Master's thesis

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

Øystein Volden

Vision-Based Positioning System for Auto-Docking of Unmanned Surface Vehicles (USVs)

Master's thesis in Cybernetics and Robotics

January 2020

NTNU
Norwegian University of Science and Technology

NTNU
Norwegian University of
Science and Technology

Master Thesis

Vision-Based Positioning System for Auto-Docking of Unmanned Surface Vehicles (USVs)

Øystein Volden

Submission date: January 20, 2020
Supervisor: Thor I.Fossen
Co-supervisor: Marco Leonardi

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Master Thesis Description

Introduction

This is a master thesis done in collaboration with the company Maritime Robotics (MR). The work is partly based on results from my project thesis.

Main objective

The main objective of the master thesis is to explore how stereo and monocular vision can be used as a supplementary positioning system in autodocking operations with computer vision and deep learning techniques. The camera system should deliver accurate localization estimates nearby the dockside, run real-time and be robust to outdoor conditions. Hopefully, the project will converge towards a supplementary positioning system for ship docking in near future and thereby add valuable insight for fully autonomous ships.

Tasks

- Literature study on autonomous systems, relevant sensors, and markers for localization tasks as well as deep learning and computer vision techniques for vision-based navigation applications.
- Prepare an embedded system for deep learning computations onboard a small USV.
- Connect a pair of industrial machine vision cameras to a camera driver for launching two high-quality camera streams in parallel. Also, enable hardware triggering between the left and right camera for reliable synchronization between the cameras.
- Record a dataset with markers in autodocking scenarios using a small, flexible USV and prepare the dataset for deep learning tasks.
- Train and validate a deep learning-based object detection model. Test the detector and evaluate its accuracy.
- Mount the cameras on a camera rig and perform stereo calibration.
- Extract a pair of bounding boxes from left and right camera, and perform stereo matching and triangulation for 3D reconstruction. Compare both monocular and stereo vision techniques.
- Merge sub-modules into a complete 3D pose estimation system in Robotic Operating System (ROS).
- Synchronize and compare camera measurements with ground truth lidar measurements. Use real auto-docking scenarios for the final experiments.
- Present some suggestions for future work based on the results and research.

Abstract

The focus on autonomy in the maritime industry has increased significantly the recent years. Autonomous vessels have the potential to reduce costs and improve safety significantly. However, there are several challenges to face before fully autonomous ships may enter the commercial market. One of them is autonomous docking.

This master thesis is a study of how a low-cost stereo vision system can be used as a redundant positioning system in docking operations. That is, when GPS signals are not available or redundant position estimates are desirable, autonomous vehicles can obtain navigation information with cameras mounted on the vehicle. A vision-based positioning system consisting of an object detector followed by different 3D reconstruction techniques are proposed. My main contribution is to extend and further develop a 3D localization pipeline based on relevant object detection frameworks, as well as designing a stereo camera rig from scratch. To achieve a low-latency pipeline with accurate and frequently localization estimates from a camera system, it is necessary to have a fast and precise detection model available. Furthermore, to face the challenge of robust detections concerning outdoor conditions, a learning-based object detection model is proposed since it can handle variations in the scene affected by environmental changes as long as such examples are widely represented in the dataset. The key idea is to utilize data-driven methods in outdoor environments where classical computer vision techniques may fail. That is, the final working system employs a complementary part-based approach that uses a combination of data-driven deep learning methods, utilizing data wherever required, and at the same time use traditional computer vision techniques when the scope and complexity of the task is reduced. Both monocular and stereo vision techniques will be investigated for comparison. To simplify the task of locating the camera relative to its surroundings, easily identifiable markers (with assumed known geocentric coordinates) are used to obtain reference points. A small USV with lidar signals available is used to verify that the vision-based positioning system produces accurate localization estimates under various docking scenarios. Experiments prove that the developed system works well and can supplement the traditional navigation system for safety-critical docking operations.

