumed distribution of the target variables. For binary classification problems, the logistic sigmoid function is commonly used [4].

Convolutional Neural Nets

A convolutional neural net or ConvNet, abbreviated CNN, is similar to ordinary neural networks in that they are made up of neurons that have learnable biases and weights. What distincts it from ordinary neural nets is that it assumes that the input data has a grid-like topology, such as an image being a 3D grid of pixels, where the depth is represented by the color channels in the image. As the name implies, the CNN employs convolution in one or more of the layers in the network, and the layers have neurons arranged in 3 dimensions. A traditional neural network use matrix multiplication by a matrix of parameters, with a separate parameter describing the interactions between each input unit and each output unit, meaning that every input unit interacts with every output unit. CNNs however, have *sparse* interactions, accomplished by using kernels which are smaller than the input for holding the weights, which are convolved with the input. This drastically reduces the parameters in the network compared to the traditional network using matrix multiplication.

If there are m inputs and n outputs, the matrix multiplication approach requires $m \times n$ parameters. For an input such as an image, which can have thousands or millions of pixels, the number of parameters needed become very large. In a CNN, the number of connections for each output is limited by the kernel size and number of kernels [20].

CNNs also employ parameter sharing, where the same parameter is used for more than one output in the model. This is accomplised by the convolution operation, where each member of the kernel is used at every position of the input. This means that rather than learning a separate set of parameters for every location, only one set is learned, further reducing the number of parameters needed in the model. A simple representation of a layer in a convolutional neural net is shown in figure 2.6. The neurons (indicated by white circles) arranged in depth in the output activation layer are connected to the same inputs through different kernels, while all neurons at the same depth share kernel parameters. As indicated in the figure, a single layer may consist of several filter kernels.

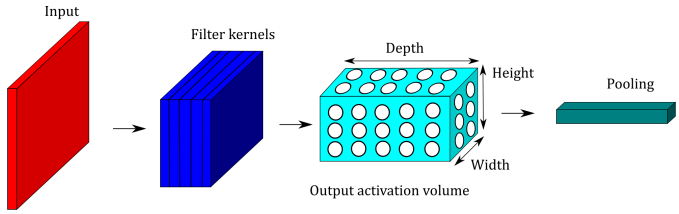


Figure 2.6: A CNN layer. The 3D input volume is transformed into a 3D volume of neuron activations.

A typical layer of a CNN consists of three stages; convolution, detection and pooling, illustrated in figure 2.7 [20].

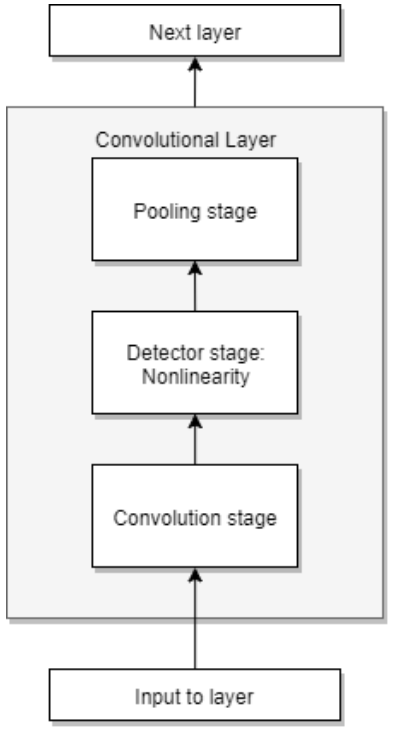


Figure 2.7: The three stages of a convolution layer.

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

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

1. The number of kernels, K.
2. The spatial extent of the filter kernel, F.
3. The stride S, the number of pixel displacements for each calculation.
4. The amount of zero padding to the brim of the image, P.

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

$$\begin{aligned} W_2 &= 1 + (W_1 - F + 2P)/S \\ H_2 &= 1 + (H_1 - F + 2P)/S \end{aligned} \tag{2.6}$$

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

Training a Neural Net

For a neural network used in classification of a input, the training of the network consists of minimizing the *error function*, which is a function of the weights of the network, $\mathbf{w}$. For K separate binary classifications, a network with K outputs can be used, where each output has a *normalized softmax* activation function [4], given as

$$\sigma(z_k) = \frac{e^{z_k}}{\sum_{i=1}^{K} e^{z_i}} \tag{2.7}$$

Associated with each output is a binary class label, $t_k \in \{0, 1\}, \quad k = 1, \ldots, K$. Assuming the class labels are independent, the conditional distribution of the targets is [4]

$$p(\mathbf{t}|\mathbf{x}, \mathbf{w}) = \prod_{k=1}^{K} y_k(\mathbf{x}, \mathbf{w})^{t_k}[1 - y_k(\mathbf{x}, \mathbf{w})^{1-t_k}] \tag{2.8}$$

Taking the negative logarithm of the corresponding likelihood function gives the error function

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\} \tag{2.9}$$

where y_{nk} denotes $y_k(\mathbf{x}_n, \mathbf{w})$. Training the network consists of finding a weight vector $\mathbf{w}$ that minimizes equation 2.9. This is done by taking small steps in weight space from $\mathbf{w}$ to $\mathbf{w} + \delta\mathbf{w}$ such that the gradient of the error function vanishes, minimizing the error function:

$$\nabla E(\mathbf{w}) = 0 \tag{2.10}$$

The error function has a nonlinear dependence on the weights and biases, and so there may be many local minima where the gradient of the error function is zero. An analytical solution of equation 2.10 is typically unobtainable, so the typical approach is to use iterative numerical procedures [4]. One approach is to use a combination of error backpropagation and stochastic gradient descent, where the error gradient calculated at the output is propagated back through the network, giving the partial derivative of the error for each hidden unit. In stochastic gradient descent, the weights are updated based on one data point at a time, according to

$$w_{ji}^{l} \to w_{ji}^{l} - \eta \frac{\partial E_n(\mathbf{w})}{\partial w_{ji}^{l}} \tag{2.11}$$

where the parameter $\eta > 0$ is known as the *learning rate*, which is a parameter provided by the user, and $E_n(\mathbf{w})$ is the error function for a single data point. For a more detailed explanation of neural network training the reader is referred to [4], in particular chapters 5.2 and 5.3.

2.3 Robot Operating System

A brief introduction to ROS will be given in this section, based on the introduction given in *Programming Robots with ROS* by Quigley et al. [21]. ROS, which is an abbreviation for Robot Operating System, is not, as the name might imply, a operating system, but a framework running on top of a traditional OS. The framework of ROS is based on several philosophical aspects:

- *Peer to peer*: A ROS system consists of several small programs (called nodes) connected to each other, continuously exchanging messages. Messages travel directly from one node to another.

- *Tools-based*: Complex ROS systems can be created from many small, generic programs. ROS does not have a integrated development and runtime environment, and tasks such as (but not limited to) navigating the source code tree, visualizing the system interconnections, generating documentation, and logging data are performed by separate programs.

- *Multilingual*: ROS software modules can be written in any language for which a *client library* has been written. Client libraries exist for C++, Python, LISP, Java, JavaScript, MATLAB, Ruby, Haskell, R, and others. This provides flexibility for the programmer, in that he or she can choose the language they are most familiar with, or is best suited for the task, when creating new functionality. ROS achieves this multilinguality by enforcing a convention for serializing messages being passed between nodes.

- *Thin*: ROS conventions encourage developers to create standalone libraries and then wrap those libraries so that they can send and receive messages to and from other ROS modules. This allows reuse of software outside ROS, and simplifies automated testing using continuous integration tools.

- *Free and open source*: The core of ROS is released under the BSD license [22], which allows commercial and noncommercial use.

In ROS, `roscore` is a service that provides connection information to nodes so that they can transmit messages to and from one another. Each node connects to `roscore` on startup to register details of the messages it publishes, and the details of the messages to which it wishes to subscribe. `roscore` is not a server in the traditional client/server sense, all messages are sent peer-to-peer between nodes. The `roscore` service is only used by nodes to know where to find their peers. Due to this fact, every ROS system needs a running `roscore`. Upon startup, every ROS node expects its process to have an environment variable `ROS_MASTER_URI`, containing a string in the format `http://hostname:portnumber`, to tell it where the `roscore` service is running, and which port it is accessible on. The connection between nodes, and between individual nodes and `roscore` is illustrated in figure 2.8.

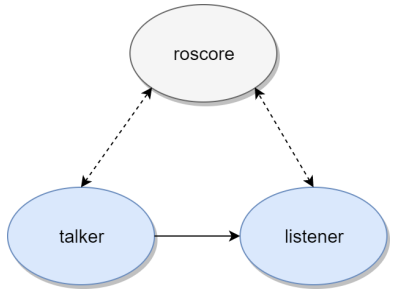


Figure 2.8: Roscore and its connection to other nodes in the system.

The ephemeral connection between nodes and `roscore` is illustrated by dashed lines, indicating periodic calls to `roscore` to find peers, while the peer-to-peer message passing is illustrated by the solid line between the nodes.

Chapter 3

Sensor Models

In this chapter, mathematical models relating measurements from the two exteroceptive sensors used in the project to world coordinates are presented. In order to associate measurements from one sensor to the other, as well as measurements from the sensors with a common world coordinate frame, a mathematical model describing this relationship is needed.

3.1 Image Formation

Camera systems have found a wide array of applications since its invention, and are today widely used in industry. Application areas range from surveillance (CCTV), to image acquisition in scientific exploration of the earth, oceans and space. In recent years, there has been a drive towards utilizing cameras for automatic object detection and recognition, made possible through advances in computing power and algorithms . There is a drive towards more and more autonomous operation of vehicles, and camera systems play an integral part in the sensing of the environment necessary for safe autonomous operation . This section presents the pinhole camera model, and how to relate image coordinates to world coordinates through transformations. The theory presented in this section is largely based on chapter 1 in [23].

kilde

kilde

3.1.1 The Pinhole Camera Model

The pinhole camera model was first proposed by Brunelleschi in the early fifteenth century [23]. Consider a rectangular box with a translucent plate at one end. At the opposite end there is a small pinhole, small enough so that exactly one ray of light would pass through the translucent plate, the pinhole, and some scene point. The distance between the pinhole and the translucent plate is called the *focal length*, denoted f. This is of course impossible in reality, as the pinhole will have a finite, though small size. As such, each point in the image plane will collect light from a cone of rays. A real camera will also be equipped with lenses, which complicates the simple model of rays passing through a

hole. Still, the pinhole camera model is mathematically convenient, and often provides an acceptable approximation of the imaging process. The pinhole perspective projection model is illustrated in figure 3.1.

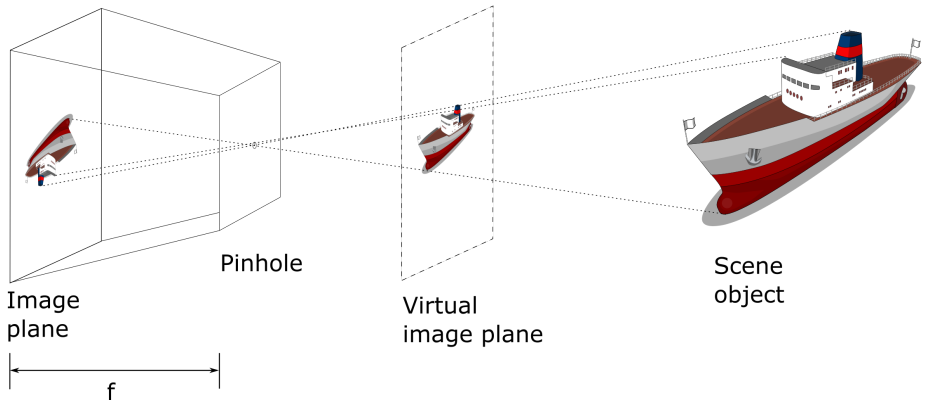


Figure 3.1: The pinhole camera model

The pinhole perspective creates inverted images, so it is sometimes convenient to consider a virtual image plane lying in front of the pinhole, at the same distance from the pinhole as the actual image plane. The virtual image is not inverted, but otherwise identical to the actual image.

3.1.2 Intrinsic and Extrinsic Parameters

The world and camera coordinate systems are related by a set of physical parameters. These parameters can be separated into *intrinsic* parameters, which relates the image coordinate system to the normalized coordinate system presented in figure 3.2, and *extrinsic* parameters, which relate the cameras coordinate system to a fixed world coordinate system and specify its position and orientation in space.

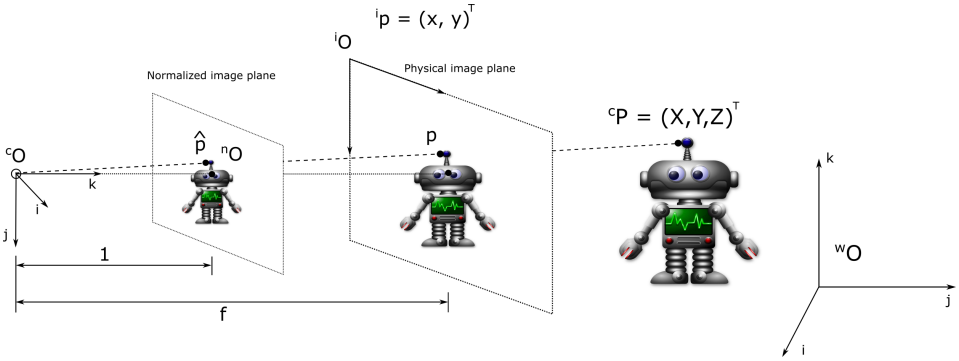


Figure 3.2: Coordinate systems

In figure 3.2, ${}^{n}O$ denotes the origin of the normalized image plane, ${}^{i}O$ is the origin of the physical image plane, ${}^{w}O$ is the origin of the world frame, and ${}^{c}O$ is the origin of the camera 3D frame. Since ${}^{c}O$, ${}^{c}\hat{p}$ and ${}^{c}P$ in figure 3.2 are collinear, we have that $\overrightarrow{{}^{c}O\hat{p}} = \lambda \overrightarrow{{}^{c}OP}$ for some constant λ such that

$$\begin{cases} \hat{x} = \lambda X \\ \hat{y} = \lambda Y \iff \lambda = \frac{\hat{x}}{X} = \frac{\hat{y}}{Y} = \frac{1}{Z} \\ 1 = \lambda Z \end{cases} \tag{3.1}$$

which gives the following relation between the normalized plane coordinates ${}^{n}\hat{p}$ and the coordinates of the point ${}^{c}P$

$$\begin{cases} \hat{x} = \frac{X}{Z} \\ \hat{y} = \frac{Y}{Z} \end{cases} \tag{3.2}$$

Homogeneous Coordinates and Rigid Transformations

Homogeneous coordinates are convenient for representing geometric transformations by a matrix product. Consider a point P in some right-handed coordinate frame with origin O and the x, y and z axis given by the unit vectors $\mathbf{i}$, $\mathbf{j}$ and $\mathbf{k}$ respectively. The point can be specified by

$$\overrightarrow{OP} = X\mathbf{i} + Y\mathbf{j} + Z\mathbf{k} \tag{3.3}$$

The nonhomogeneous coordinate vector P is the vector $(X, Y, Z)^T$ in $\mathbb{R}^3$, while its homogeneous counterpart is the vector $(X, Y, Z, 1)^T$ in $\mathbb{R}^4$. The change of coordinates between two (arbitrary) Euclidean coordinate systems a and b can be represented by a rotation matrix $\mathcal{R}_b^a$ and a translation vector $\mathbf{t}$ in $\mathbb{R}^3$ as

$$\mathbf{P}^a = \mathcal{R}_b^a \mathbf{P}^b + \mathbf{t} \tag{3.4}$$

Using homogeneous coordinates, the same transformation can be written as

$$\mathbf{P}^a = \mathcal{T}\mathbf{P}^b, \quad \text{where} \quad \mathcal{T} = \begin{bmatrix} \mathcal{R}_b^a & \mathbf{t} \\ \mathbf{0}^T & 1 \end{bmatrix} \tag{3.5}$$

where $\mathbf{P}^a$ and $\mathbf{P}^b$ are now vectors in $\mathbb{R}^4$. There are several ways to parameterize the rotation matrix $\mathcal{R}$, such as by Euler angles or quaternions. For details on parameterizations of rotation matrices, the reader is referred to [24], [25] and [26].

Intrinsic Parameters

Using homogeneous coordinates, equation 3.2 can be rewritten as

$$\hat{\mathbf{p}}^n = \frac{1}{Z}\begin{bmatrix}\mathbf{I}^{3\times 3} & \mathbf{0}^{3\times 1}\end{bmatrix}^c \mathbf{P} \tag{3.6}$$

where $\mathbf{I}^{3\times 3}$ is the 3-by-3 identity matrix, and $\hat{\mathbf{p}}^n \triangleq \begin{bmatrix}\hat{x} & \hat{y} & 1\end{bmatrix}^T$ is the homogeneous coordinate vector of the projection $\hat{p}$ of the point P into the normalized image plane, and $\mathbf{P}^c$ is the homogeneous coordinate vector of the point P in the camera coordinate frame. The following section relates the homogeneous coordinates of the normalized image plane to those of the physical image plane. The physical retina of a camera is located at a distance $f \neq 1$ from the pinhole (this assumes that the camera is focused at infinity, so that the physical distance from the pinhole to the image plane equals the focal length), and the physical image point p is usually expressed in pixel units. In addition to this, pixels may be rectangular instead of square, introducing two additional scale parameters k and l, such that

$$\begin{cases} x = kf\frac{X}{Z} = kf\hat{x} \\ y = lf\frac{Y}{Z} = kf\hat{y} \end{cases} \tag{3.7}$$

We have that f represents a distance, in meters say, and a pixel has dimensions $\frac{1}{k} \times \frac{1}{l}$, where k and l are expressed in $\frac{\text{pixel}}{m}$. The parameters can be replaced by a magnification $\alpha = kf$ and $\beta = lf$ expressed in pixels. In general the origin of the physical image plane is in a corner c of the physical image plane (often the upper left corner, but this may vary) and not at it's center, and the center of the CCD matrix typically does not coincide with the image center c_0. This adds two more parameters x_0 and y_0 that define the position of c_0 in pixel units in the physical image coordinate system. Equation 3.7 is then replaced by

$$\begin{cases} x = \alpha\hat{x} + x_0 \\ y = \beta\hat{y} + y_0 \end{cases} \tag{3.8}$$

This assumes that the angle between the image axes equals exactly 90 degrees. In reality, however, the image plane axes may be skewed, such that the angle between the image plane axes is $\theta \neq 90°$. In this case, equation 3.8 is replaced by

$$\begin{cases} x = \alpha\hat{x} - \alpha\cot\theta\hat{y} + x_0 \\ y = \frac{\beta}{\sin\theta}\hat{y} + y_0 \end{cases} \tag{3.9}$$

In matrix form, equation 3.9 can be written as

$$\mathbf{p} = \mathcal{K}\hat{\mathbf{p}}, \quad \mathbf{p} = \begin{bmatrix}x\\y\\1\end{bmatrix}, \quad \mathcal{K} \overset{\text{def}}{=} \begin{bmatrix}\alpha & -\alpha\cot\theta & x_0 \\ 0 & \frac{\beta}{\sin\theta} & y_0 \\ 0 & 0 & 1\end{bmatrix} \tag{3.10}$$

The matrix $\mathcal{K}$ is called the *calibration matrix* of the camera. Putting the equations 3.6 and 3.10 together, we obtain

$$\mathbf{p}^i = \frac{1}{Z}\mathcal{K}\begin{bmatrix}\mathbf{I}^{3\times3} & \mathbf{0}^{3\times1}\end{bmatrix}\mathbf{P}^c = \frac{1}{Z}\mathcal{M}\mathbf{P}^c, \quad \text{where} \quad \mathcal{M} \stackrel{\text{def}}{=} \begin{bmatrix}\mathcal{K} & \mathbf{0}^{3\times1}\end{bmatrix} \tag{3.11}$$

The parameters α, β, θ, x_0 and y_0 are called the *intrinsic parameters* of the camera, typically obtained through a calibration procedure like the one described in section A.1.

Extrinsic Parameters

Equation 3.11 is written in a coordinate frame rigidly attached to the camera. However, for applications like autonomous vehicles, where the vehicle itself moves in some inertial coordinate system, we can express equation 3.11 in some world coordinate system using a rigid homogeneous transformation. The change of coordinates between the camera frame and the world frame can be expressed as presented in equation 3.5:

$$\mathbf{P}^c = \begin{bmatrix}\mathcal{R} & \mathbf{t} \\ \mathbf{0}^T & 1\end{bmatrix}\mathbf{P}^w \tag{3.12}$$

Here, $\mathbf{P}^w$ represents the point P in some world coordinate frame w. Substituting this into equation 3.11 yields

$$\mathbf{p}^i = \frac{1}{Z}\mathcal{M}^w\mathbf{P}, \quad \text{where} \quad \mathcal{M} = \mathcal{K}\begin{bmatrix}\mathcal{R} & \mathbf{t}\end{bmatrix} \tag{3.13}$$

Knowing $\mathcal{M}$ determines the position of the camera's optical center in the world coordinate frame w. A rotation matrix $\mathcal{R}$ is defined by three independent parameters (such as Euler angles), and together with the translation vector $\mathbf{t}$ defines six *extrinsic parameters* that define the position and orientation of the camera relative to the world coordinate frame. The depth Z in equation 3.13 is dependent on $\mathcal{M}$ and $\mathbf{P}^w$, which can be seen directly from the equation. If we denote the three rows of $\mathcal{M}$ as $\mathbf{m}_1^T$, $\mathbf{m}_2^t$ and $\mathbf{m}_3^T$, it follows that $Z = \mathbf{m}_1^T \cdot {}^w\mathbf{P}$. It's important to note that the matrix $\mathcal{M}$ is only defined up to scale, and multiplying $\mathcal{M}$ by a arbitrary constant $\lambda \neq 0$ does not change the resulting image coordinates since

$$\begin{cases} {}^ix = \frac{\mathbf{m}_1^T \cdot {}^w\mathbf{P}}{\mathbf{m}_3^T \cdot {}^w\mathbf{P}} \\ {}^iy = \frac{\mathbf{m}_2^T \cdot {}^w\mathbf{P}}{\mathbf{m}_3^T \cdot {}^w\mathbf{P}} \end{cases} \tag{3.14}$$

and the constant λ cancels out. The perspective projection matrix $\mathcal{M}$ can be written explicitly as a function of its five intrinsic parameters, the rows $\mathbf{r}_1^T$, $\mathbf{r}_2^T$ and $\mathbf{r}_3^T$ of the rotation matrix $\mathcal{R}$ and the coordinates of of the vector $\mathbf{t} = \begin{bmatrix}t_1 & t_2 & t_3\end{bmatrix}^T$ as

$$\mathcal{M} = \begin{bmatrix} \alpha\mathbf{r_1} - \alpha\cot\theta\mathbf{r}_2^T + x_0\mathbf{r}_3^T & \alpha t_1 - \alpha\cot\theta t_2 + x_0 t_3 \\ \frac{\beta}{\sin\theta}\mathbf{r}_2^T + y_0\mathbf{r}_3^T & \frac{\beta}{\sin\theta}t_2 + y_0 t_3 \\ \mathbf{r}_3^T & t_3 \end{bmatrix} \tag{3.15}$$

The intrinsic parameters are typically found through a calibration routine, while the extrinsic parameters depend on the location of the camera frame relative to the world frame the user chooses. A detailed explanation of camera calibration can be found in appendix A.1.

3.2 Lidar

A lidar is an *active* electro-optical sensor that sends out a laser pulse, and subsequently measures the parameters of the return signal bounced off some object. The lidar is known under several names, some examples being ladar, lidar, LIDAR, LADAR or laser radar. The term lidar is the most common, and will be used throughout this report. Lidars have seen a growing range of applications in recent years, with the current drive towards autonomous, driverless cars as a significant driving factor . In a lidar, a waveform generator generates a laser waveform. Depending on the type of lidar, the setup can include a single laser or a master oscillator with multiple lasers or laser amplifiers. There are many types of lidars, however this report focuses on a particular type of 3D lidar with a rotating detector array, which measures azimuth and elevation angle, and range.

Kilde

The measurement is performed by the laser pulse being guided through transmit optics, traversing some medium, typically atmosphere, to a target. The laser pulse bounces off the target, and traverses the media again until receive optics captures the reflected pulse, guiding it to a detector or a detector array [27]. The laser and detector arrays rotate, taking multiple measurements as it scans the full 360 degree field of view. The range to a target can be determined based on the travel time of the laser pulse by

$$R = \frac{c}{2}(t_{rx} - t_{tx}) \tag{3.16}$$

where c is the speed of light in the intervening medium, t_{tx} and t_{rx} is the transmission and reception time of the laser pulse, respectively. The laser travels twice the distance to the target between transmission and reception time, hence the resulting range is halved. The range resolution is determined by

$$\Delta R = \frac{c}{2B} \tag{3.17}$$

where B is the system bandwidth, typically given by the transmit signal bandwidth.

3.2.1 Velodyne VLP-16 Lidar Specifications

The lidar used in this project is the Velodyne VLP-16, which is a small, real-time rotating lidar which steams data over a TCP/IP connection when it is powered up. The lidar has multiple return modes, where it can report either the strongest return signal, the last return signal, or both. The default return mode is to report the strongest return, and this mode is what is used in this project. The Velodyne VLP-16 features 16 laser/detector pairs mounted in a rotating housing, rapidly spinning to produce 360° 3D image.

Some features of the VLP-16 are [28]:

- Horizontal FOV of 360°.
- Weight of 830 grams.
- Rotational speed of 5-20 RPS (adjustable).
- Vertical FOV of 30° (+15° to -15°).
- Vertical angular resolution of 2°.

- Horizontal angular resolution of 0.1°-0.4°.
- Range of up to 100 meters (range depends on application).
- Accuracy ± 3 cm (typical).
- 903 nm wavelength laser.

The Velodyne VLP-16 is shown in figure 3.3.

Figure 3.3: The Velodyne VLP-16 lidar.

3.2.2 Lidar Sensor Model

A lidar is an optical detector, similar to a digital camera in that it responds to the intensity of the light hitting the detector, generating a voltage equal to the square of the intensity of the impinging light [27]. A rotating 3D lidar returns the intensity of the measured signal along with the range calculated from the travel time, as well as the azimuth (rotation) angle and the vertical angle of the point. The lidar has its own reference frame (in sperical coordinates), and it keeps track of its laser array angle, giving the azimuth angle of a point relative to some reference angle (typically determined by the manufacturer), and the vertical angle is referenced to the horizontal plane in the lidar frame of reference. The coordinates of a point in the lidar frame therefore consists of two angles, α and β, as well as the distance R from the lidar to the point. Figure 3.4 illustrates this.

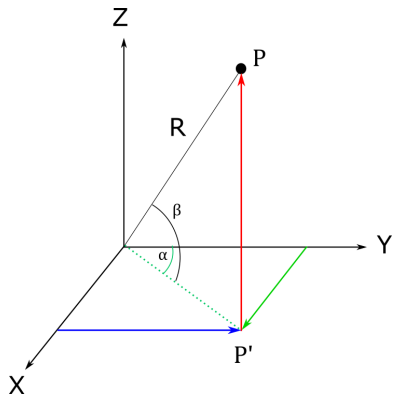
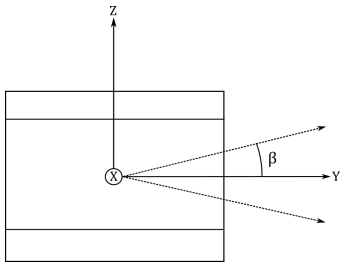


Figure 3.4: Lidar spherical coordinate system, with equivalent cartesian coordinates.

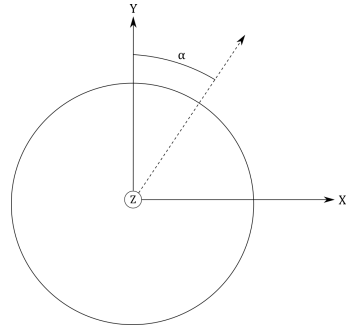
In figure 3.4, α is the azimuth angle, relative to the y-axis. β is the vertical/elevation angle, and R is the distance from the lidar to the point P. From figure 3.4, using basic trigonometry, we get that the transformation from spherical to cartesian coordinates is given by

$$\begin{aligned} X &= P' \sin\alpha = R\cos\beta\sin\alpha \\ Y &= P' \cos\alpha = R\cos\beta\cos\alpha \\ Z &= R\sin\beta \end{aligned} \tag{3.18}$$

The range returned by the Velodyne VLP-16 lidar is measured along a beam at known azimuth angles α, and the 16 laser/detector pairs are mounted vertically at 2° intervals from -15° to 15° referenced to the horizon. When the lidar is rotating at a frequency of 10 Hz the lasers fire at every 0.2°, with a total of 1800 laser firings for a full rotation. The theoretical maximum number of range measurements for a full 360° scan is therefore $360/0.2 \times 16 = 28800$ measurements, where it is assumed that every single laser beam is reflected, and subsequently detected at the lidar.



(a) Side view.



(b) Top view.

Figure 3.5: The lidar cartesian coordinate frame, with angles from the spherical coordinate frame drawn in.

A single reflection from the lidar can be represented in a cartesian coordinate frame centered in the lidar, shown in figure 3.5, using the transformation given in equation 3.18 as:

$$\mathbf{P}_{ij}^{l} = \begin{bmatrix} R_i \cos\beta_i \sin\alpha \\ R_i \cos\beta_i \cos\alpha \\ R_i \sin\beta_i \end{bmatrix} \tag{3.19}$$

Here, α is the azimuth angle in the scan, and β_i is the angle of the i^{th} vertical laser measured from the horizon. The superscript l denotes the cartesian frame centered in the lidar. R_i is the range measured at the given azimuth and elevation angle. This model assumes that the range measurement R_i reported by the lidar is exact, as well as that the angles β_i and α are known with perfect accuracy, which is generally not the case. Glennie et al. [29] did a calibration and stability analysis of the VLP-16 lidar, where six calibration parameters for each laser was identified. The proposed model was given as

$$\mathbf{P}_{ij}^{l} = \begin{bmatrix} (s^i R_i + D_o^i)\cos\beta_i(\sin\alpha\cos\delta_i - \cos\alpha\sin\delta_i) - H_o^i(\cos\alpha\cos\delta_i + \sin\alpha\sin\delta_i) \\ (s^i R_i + D_o^i)\cos\beta_i(\cos\alpha\cos\delta_i + \sin\alpha\sin\delta_i) - H_o^i(\sin\alpha\cos\delta_i - \cos\alpha\sin\delta_i) \\ (s^i R_i + D_o^i)\sin\beta_i + V_o^i \end{bmatrix}$$

where the calibration parameters are given in table 3.1.

s^i	The distance scale factor for laser i
D_o^i	The distance offset for laser i
β_i	The vertical rotation correction for laser i
δ_i	The horizontal rotation correction for laser i
V_o^i	the vertical offset from scanner frame origin for laser i
R_i	The raw distance measurement for laser i
α	The encoder angle measurement

Table 3.1: Lidar calibration parameters.

Note that β_i and α in table 3.1 correspond to β_i and α in equation 3.19. The Velodyne VLP-16 comes pre-calibrated from the factory, and Glennie et al. found that the factory supplied calibration is whithin the stated 3 cm ranging accuracy [29].

3.2.3 Coordinate Transformation

In order to use the measurements in conjunction with other measurements, such as those from a camera, they must be transformed to some common coordinate frame. As described in section 3.1.2, the measurement from the lidar can be transformed to another frame of reference by a rotation followed by a translation. If a navigation system gives the translation $\mathbf{t}_{wl}^{w}$ and the rotation $\mathcal{R}_{l}^{w}$ from the lidar frame to the world frame, the lidar measurement can be written in the world frame as

$$\mathbf{z}_{i}^{w} = \left[\mathbf{t}_{wl}^{w} + \mathcal{R}_{l}^{w}\mathbf{P}_{i}^{l}\right] \tag{3.20}$$

where $\mathbf{z}_{i}^{w}$ is a single point measurement given in the world frame.

3.2.4 The Lidar Point Cloud

For an entire 360° scan, the measurement returned by the lidar can be represented as an aggregation of single point measurements as

$$\mathbf{Z}^{w} = \begin{bmatrix} \mathbf{z}_{11}^{w} & \mathbf{z}_{12}^{w} & \dots & \mathbf{z}_{1j}^{w} \\ \mathbf{z}_{21}^{w} & \mathbf{z}_{22}^{w} & \dots & \mathbf{z}_{2j}^{w} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{z}_{i1}^{w} & \mathbf{z}_{i2}^{w} & \dots & \mathbf{z}_{ij}^{w} \end{bmatrix} \tag{3.21}$$

where, as before, i denotes the index of the vertical laser (from 1 to 16 for the velodyne VLP-16), and j is the index of the asimuth angle in the right-open interval $[0°, 360°[$. When the Velodyne VLP-16 rotates at 10 Hz, j ranges from 1 to 1800. $\mathbf{Z}^{w}$ represents a *point cloud* made by a full rotation of the lidar, decomposed in the cartesian world frame.

Chapter 4

Method

4.1 Sensors, Hardware and Processing Pipeline

The physical sensor setup used in the data collection is illustrated in figure 4.1. A HP zbook 15 running Ubuntu 16.04 with ROS was used as a data collection platform.

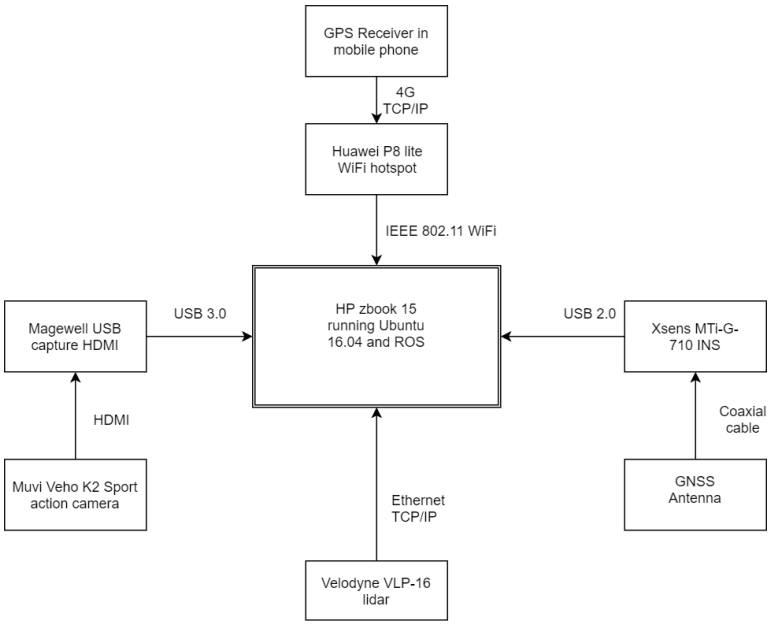


Figure 4.1: Sensor setup with physical connections.

The camera used to gather image data is a commercial off-the-shelf action camera by Muvi, named Veho K2 Sport. The reasoning for using such a camera was the affordable

price, relatively high resolution, and a wide angle of view. The specifications of the camera is presented in table 4.1. The specifications are pulled from the cameras user manual.

Model	MuVi Veho K2 Sport
Resolution	1920×1080 pixels (1080p), 1280×960 pixels (960p) or 1280×720 pixels (720p)
Framerate @ 1080p	24, 25, 48 or 50 frames per second
Framerate @ 960p	48 or 50 frames per second
Framerate @ 720p	50 or 100 frames per second
Angle of view	140° (wide), 120° (medium), 100° (narrow) or 80° (small)
External connections	HDMI micro, Mini USB
Internal storage	microSD card

Table 4.1: MuVi Veho K2 Sport specifications.

The settings used throughout the project is resolution at 1080p, with a framerate of 25 frames per second, with a wide angle of view (140°). A drawback with such a camera for computer vision applications, however, is the fact that it only sends images as a steady stream over its HDMI interface, with little options for the user to control the capture of individual frames. Moreover, HDMI is not a standard input interface on most computers, so a HDMI-to-USB3 adapter was purchased in order to be able to input the video stream on the computer. The HDMI-to-USB3 adapter is a USB Capture HDMI Gen 2 by the manufacturer Magewell. Using the camera with this interface, there is a noticeable processing delay between the camera and the computer.

The specifications of the computer used in the data gathering is listed in table 4.2, and the specifications for the computer used in post-processing is listed in table 4.3.

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

Table 4.2: Laptop computer specifications.

CPU	Intel Core i7-920 4×2.67 GHz
Memory	16 GB RAM
Graphics	NVIDIA GeForce GTX 1070
Storage	250 GB Solid State HDD
Operating System	Microsoft Windows 10 Enterprise

Table 4.3: Desktop computer specifications.

4.2 Visual Detection based on Faster R-CNN

The visual detection of boats in images used in this project is based on the framework of Faster R-CNN, developed by Ren et al. [30]. This framework was used due to the availability of pre-trained networks developed during a Master thesis project at NTNU [31]. A brief introduction to the framework of Faster R-CNN is given in this section, some background on how the implemented model was trained, and some details concerning the implementation of the network in MATLAB. For a detailed explanation of the full framework of Faster R-CNN the reader is referred to [30].

4.2.1 Faster R-CNN

Faster R-CNN is a region-based CNN, employing *Region Proposal Networks* (RPN) to form a single, unified network for object detection. Faster R-CNN is composed of two modules. The first module is a deep fully convolutional neural network which proposes

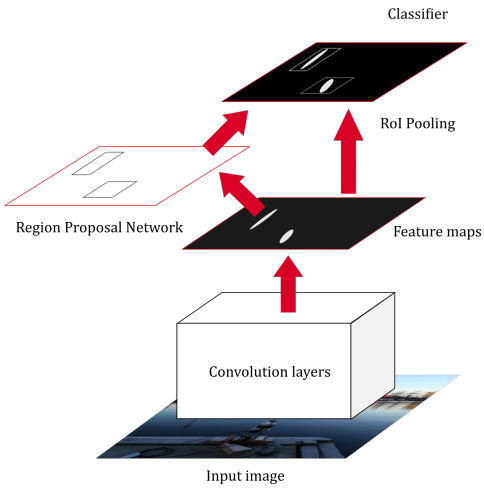


Figure 4.2: Faster R-CNN as a single unified network for object detection. The figure is adapted from [30].

regions, and the second module is the Fast R-CNN detector [32] that uses the proposed regions in classification. Figure 4.2 illustrates the system of Faster R-CNN. The Region Proposal Network takes an image of any size as inputs, feeding it through convolutional layers to generate feature maps. The area of activation is then fed into the RPN, which outputs object proposals in the form of rectangular regions, each with a objectness score. The objectness score is a measure of membership to a set of object classes versus background. The rectangular regions are subsequently fed into the classification layers where the classification score is obtained. In their paper, Ren et al. investigated two models for the classification network, the 5 convolutional layer deep Zeiler and Fergus model [33] dubbed ZF-net, and the 16 convolutional layer deep Simonyan and Zisserman model [34], dubbed VGG-16. In this project, the VGG-16 model is used, as this network achieved the best results in the Master thesis work of Tangstad [31].

4.2.2 Training and validation data

The network used in the project is the best-performing model developed in Tangstads thesis work [31]. This model is trained as a single-class classifier,

4.2.3 Implementation Aspects

A complete MATLAB implementation of Faster R-CNN is available on Github. In order to run Faster R-CNN, there are some software requirements that need to be met. The steps are listed in the Github repository for the framework [35], and will be repeated here for convenience. The steps are setup-dependent, depending on whether or not the framework will be executing on a GPU or a CPU, and also on the version of GPU used. The Faster R-CNN implementation available at the Github repository uses Caffe [36], which is a

deep-learning framework utilizing Nvidia CUDA for parallel computing, developed by Berkeley AI Research. Caffe is well documented online, with installation instructions, tutorials and more [37].

In order to use Faster R-CNN, a Caffe MEX file is needed. A MEX file is a dynamically linked subroutine that the MATLAB interpreter loads and executes, enabling MATLAB to utilize Caffe routines. A pre-compiled Caffe version for Nvidia CUDA version 6.5 is available in the repository, but for newer Nvidia GPUs a newer version of CUDA will need to be compiled along with Caffe. If a new compiling of Caffe is necessary, all the steps needed are provided in the Faster R-CNN branch of Caffe on Github [38]. If a GPU is utilized for parallel computing, a GPU with minimum 8 Gigabytes of memory is needed to run the VGG-16 version. In this project Caffe was recompiled with CUDA version 9.0, due to CUDA version 6.5 not supporting the newer Pascal architecture on the Nvidia GTX 1070 graphics card used. MATLAB is also needed to run the Faster R-CNN implementation. Once all requirements are met, the steps necessary before testing are

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

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

4.3 ROS Implementation

In order to capture, organize, timestamp and save the collected data, the open-source robotics framework of ROS was used. The main ROS client libraries are geared towards a UNIX/Linux platform, mainly due to the dependency on large collections of open-source software dependencies . Ubuntu Linux is a supported distribution, while others, such as Fedora Linux, Mac OS, and Windows, are designated as experimental and are supported by the community. Due to the simplicity of installation and ”out-of-the-box” functionality of ROS on Ubuntu Linux, the most recent long-term supported version of ROS, ROS Kinetic, was installed on a laptop running Ubuntu 16.04 LTS, which was used as a platform for running the various sensor drivers and collecting the data.

kilde

4.3.1 ROS Nodes and Data Flow

An overview of the implemented sensor driver and data processing nodes is illustrated in figure 4.3. The lidar nodes and the camera node are all based on open-source ROS packages freely available from the ROS community. The camera driver used is the `libuvc_camera` driver. This node reads the image stream from the camera over the USB port, providing a ROS interface for cameras meeting the USB Video Class standard [39]. This node publishes messages containing an image corresponding to a single frame of the video stream, with a time stamp corresponding to the arrival time of the image frame over USB. The source code is available at the ROS device drivers repository [40].

The lidar nodes are in the `velodyne_driver`-package from the ROS device drivers

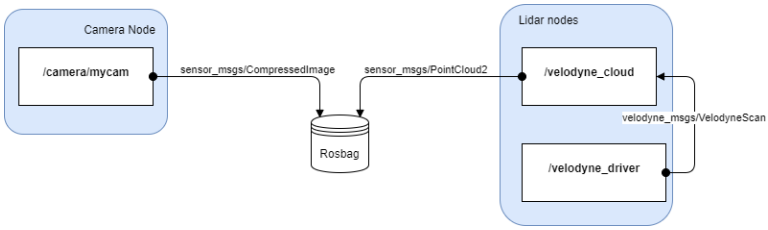


Figure 4.3: ROS nodes and topic flow.

repository [40]. This package provides two nodes, the first is `/velodyne_driver` for collecting all TCP packages sendt over ethernet from the lidar and publishing a message containing the raw data from a full 360-degree scan from the lidar. The second node, `/velodyne_cloud`, transforms the raw data into a point cloud given in the sensor frame in XYZ-coordinates, using the transformation in equation 3.19 in section 3.2.2. In addition to the coordinates of each point, the corresponding intensity of the reflected signal strength and the ring data is published **velodyne˙dr¡iver**.

The point cloud is then published as a ROS message. In addition to the coordinates of each point, the corresponding intensity of the reflected signal strength and the ring data is also included in this message [41].

Topic	Content
`sensor_msgs/PointCloud2`	Message containing point cloud data from the lidar
`velodyne_msgs/VelodyneScan`	Message containing raw data from a 360deg lidar scan
`sensor_msgs/CompressedImage`	Message containing a compressed image in .JPEG format

Table 4.4: ROS Topics used in data collection.

4.3.2 Rosbag

ROS provides a built-in set of tools for recording and playing back ROS topics, `rosbag` [42]. This toolset stores the topics of interest, specified as parameters in the command-line, to a file of the ROS bag format, with extension .bag. This toolset provides a simple method for storing data from experiments, and the playback function provides the option to simulate sensor inputs to the system without actually reading sensors. A basic use case,

where for example two topics named `/velodyne_points` and `/camera/image` are to be recorded, the proper command line input is

```
$ rosbag record -o file /pointcloud /camera/image
```

where `file` is the user-specified filename for the .bag file. ROS appends the filename with the system time automatically. Rosbag was used in recording all experimental data.

4.3.3 MATLAB implementation

MATLAB provides ROS interfaces via the Robotics System Toolbox, making it possible to use ROS topics directly in MATLAB. The standard ROS message types are readily available, and custom ROS messages can easily be compiled provided the user supplies the messages in a folder following the ROS convention, with a `package.xml` file listing all dependencies for the custom messages. The proper folder structure is shown in figure 4.4. For details on writing custom ROS messages and a corresponding `package.xml` file, the reader is referred to the ROS documentation, available online [43].

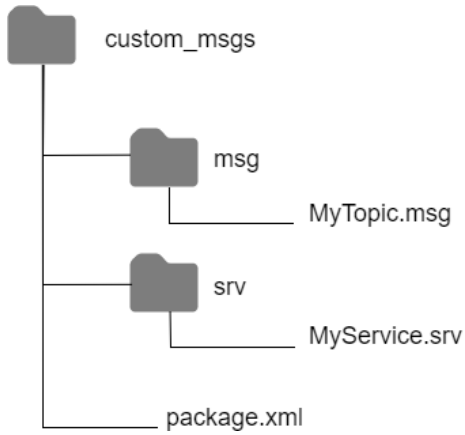


Figure 4.4: ROS custom message folder structure.

Building the custom messages can now be done by calling `rosgenmsg` in the MATLAB terminal with the path to the custom message files as a input parameter. The MATLAB command window gives prompts on how to add the custom files to the Java class path and the MATLAB path, which must be done in order to use the custom messages.

4.4 Camera Calibration

As explained in section 3.1.2, one needs the cameras intrinsic parameters in order to associate pixel coordinates with world coordinates. MATLABs Computer Vision System Toolbox includes the Camera Calibrator application [44], which was used in calibrating the camera. The Camera Calibrator application implements the calibration routine by

Zhang [45], described in appendix A.1. The steps taken for calibrating the camera using this application can be summarized as follows:

1. Prepare a checkerboard calibration pattern with known dimensions.

2. Capture 20-40 images of the checkerboard with the camera, from varying orientations.

3. Initiate the calibration.

4. Evaluate calibration accuracy.

5. Adjust parameters to improve accuracy (if necessary).

6. Export parameters as a cameraParameters object.

The accuracy in the calibration routine can be improved by removing outlier images, which contribute more to the reprojection errors than others. Figure 4.5 shows an example of a blurred image, which due to the error introduced to keypoint detection by the blurryness of the image causes a larger reprojection error.

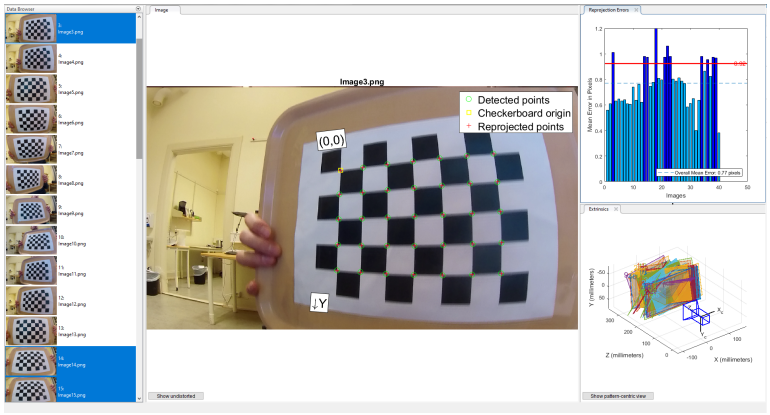
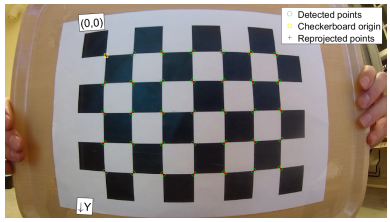
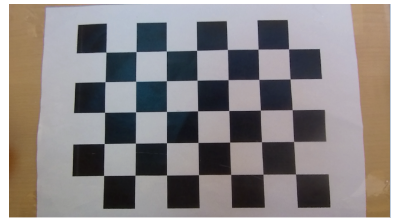


Figure 4.5: Camera Calibration app showing an outlier image.

The outliers can be removed by adjusting the threshold for mean pixel reprojection error shown in the upper right corner of figure 4.5. The calibration app also computes the extrinsic parameters and displays the calibration pattern planes in 3D, as seen in the lower right corner in figure 4.5. Having calibrated the camera, the radial distortion in the images can be removed, as seen in figure 4.6, where a unprocessed image with the calibration pattern key points overlaid can be seen next to the undistorted version.



(a) Original image with radial distortion.



(b) Undistorted image.

Figure 4.6: Distorted checkerboard image, along with the undistorted version.

The numerical values for the intrinsic matrix was found using this routine as

$$\mathcal{K} = \begin{bmatrix} \alpha & -\alpha\cot\theta & x_0 \\ 0 & \frac{\beta}{\sin\theta} & y_0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1068.7558 & -1.9413 & 893.7339 \\ 0 & 1066.6539 & 516.2405 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

4.4.1 Time Synchronization

Ikke ferdig, kanskje ikke nødvendig siden Kalibr-toolboxen ikke kunne brukes.

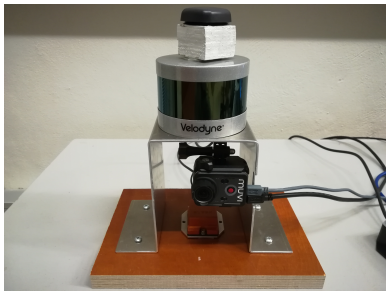
Chapter 5

Experiments

In order to get data to analyse and evaluate the proposed methodology, a series of experiments was performed using a small boat in the channel near Ravnkloa in Trondheim. The sensors were mounted on a rigid mount, and the boat performed a series of maneuvers in front of the sensors, in order to gather sensor data to be post-processed and analyzed. The data was saved in rosbags during the experiments, to be post-processed in ROS and MATLAB after the data gathering. The experiments were performed at Ravnkloa in the Nidelven river in Trondheim. This location was chosen due to easy access to the waterfront via a floating dock and access to electrical power outlets for the sensors and computer, courtesy of the Trondheim harbor authorities (Trondheim Havn). Figure 5.1 shows the experimental site in a satellite photo of downtown Trondheim. The circle in figure 5.1 illustrates the area where the boat maneuvers were performed, and the arrow illustrates where the sensors were placed.

Figure 5.1: The experiment site in downtown Trondheim. Image taken from Google Maps.

The sensors were mounted on a simple rig custom made for the purpose by the workshop at NTNU ITK. Figure 5.2 shows the sensors mounted on the rig, and the placement of the rig at the end of the floating dock at Ravnkloa.



(a) The sensor rig, with (from the top) GNSS antenna, lidar, camera and IMU installed.

(b) The sensor placement at Ravnkloa.

Figure 5.2: The sensor rig, and its placement at the floating dock at Ravnkloa.

The target boat used in the experiments is a 5.18 meters (17 imperial feet) long leisure craft, owned by two PhD students at NTNU. The boat is shown in figure 5.3 (the blue boat). The relative small size of the boat makes it a good test target for the proposed system, as detection of small leisure crafts will be important from a safety standpoint when an autonomous vessel is maneuvering in confined environments and close to shore, where such crafts are typically found.

Figure 5.3: The boat used as a target in the experiments.

5.1 Performed Maneuvers

In order to be able to compare detections in images with data from the lidar, and to gain insight into how well image detections of the boat corresponds to detections in the lidar point cloud, a series of maneuvers with different geometries were performed in the river around the sensor placement. Figure 5.4 illustrates the recorded maneuvers.

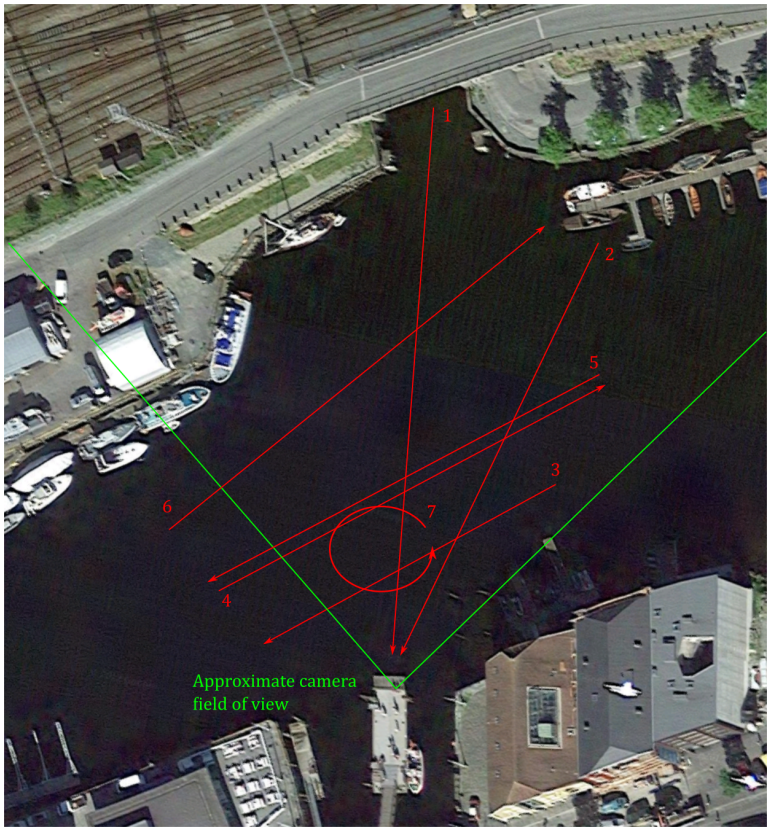


Figure 5.4: The recorded maneuvers overlaid a satellite image of Ravnkloa. Satellite image taken from Google maps.

The maneuvers, with reference to the indices in figure 5.4, were:

1. Boat arriving head on, at approximately 1.5 meters/second. Analysing detections for a vessel arriving head on is interesting from a collision detection and avoidance perspective.

2. Boat arriving head on, but from an angle relative to the center of the camera field of view. Speed approximately 1.5 meters/second.

3. Boat crossing in front of sensors, passing at a distance of 25 to 15 meters. Speed approximately 1 meter/second.

4. Boat crossing in front of sensors, passing at a distance of 25 to 45 meters. Speed approximately 1.5 meters/second.

5. Boat crossing in front of sensors, passing at a distance of 45 to 25 meters. Speed approximately 2.5 meters/second.

6. Boat crossing in front of sensors, passing at a distance of 45 to 65 meters. Speed approximately 1 meter/second.

7. Boat driving in circles in front of sensors, with a radius of approximately 10 meters, centered at a distance of 25 meters from the sensors. This provides a full 360° view of the boat.

The increased speed in experiment 6 was an attempt to generate wakes, in order to analyse the effects of wakes in the lidar data, as well as testing how detection in images compare to detections in the lidar data for a faster moving target. The imposed speed limit on the Nidelven river is 5 knots, or 2.57 meters/second, so this pass was close to the legal speed limit on the river. For all experiments, lidar data was stored as a point cloud with a update frequency of 10 Hz, while images was stored at a frame rate of 25 Hz.

Chapter 6

Results

In this chapter, the data generated from the experiments presented in the previous chapter are analyzed. The preprocessing methods used on the data is discussed, the metrics used in evaluating the correspondence between image detections and lidar data are introduced, and the results from each experiment are presented.

6.1 Post-processing

In order to associate boat detections in the images to world coordinates, the inverse calibration matrix from equation 3.10 using the values found for the matrix $\mathcal{K}$ in section 4.4 was multiplied with the pixel coordinates of the center point of the bounding boxes found by the faster R-CNN implementation. This associates the pixel coordinates with the normalized image plane presented in figure 3.2 in section 3.1.2, giving a direction to the center of the bounding box valid up to a scale factor. Due to the image data being in 2D coordinates, one is only able to associate a detection in an image with a direction, or ray, in 3D space. This ray was subsequently transformed from the camera frame, denoted c, to the lidar frame of reference, denoted l, using the rotation matrix

$$\mathcal{R}_c^l = \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & 0 \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos-\frac{\pi}{2} & 0 & \sin-\frac{\pi}{2} \\ 0 & 1 & 0 \\ -\sin-\frac{\pi}{2} & 0 & \cos-\frac{\pi}{2} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ -1 & 0 & 0 \\ 0 & -1 & 0 \end{bmatrix} \tag{6.1}$$

which is a rotation of $\frac{\pi}{2}$ radians around the cameras z-axis, followed by a rotation of $-\frac{\pi}{2}$ radians around the new y-axis, and the translation vector

$$\mathbf{t_c^l} = \begin{bmatrix} 0.05 & 0 & -0.097 \end{bmatrix}^T \tag{6.2}$$

given in meters. The rotation angles and translation vector were measured by hand.

The location used in the experiments gave many returns in the lidar point cloud from the surroundings, which can be considered clutter or noise in the measurements when the task is locating the boat. Figure 6.1 shows a point cloud of the surroundings projected

down to two dimensions, illustrating the many unwanted returns. The red circle indicates the area where the maneuvers were performed, and the black arrow indicate the flow of the Nidelven river.

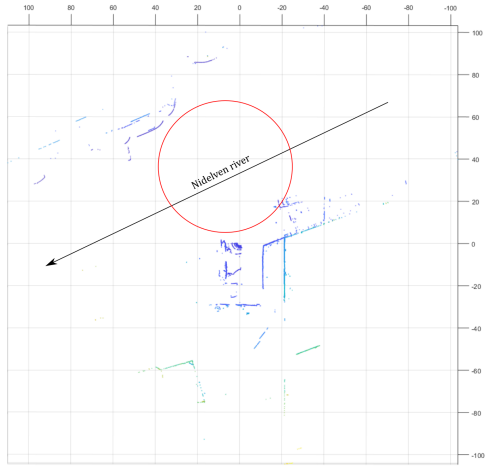


Figure 6.1: The lidar point cloud including surroundings at Ravnkloa, projected down to the xy-plane.

The returns from the boat in the lidar data was compared with detections from the Faster R-CNN network with the lidar data. In order to remove all returns from the lidar that were known to originate from the surroundings were removed, using a ”pass through” filter removing all points which lies outside a specified rectangular region in the xy-plane, keeping only the returns from the designated testing area. The point cloud was also reduced to two dimensions by projecting all points down to the plane $z = 0$. In addition, the point cloud returns were limited in post-processing to only those lying in the cameras field of view. Once the point cloud was reduced to two dimensions, and all returns originating from the surroundings were removed, k-Means clustering (with k=1) was used to find the centroid of the points returned by the boat. An overview of the post-processing pipeline is given in figure 6.2.

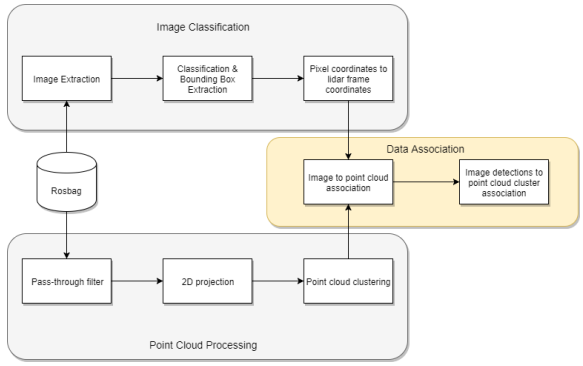


Figure 6.2: The data post-processing pipeline.

An example of a image detection associated with the point cloud cluster center is shown in figure 6.3. The line from the origin to the cluster center is the projection of the bounding box centroid of the detected boat in the image. The corresponding image generating the detection is shown in figure 6.4.

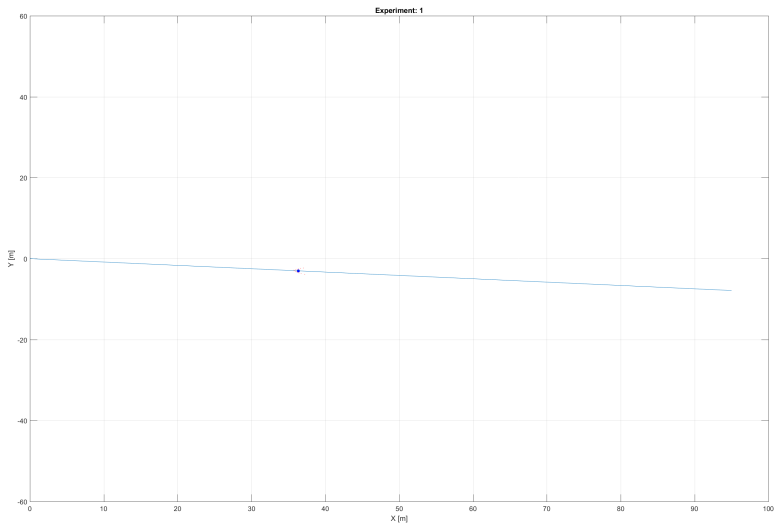


Figure 6.3: Point cloud cluster, with image detection drawn in.

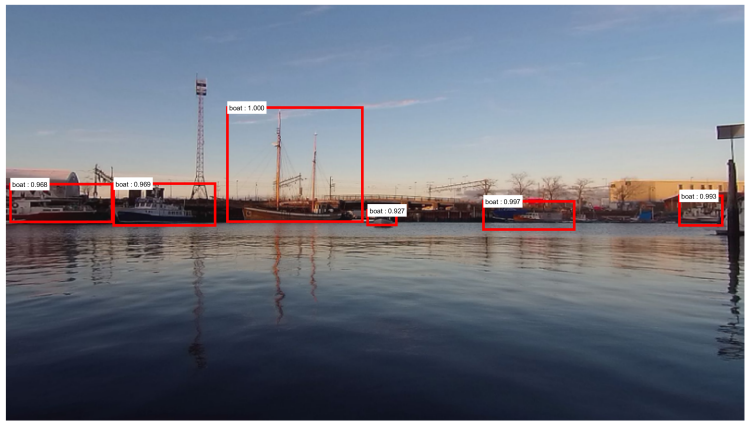


Figure 6.4: Corresponding image from the camera, with bounding boxes corresponding to detected boats overlaid.

6.2 Metrics

Faster R-CNN gives the bounding boxes for its detections, along with the score for the detection, indicating the ”confidence” in the classification. The threshold for the score was set to 0.6, where 1.0 indicates absolute certainty (from the detectors perspective) and 0 indicates zero confidence. The 0.6 threshold was the default set in the faster R-CNN implementation from the Github repository. The detection threshold has not been analyzed with particular attention in this project.

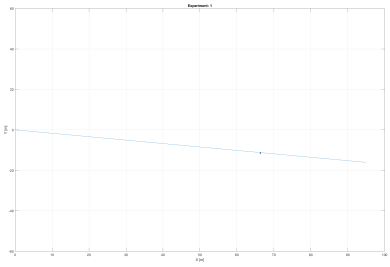
As a metric in determining whether or not a detection from the convolutional net corresponds with the boat, the minimum distance between the ray projected by the centroid of the bounding box and the centroid of the point cloud cluster corresponding to the boat was used. This distance was set to 2.5 meters, which is approximately half the length of the boat used in the experiments. This value was heuristically determined through experimentation, and was chosen as a compromise between the target boat being detected versus boats in the background counting as a detection of the target boat. Due to the geometry of the boat, for some maneuvers the majority of the returns found by the lidar come from the front part of the boat, while the detections in the image are more centered on the projection of the boat into the image plane. The 2.5 meters distance also allow for some time difference between the images and the point cloud, as the data is not time stamped at capture time, but at arrival time in the computer, not accounting for intermediary processing and transmission delay. The images corresponding to a particular point cloud were found by searching through the image set and selecting the image with the temporally closest

time stamp, since the time offset between the lidar and the camera is unknown. Lastly, the transformation between the camera frame and the lidar frame is hand-measured and assumes that the camera z-axis is perfectly aligned with the lidar x-axis, not taking into account misalignments. The minimum distance of 2.5 meters allows for some leeway in the alignment of the sensors. In order to avoid multiple detections in cases where the target boat pass in front of a boat in the background, only the detection closest to the point cloud cluster center is considered. For each experiment, the total number of image detections corresponding to a point cloud are calculated. The accumulated point cloud cluster centers are plotted as a track for each experiment, and color coded to indicate if there is a detection in a image corresponding to that particular point cloud cluster center.

6.3 Experimental Results

6.3.1 Experiment 1

Experiment 1 was performed by the vessel arriving head on towards the sensor setup, see figure 5.4. The first return from the boat was attained at a range of 67.39 meters, shown in figure 6.5.



(a) First return from the lidar.

(b) Corresponding image from the camera, with bounding boxes overlaid. The target is the smallest rectangle in the center of the image.

Figure 6.5: The first return from the lidar, along with the corresponding detections in the image.

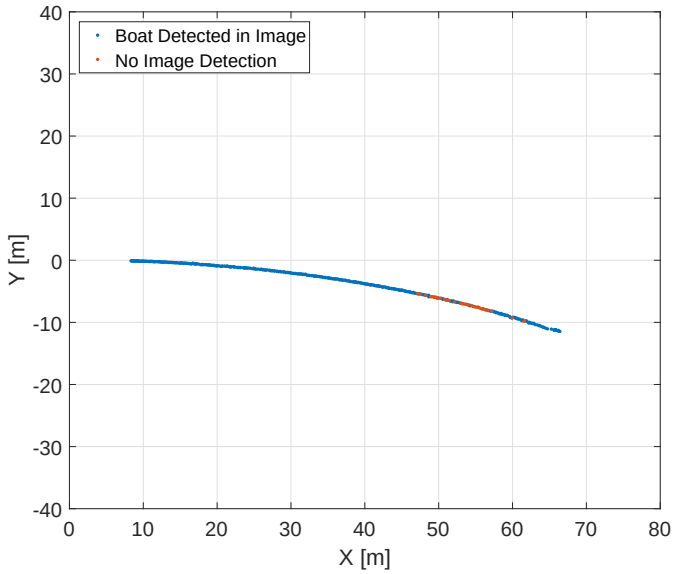
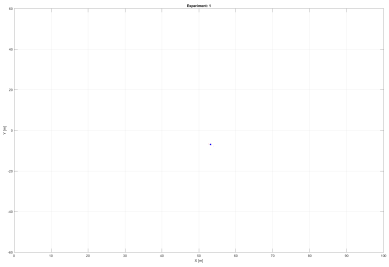


Figure 6.6: Track generated by point cloud cluster center in experiment 1.

As seen in figure 6.6, there are some missed detections in the image data around $x = 50$ meters. Looking at figure 6.7, it's hard to determine an exact reason for this. One possibility may be that a combination of lighting conditions and background kept the detection score below the threshold of 0.6 for some time.



(a) Lidar return.

(b) Corresponding image frame.

Figure 6.7: Missed detection in experiment 1.

As figure 6.6 shows, once the vessel was within 40 meters, every single point cloud had a corresponding detection of the vessel. The total number of point clouds, total number of detections and the fraction of boat detections in images over the total number of point clouds for experiment 1 is shown in table 6.1.

Number of point clouds	409
Number of detections	365
Detections/Point clouds	0.8924

Table 6.1: Data from experiment 1.

6.3.2 Experiment 2

Experiment 2 was performed by the vessel approaching the sensor location from an angle, see figure 5.4. The track generated by the point cloud cluster centers are shown in figure 6.8.

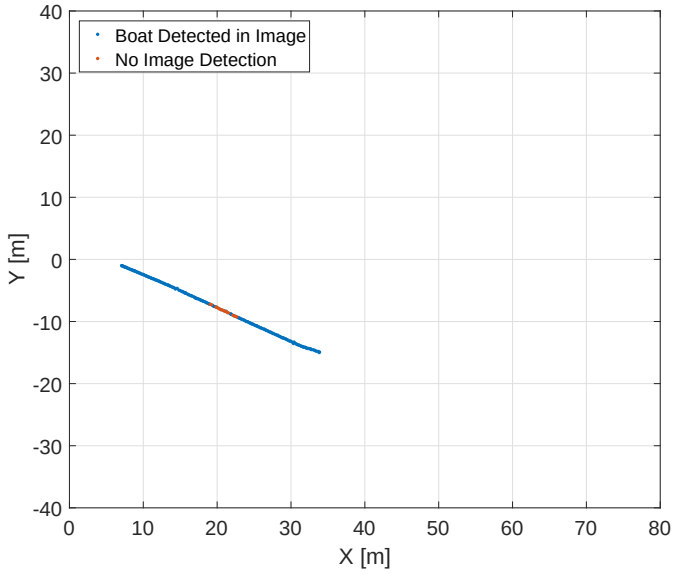
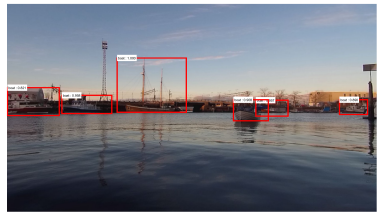


Figure 6.8: Track generated by point cloud cluster center in experiment 2.

As seen in figure 6.8, there is an interval around $x = 20$ meters where the image detections dissapear. Figure 6.9a shows the reason for this, as a ship in the background is influencing the detection of the target. In figure 6.9b, the target vessel has moved enough for the two boats to be detected separately.

(a) Background ship confusing the detection of the target.



(b) Large enough separation to be detected as two separate ships

Figure 6.9: Two examples of multiple ships being detected and classified as one.

The total number of point clouds, total number of detections and the fraction of boat detections in images over the total number of point clouds for experiment 2 is shown in table 6.2.

Number of point clouds	203
Number of detections	186
Detections/Point clouds	0.9163

Table 6.2: Data from experiment 2.

6.3.3 Experiment 3

Experiment 3 was performed by the boat passing across of the sensors, at a distance of approximately 25 to 15 meters in the x-direction. As seen in figure 6.10, the target is detected in almost all of the images corresponding to the point clouds in the track. There are some missed detections near the end of the track, around $x = 20$ and $y = 10$ in figure 6.10. As seen in figure 6.11, there are boats in the background at this part of the track.

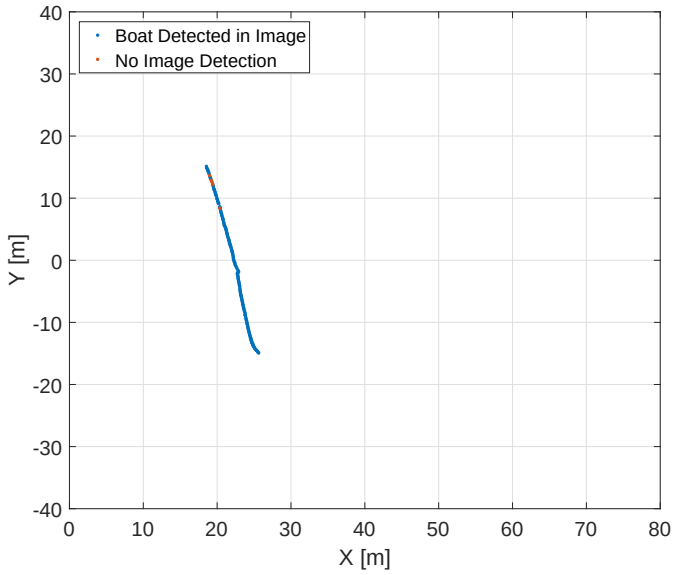
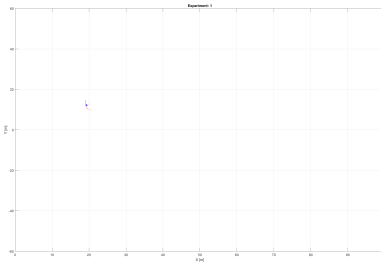


Figure 6.10: Track generated by point cloud cluster center in experiment 3.

(a) Background ship confusing the detection of the target.



(b) Corresponding point cloud.

Figure 6.11: Background confusing the detection of the target in the image in experiment 3.

The total number of point clouds, total number of detections and the fraction of boat detections in images over the total number of point clouds for experiment 3 is shown in table 6.3.

Number of point clouds	214
Number of detections	207
Detections/Point clouds	0.9673

Table 6.3: Data from experiment 3.

6.3.4 Experiment 4

Experiment 4 was performed by the boat passing across the sensor setup, this time at a distance of approximately 30 to 40 meters in the x-direction. Figure 6.12 shows the track generated by the point cloud cluster centers.

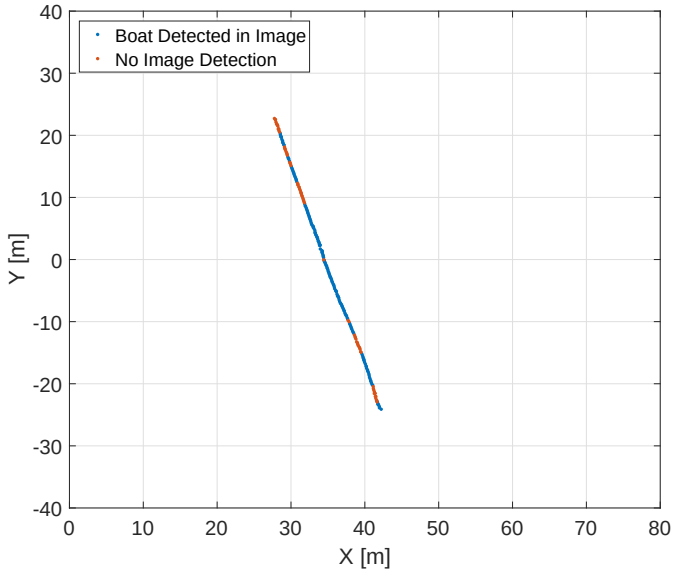
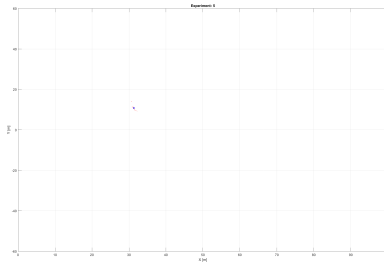


Figure 6.12: Track generated by point cloud cluster center in experiment 4.

Again, we see from figure 6.12 that there are certain parts of the track where the image detections suffer. Figure 6.13 shows that there are boats in the background of the image at this part of the track. Referring to figure 6.13a and comparing with figure 6.12, we see that for all parts of the track where image detections are missing, there are boats in the background of the image.

(a) Detections in the image at the start of the track.



(b) Corresponding point cloud.

Figure 6.13: Background confusing the detection of the target in the image in experiment 4.

The total number of point clouds, total number of detections and the fraction of boat detections in images over the total number of point clouds for experiment 4 is shown in table 6.4.

Number of point clouds	240
Number of detections	171
Detections/Point clouds	0.7125

Table 6.4: Data from experiment 4.

6.3.5 Experiment 5

In experiment 5, the boat was passing across the sensor setup at a distance of approximately 45 to 30 meters in the x-direction, this time with a speed of approximately 2.5 meters/second. As seen in figure 6.14, this experiment generated few image detections, compared to the previous experiments.

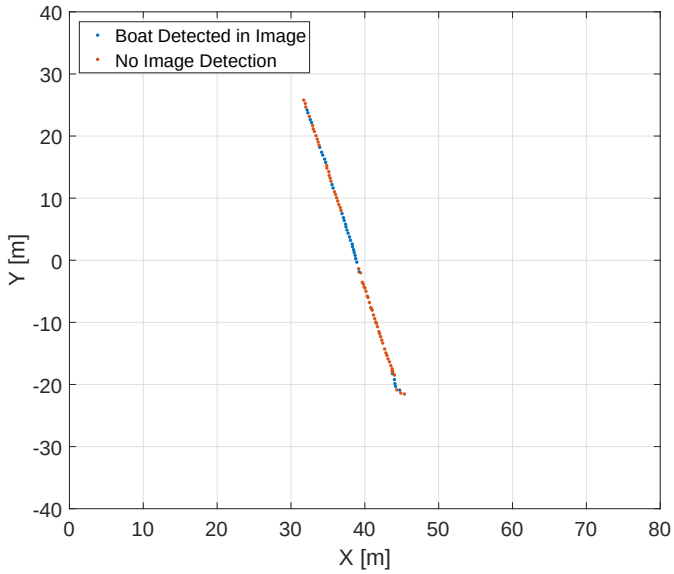
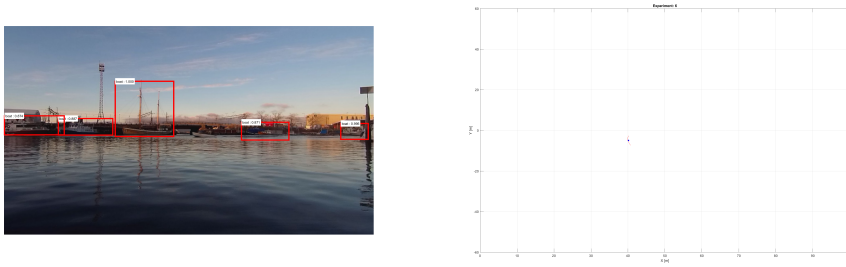


Figure 6.14: Track generated by point cloud cluster center in experiment 5.

Figure 6.15 shows an image along with the corresponding point cloud for a case where no boat was detected during experiment 5. As seen in figure 6.15a, there are no boats detected in the background around the target boat in this case.



(a) Target boat not detected. **(b)** Corresponding point cloud.

Figure 6.15: Target not detected in image in experiment 5.

The total number of point clouds, total number of detections and the fraction of boat detections in images over the total number of point clouds for experiment 5 is shown in table 6.5.

Number of point clouds	94
Number of detections	33
Detections/Point clouds	0.3511

Table 6.5: Data from experiment 5.

6.3.6 Experiment 6

In experiment 6, the boat was passing across the sensor setup at a distance between approximately 45 to 65 meters in the x-direction. Figure 6.16 shows the recorded track for this experiment.

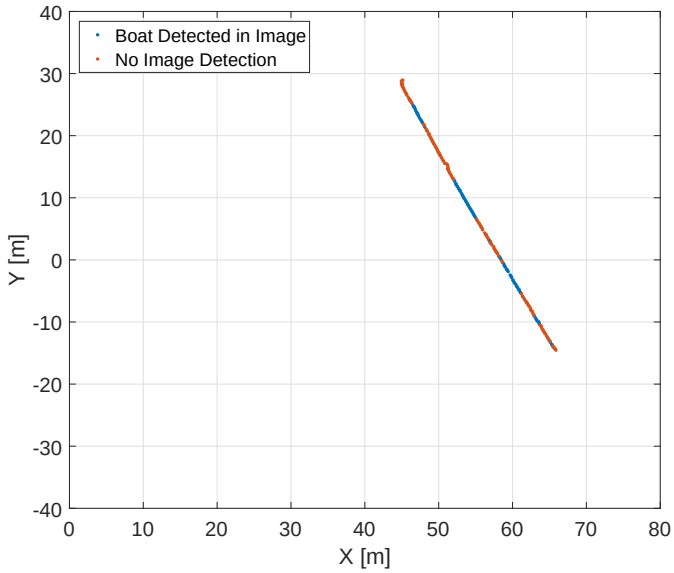
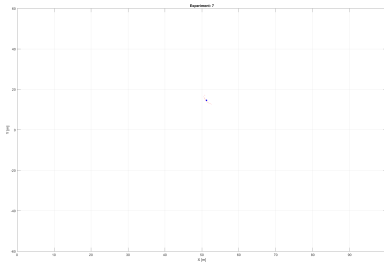


Figure 6.16: Track generated by point cloud cluster center in experiment 6.

Figure 6.17 shows an example of an image where the target was not detected in the image, along with the corresponding point cloud. The target can be seen between bounding box number two and three in figure 6.17a, counting from the left.

(a) Target boat not detected.



(b) Corresponding point cloud.

Figure 6.17: Target not detected in image in experiment 6.

The total number of point clouds, total number of detections and the fraction of boat detections in images over the total number of point clouds for experiment 6 is shown in table 6.6.

Number of point clouds	266
Number of detections	114
Detections/Point clouds	0.4286

Table 6.6: Data from experiment 6.

6.3.7 Experiment 7

In experiment 7, the boat moved in a circular pattern, as seen in the track in figure 6.18. The start of the track, and the experiment, is the top most data point in figure 6.18, subsequently entering the cicular pattern. The boat completed three full circles during the experiment. As seen in the track in figure 6.18, the missed image detections occur at the same part of the track for each completed circle, at both sides of the circle lying on a ray originating in the origin, as well as at the top of the circle in figure 6.18. The missed image detections at the start of the track is due to the boat not being fully in the field of view of the camera at the start of the recording.

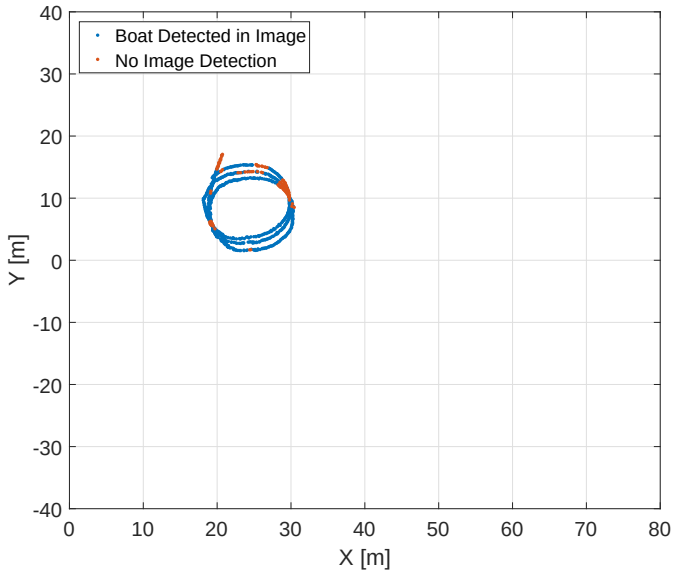
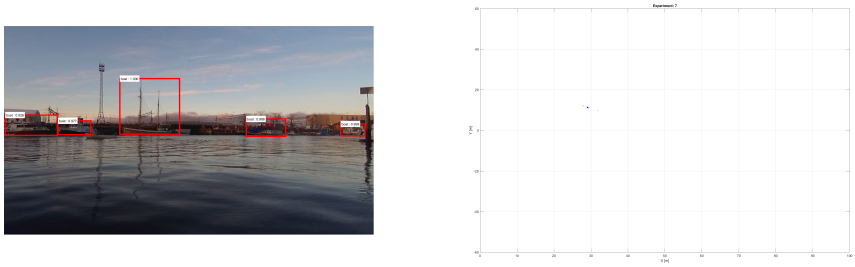


Figure 6.18: Track generated by point cloud cluster center in experiment 7.

Figure 6.19 shows an image and the corresponding point cloud where the boat was not detected at the far side of the circle from the sensors. As seen in figure 6.19a, this point appears to lie between two boats detected in the background.



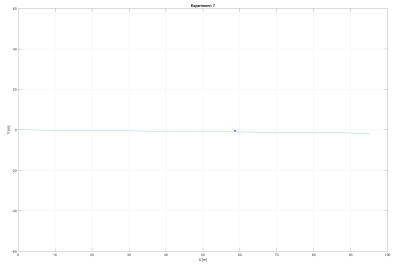
(a) Target boat not detected.

(b) Corresponding point cloud.

Figure 6.19: Target not detected in image in experiment 7.

An example image and corresponding point cloud from the topmost part of the track in figure 6.18 is shown in figure 6.20. At this part of the track, the boat is between the two leftmost boats in the background of the image.

(a) Target boat not detected.



(b) Corresponding point cloud.

Figure 6.20: Another example of target not detected in image in experiment 7.

The total number of point clouds, total number of detections and the fraction of boat detections in images over the total number of point clouds for experiment 6 is shown in table 6.6.

Number of point clouds	792
Number of detections	663
Detections/Point clouds	0.8371

Table 6.7: Data from experiment 7.

6.3.8 Faster R-CNN Performance

The processing performance of the Faster R-CNN network was analyzed for each data set produced in the experiments. The time taken to process the images for an entire experiment was measured, and dividing by the number of images gives an indication on the average throughput of the network for the hardware and software used. The results are presented in table 6.8.

Experiment	Number of images	Processing time	Average processing time, single image
1	2010	378.282 seconds	0.188 seconds
2	1600	244.480 seconds	0.153 seconds
3	1638	251.597 seconds	0.154 seconds
4	1629	251.518 seconds	0.154 seconds
5	1532	231.056 seconds	0.151 seconds
6	855	130.131 seconds	0.152 seconds
7	2830	590.621 seconds	0.209 seconds

Table 6.8: Faster R-CNN throughput.

The total average processing time for a single image, averaging over all experiments, is 0.166 seconds.

Chapter 7

Discussion

The results presented are weakened by the fact that no position ground truth, using e.g. GNSS, was generated throughout the experiments, and the results presented in the previous chapter assumes that the point cloud data generated represent the true location of the boat. When examining the data, however, there is reason to believe that this is not inaccurate. The tracks generated by the point cloud cluster centroids are a realistic representation of the maneuvers performed, and the tracks are coherent with no observable large jumps or discontinuities. No returns not believed to originate from the target boat were found in the lidar data. It should be noted that the water surface was calm at the time of the experiments, rough seas could potentially produce a different scenario.

7.0.1 Detection of Boats in Images

As the results show, there are some places where the detection of boats in the images suffer. Upon examining the images where detection largely failed, for many of them this is the case when boats in the background are close to the target boat in the image, leading them to be detected as one. Figure 7.1 shows two cases of this.

(a) Three ships detected as one (far left of the image).

(b) Two ships detected as one (second bounding box from the left).

Figure 7.1: Two examples of multiple ships being detected and classified as one.

This could have been avoided by using e.g. background subtraction, removing the static background in the image [23], so that no boats other than the target boat was visible in the images. It is, however, an interesting observation and a potential weakness in the proposed visual detection model in a realistic scenario.

In evaluating the classification results for the visual detection, it would be useful to have hand-generated a ground truth bounding box for the target boat. The large size of the datasets made this prohibitively time consuming, as a total of 12094 images were generated. The goal of the project was to compare detections between the sensors, and spot checks taken

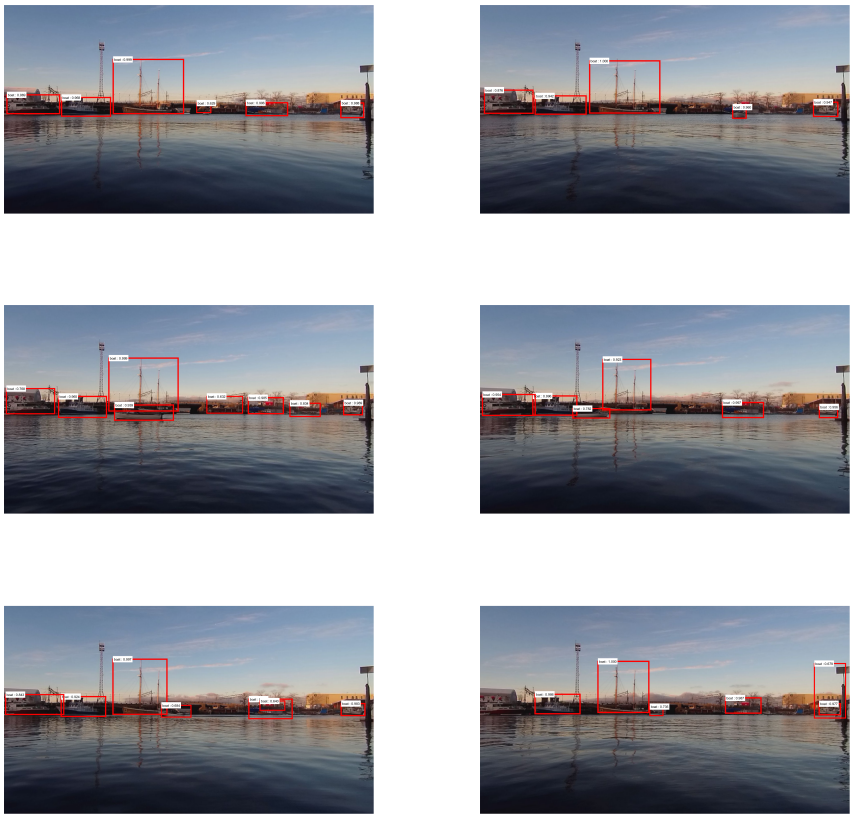


Figure 7.2: Examples of successfull detections of the target boat.

Chapter 8

Conclusion

Conclusion, and suggestion of further work.

Appendix A

Appendix

A.1 Geometric Camera Calibration

Camera calibration is a necessary step in computer vision applications in order to extract metric information from 2D images. The intrinsic and extrinsic parameters of a camera can be estimated from the image positions of scene features whose geometry in 3D space is known with very good precision [23]. The calibration setup usually consists of two or three orthogonal planes with a known calibration pattern (such as a chessboard). In fact, if the 3D reference points lie in the same plane, calibration will not work [46]. This setup, however, requires an expensive calibration apparatus, and an elaborate setup. Zhang proposed an alternative, flexible approach that doesn't require knowledge of the 3D space geometry of the calibration setup [45]. The technique described by Zhang only requires the camera to observe a planar pattern shown at at least two different orientations. The calibration procedure performed in this project is based on this work, and a introduction to the theory is presented in this section.

A.1.1 Perspective Projection between the Model Plane and Its Image

In order to be consistent with the notation used in Zhang's paper, let's rewrite equation 3.13 as

$$s\tilde{\mathbf{m}} = \mathbf{A}\begin{bmatrix}\mathcal{R} & \mathbf{t}\end{bmatrix}\tilde{\mathbf{M}}, \quad \text{where} \quad \mathbf{A} = \begin{bmatrix}\alpha & \gamma & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1\end{bmatrix} \tag{A.1}$$

where $\tilde{\mathbf{m}} = \begin{bmatrix}u & v & 1\end{bmatrix}^T$ is the image plane coordinates, $\tilde{\mathbf{M}} = \begin{bmatrix}X & Y & Z & 1\end{bmatrix}^T$ as the 3D point coordinates expressed in the world frame, s is an arbitrary scale factor, $\begin{bmatrix}u_0 & v_0\end{bmatrix}^T$ describe the position of the image center in pixel units, α and β are the scale factors in the u and v axes, and γ describes the skew between the two image axes. Now, without loss of generality, assume that the model plane is located at $Z = 0$, and let $\mathbf{r}_i$ denote the ith

column of $\mathcal{R}$. Equation A.1 can then be expressed as

$$s\begin{bmatrix}u\\v\\1\end{bmatrix} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{r}_3 & \mathbf{t}\end{bmatrix}\begin{bmatrix}X\\Y\\0\\1\end{bmatrix} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix}\begin{bmatrix}X\\Y\\1\end{bmatrix} \tag{A.2}$$

By slight abuse of notation, let $\tilde{\mathbf{M}}$ still represent a point on the model plane, but since Z is always zero, we write $\tilde{\mathbf{M}} = \begin{bmatrix}X\\Y\\1\end{bmatrix}$. With this, a point on the model plane and its corresponding image coordinates are related by a projective transformation (also known as a *homography*) matrix $\mathbf{H}$:

$$s\tilde{\mathbf{m}} = \mathbf{H}\tilde{\mathbf{M}}, \quad \text{where} \quad \mathbf{H} = \mathbf{A}\begin{bmatrix}\mathbf{r}_1 & \mathbf{r}_2 & \mathbf{t}\end{bmatrix} \tag{A.3}$$

The 3×3 matrix $\mathbf{H}$ is only defined up to a scale factor. Denoting $\mathbf{H} = \begin{bmatrix}\mathbf{h}_1 & \mathbf{h}_2 & \mathbf{h}_3\end{bmatrix}$, and knowing that since $\mathbf{r}_1^T\mathbf{r}_2 = 0$, we have

$$\mathbf{h}_1^T(\mathbf{A}^{-1})^T\mathbf{A}^{-1}\mathbf{h}_2 = 0 \tag{A.4}$$

and

$$\mathbf{h}_1^T(\mathbf{A}^{-1})^T\mathbf{A}^{-1}\mathbf{h}_1 = \mathbf{h}_2^T(\mathbf{A}^{-1})^T\mathbf{A}^{-1}\mathbf{h}_2 \tag{A.5}$$

These equations are the two basic constraints on the intrinsic parameters, given a single projective transformation. Due to the fact that a projective transformation has 8 degrees of freedom, and there are six extrinsic parameters, one can only obtain two constraints on the intrinsic parameters [45].

A.1.2 Solving the Calibration Problem

Let

$$\begin{aligned}\mathbf{B} = (\mathbf{A}^{-1})^T\mathbf{A}^{-1} &= \begin{bmatrix}B_{11} & B_{12} & B_{13}\\B_{12} & B_{22} & B_{23}\\B_{13} & B_{23} & B_{33}\end{bmatrix}\\ &= \begin{bmatrix}\frac{1}{\alpha^2} & -\frac{\gamma}{\alpha^2\beta} & \frac{v_0\gamma-u_0\beta}{\alpha^2\beta}\\ -\frac{\gamma}{\alpha^2\beta} & \frac{\gamma^2}{\alpha^2\beta^2}+\frac{1}{\beta^2} & -\frac{\gamma(v_0\gamma-u_0\beta)}{\alpha^2\beta^2}-\frac{v_0}{\beta^2}\\ \frac{v_0\gamma-u_0\beta}{\alpha^2\beta} & -\frac{\gamma(v_0\gamma-u_0\beta)}{\alpha^2\beta^2}-\frac{v_0}{\beta^2} & \frac{(v_0\gamma-u_0\beta)^2}{\alpha^2\beta^2}+\frac{v_0^2}{\beta^2}+1\end{bmatrix}\end{aligned} \tag{A.6}$$

$\mathbf{B}$ is symmetric, defined by a 6D vector

$$\mathbf{b} = \begin{bmatrix}B_{11} & B_{12} & B_{22} & B_{13} & B_{23} & B_{33}\end{bmatrix}^T \tag{A.7}$$

Defining the i^{th} column vector of $\mathbf{H}$ as $\mathbf{h}_i = \begin{bmatrix}h_{i1} & h_{i2} & h_{i3}\end{bmatrix}^T$, we can rewrite equation A.4 as

$$\mathbf{h}_i^T\mathbf{B}\mathbf{h}_j = \mathbf{v}_{ij}^T\mathbf{b} \tag{A.8}$$

where

$$\begin{aligned}\mathbf{v}_{ij} =&[h_{i1}h_{j1} \;\; h_{i1}h_{j2} + h_{i2}h_{j1} \;\; h_{i2}h_{j2} \\ & h_{i3}h_{j1} + h_{i1}h_{j3} \;\; h_{i3}h_{j2} + h_{i2}h_{j3} \;\; h_{i3}h_{j3}]^T\end{aligned} \tag{A.9}$$

The two constraints on the intrinsic parameters can then be written as two homogeneous equations in $\mathbf{b}$ as

$$\begin{bmatrix} \mathbf{v}_{12}^T \\ (\mathbf{v}_{11} - \mathbf{v}_{22})^T \end{bmatrix} \mathbf{b} = \mathbf{0} \tag{A.10}$$

If n images of the model plane are observed, we can stack n equations as equation A.10 and get

$$\mathbf{V}\mathbf{b} = \mathbf{0} \tag{A.11}$$

where $\dim(\mathbf{V}) = 2n \times 6$. For $n \geq 3$ we will generally have a unique solution b given as the eigenvector og $\mathbf{V}^T\mathbf{V}$ associated with the smallest eigenvalue, defined up to a scale factor. Once $\mathbf{b}$ is estimated, one can find all the intrinsic parameters in $\mathbf{A}$ from $\mathbf{b}$ by

$$\begin{aligned} v_0 &= \frac{B_{12}B_{13} - B_{11}B_{23}}{B_{11}B22 - B_{12}^2} \\ \lambda &= B_{33} - \frac{B_{13}^2 + v_0(B_{12}B_{13} - B_{11}B_{23})}{B_{11}} \\ \alpha &= \sqrt{\frac{\lambda}{B_{11}}} \\ \beta &= \sqrt{\frac{\lambda B_{11}}{B_{11}B22 - B_{12}^2}} \\ \gamma &= -\frac{B_{12}\alpha^2\beta}{\lambda} \\ u_0 &= \frac{\gamma v_0}{\alpha} - \frac{B_{13}\alpha^2}{\lambda} \end{aligned} \tag{A.12}$$

Now that $\mathbf{A}$ is known, the extrinsic parameters can easily be computed from equation A.3 as

$$\begin{aligned} \mathbf{r}_1 &= \lambda\mathbf{A}^{-1}\mathbf{h}_1 \\ \mathbf{r}_2 &= \lambda\mathbf{A}^{-1}\mathbf{h}_2 \\ \mathbf{r}_3 &= \mathbf{r}_1 \times \mathbf{r}_2 \\ \mathbf{t} &= \lambda\mathbf{A}^{-1}\mathbf{h}_3 \end{aligned} \tag{A.13}$$

where $\lambda = 1/||\mathbf{A}^{-1}\mathbf{h}_1||$. The parameters can be further refined through maximum likelihood inference. Assuming that the image points are corrupted by independent and identically distibuted (i.i.d.) noise, the maximum likelihood estimate can be obtained by minimizing

$$\sum_{j=1}^{n}\sum_{i=1}^{m} ||\mathbf{m}_{ij} - \hat{\mathbf{m}}(\mathbf{A}, \mathcal{R}_i, \mathbf{t}_i, \mathbf{M}_j)||^2 \tag{A.14}$$

where $\hat{\mathbf{m}}(\mathbf{A}, \mathcal{R}_i, \mathbf{t}_i, \mathbf{M}_j)$ is the projection of point $\mathbf{M}_j$ in image i as described in equation A.3.

A.1.3 Radial Lens Distortion

In the previous sections lens distortion models has not been discussed. A typical camera, however, will to a varying degree exhibit lens distortion, especially radial distortion. Figure A.1 illustrates positive and negative radial distortion, known as barrel distortion and pincushion distortion, respectively. Zhang only considered radial distortion in his work, the argument being that more elaborate models not only did not improve the result, but could also produce numerical instability [47]. The interested reader is referred to [48]–[50] for a more thorough description of distortion models considering effects as decentering and thin prism distortion in addition to radial distortion.

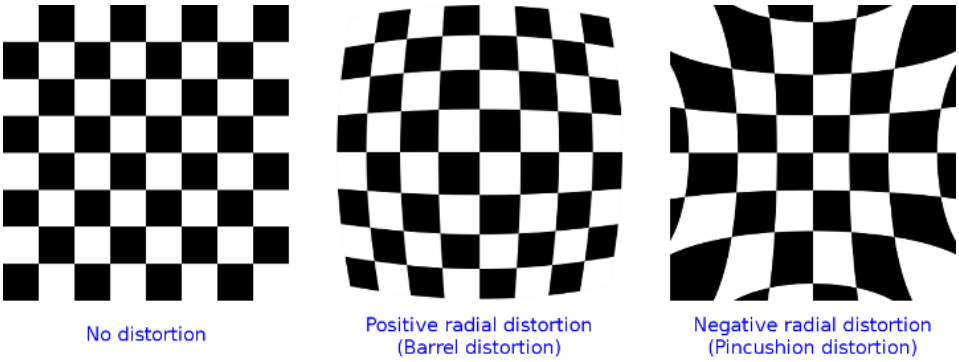


Figure A.1: Examples of radial distortion. [1]

Considering only radial distortion, let (u, v) represent the ideal (distortion-free) pixel image coordinates, and $(\breve{u}, \breve{v})$ the corresponding observed real coordinates. Then, using equation A.1 and assuming $\gamma = 0$, we have [47]

$$\breve{u} = u + (u - u_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \tag{A.15}$$

$$\breve{v} = v + (v - v_0)[k_1(x^2 + y^2) + k_2(x^2 + y^2)^2] \tag{A.16}$$

where k_1 and k_2 are coefficients of the radial distortion. This can be rewritten as

$$\begin{bmatrix} (u - u_0)(x^2 + y^2) & (u - u_0)(x^2 + y^2)^2 \\ (v - v_0)(x^2 + y^2) & (v - v_0)(x^2 + y^2)^2 \end{bmatrix} \begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \begin{bmatrix} \breve{u} - u \\ \breve{v} - v \end{bmatrix} \tag{A.17}$$

In a similar fashion as for equation A.11, for m points in n images, this can be stacked to obtain $2mn$ equations in matrix form as $\mathbf{Dk} = \mathbf{d}$, $\mathbf{k} = \begin{bmatrix} k_1 & k_2 \end{bmatrix}^T$. The linear least-squares solution is given by

$$\mathbf{k} = (\mathbf{D}^T\mathbf{D})^{-1}\mathbf{D}^T\mathbf{d} \tag{A.18}$$

Once k_1 and k_2 are estimated, equation A.14 can be extended to estimate the complete set of parameters including k_1 and k_2 by minimizing

$$\sum_{j=1}^{n}\sum_{i=1}^{m} ||\mathbf{m}_{ij} - \breve{\mathbf{m}}(\mathbf{A}, k_1, k_2, \mathcal{R}_i, \mathbf{t}_i, \mathbf{M}_j)||^2 \tag{A.19}$$

[1] Image courtesy: https://docs.opencv.org/2.4.13.2/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

where $\breve{\mathbf{m}}(\mathbf{A}, k_1, k_2, \mathcal{R}_i, \mathbf{t}_i, \mathbf{M}_j)$ is the projection of the point $\mathbf{M}_j$ in image j according to equation A.3, followed by distortion according to equations A.15 and A.16. This is a nonlinear minimization problem, which is solved by the Levenberg-Marquardt algorithm. This requires an initial guess for $\mathbf{A}$ and $\mathcal{R}_i, \mathbf{t}_i,\ i = 1 \dots n$ which can be obtained by solving equation A.11. k_1 and k_2 can be estimated by solving A.18, or simply set to 0 initially [47]. Typically, the rotation matrix $\mathcal{R}$ returned from this procedure will not be a true rotation matrix (where the rows and columns are orthogonal unit vectors) due to image noise. A method for estimating the best rotation matrix from a general 3×3 matrix is given in [47].

kilde på dette?

A.1.4 Calibration Procedure

In Zhang's paper, the recommended calibration procedure is summarized as [45]:

1. Print a calibration pattern and attach it to a planar surface
2. Take a few images of the model plane under different orientations by moving either the plane or the camera
3. Detect the feature points in the images
4. Estimate the five intrinsic parameters and all the extrinsic parameters using the solution of equation A.11
5. Estimate the coefficients of the radial distortion by solving equation A.18
6. Refine all parameters by minimizing equation A.19

Bibliography

[1] B. Siciliano and O. Khatib, *Springer Handbook of Robotics*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2007, ISBN: 354023957X.

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

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

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

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

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

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

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

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

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

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

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

[13] A. Steinberg and C. Bowman, "Revisions to the jdl data fusion model," in *Handbook of Multisensor Data Fusion - Theory and Practice*, M. E. Liggins, D. L. Hall, and J. Llinas, Eds., CRC Press, 2008, ch. 3, pp. 45–66.

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

[15] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Pearson, 2008.

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

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

[18] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. arXiv: 1311.2901. [Online]. Available: http://arxiv.org/abs/1311.2901.

[19] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012. arXiv: 1207.0580. [Online]. Available: http://arxiv.org/abs/1207.0580.

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

[21] M. Quigley, B. Gerkey, and W. D. Smart, *Programming Robots with ROS*. O'Reilly, 2015.

[22] *The 3-clause bsd license*, https://opensource.org/licenses/BSD-3-Clause, Accessed: 2017-12-06.

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

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

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

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

[27] P. McManamon, *Field Guide to Lidar*. SPIE, 2015.

[28] I. Velodyne LiDAR, *User manual and programming guide, vlp-16*, English, version 63-9243 Rev B Mar 16, Velodyne LiDAR, Inc., 49 pp., March, 2016.

[29] C. L. Glennie, A. Kusari, and A. Facchin, "Calibration and stability analysis of the vlp-16 laser scanner," vol. XL-3/W4, Gottingen: Copernicus GmbH, 2016, pp. 55–60.

[30] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[31] E. J. Tangstad, "Visual detection of maritime vessels," Master Thesis, Norwegian University of Science and Technology.

[32] R. Girshick, "Fast r-cnn," in *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

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

[34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. arXiv: 1409.1556. [Online]. Available: http://arxiv.org/abs/1409.1556.

[35] *Faster r-cnn: Towards real-time object detection with region proposal networks*, https://github.com/ShaoqingRen/faster_rcnn, Accessed: 2017-12-15.

[36] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

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

[38] *Caffe for faster r-cnn*, https://github.com/ShaoqingRen/caffe/tree/faster-R-CNN, Accessed: 2017-12-17.

[39] O. S. R. Foundation, *Libuvc_camera - ros wiki*, http://wiki.ros.org/libuvc_camera?distro=kinetic, Accessed: 2017-12-06.

[40] *Ros device drivers*, https://github.com/ros-drivers, Accessed: 2017-12-06.

[41] ——, *Velodyne - ros wiki*, http://wiki.ros.org/velodyne?distro=kinetic, Accessed: 2017-12-06.

[42] *Rosbag - ros wiki*, http://wiki.ros.org/rosbag, Accessed: 2017-12-09.

[43] *Ros tutorials: Creating a ros msg and srv*, http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv, Accessed: 2017-12-18.

[44] MathWorks, *Estimate geometric parameters of a single camera - matlab*, https://se.mathworks.com/help/vision/ref/cameracalibrator-app.html, Accessed: 2017-12-08.

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

[46] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*. The MIT Press, 1993.

[47] Z. Zhang, “A Flexible New Technique for Camera Calibration,” Microsoft Corporation, Tech. Rep., Dec. 1998.

[48] J. Weng, P. Cohen, and M. Herniou, “Camera calibration with distortion models and accuracy evaluation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 965–980, 1992.

[49] G. Wei and S. De Ma, “Implicit and explicit camera calibration: Theory and experiments,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 5, pp. 469–480, 1994.

[50] D. C. Brown, “Close-range camera calibration,” *PHOTOGRAMMETRIC ENGINEERING*, vol. 37, no. 8, pp. 855–866, 1971.