Keywords: Safety-Critical Operations, Autonomous Docking, Vision-Based Navigation, Convolutional Neural Network, Object Detection, 3D Reconstruction, Computer Vision

Sammendrag

Fokuset på autonomi i den maritime industrien har økt betraktelig de siste årene. Autonome fartøy har potensiale til å redusere kostnader og øke sikkerheten betraktelig. Men det er flere utfodringer som må løses før helt autonome skip kan brukes i det kommersielle markedet. En av dem er autonom dokking.

Denne masteroppgaven er en studie om hvordan et lavkost stereo kamerasystem kan bli brukt som et redundant posisjonsystem i dokkingoperasjoner. Det vil si, når GPS signaler ikke er tilgjengelige eller redundante posisjonsestimat er ønskelig, kan autonome fartøy skaffe navigasjonsinformasjon med kameraer montert på fartøyet. Et kamerabasert posisjonssystem bestående av en objektdetektor oppfulgt av forskjellige 3D rekonstruksjonsteknikker er foreslått. Mitt hovedbidrag er å utvide og videreutvikle et 3D lokaliseringssystem basert på relevante objektdeteksjonsrammeverk, samt designe en kamerarigg fra bunnen av. For å oppnå lav forsinkelse i systemet med nøyaktige og hyppige lokaliseringsestimat fra et kamerasystem, er en rask og presis deteksjonsmodell nødvendig. Videre, for å takle utfodringen med robuste deteksjoner under utendørsforhold, er en læringsbasert objektdeteksjonsmodell foreslått siden den kan takle variasjon i omgivelsene påvirket av miljøendringer, så lenge slike eksempler er bredt representert i datasettet. Nøkkelideen er å utnytte datadrevne metoder i utendørsmiljø hvor klassiske datasyn-metoder kanskje feiler. Altså, det endelige fungerende systemet tar i bruk datadrevne dyp læringsmetoder hvor det er nødvendig, og samtidig anvender tradisjonelle datasyn-metoder hvor omfanget og kompleksiteten av oppgaven er redusert. Både monokulært og stereokamera vil bli utforsket for sammenligning. For å forenkle oppgaven med å lokalisere kameraet relativt til sine omgivelser, er enkelt identifiserbare markører (med antatt kjente globale koordinater) brukt til å finne referansepunkt. En liten USV med lidarsignal tilgjengelig er brukt for å verifisere at det kamerabaserte posisjonssystemet produserer nøyaktige lokaliseringsestimat under ulike dokkingoperasjoner. Eksperimenter beviser at det utviklede systemet fungerer bra og kan supplementere det tradisjonelle navigasjonssystemet for sikkerhetskritiske dokkingoperasjoner.

Nøkkelord: Sikkerhetskritiske Operasjoner, Autononom Dokking, Kamerabasert Navigasjon, Foldings Nevrale Nettverk, Objekt Deteksjon, 3D Rekonstruksjon, Datasyn

Preface

This master thesis is carried out in the Department of Engineering Cybernetics, at NTNU in Trondheim, the fall of 2019. It is submitted as a requirement for the master thesis.

First, I want to thank my supervisor, Professor Thor I.Fossen, for valuable guidance throughout every stage of this thesis as well as funding for the equipment. In the same regard, this work would not be possible without help from my co-supervisor Marco Leonardi which has given great support and supervising, both on theoretical and practical aspects related to my thesis. I would also like to thank Terje Haugen and the rest of the staff at the Mechanical lab at Dept. of Eng. Cybernetics for helping me with designing the camera rig and housing the embedded computing system.

I am also very grateful for Maritime Robotics who letting me use one of their USVs for testing and experiments related to the work. They have been especially helpful and open-minded when interfacing my implementation into their USVs for outdoor experiments. Wherever practical issues occurred, they were always happy to help. This has without any doubt accelerated the progress of my work.

This thesis is a continuation of the authors' specialization project during the spring of 2019. At an early stage, it was decided to succeed parts of the final (master thesis) implementation in the specialization project. Here, a data-driven object detection model was investigated, to achieve robust, fast and accurate detections in various outdoor conditions. By this, the master thesis could focus on other aspects to accelerate the progress towards full-scale experiments. Regarding equipment and software used throughout the thesis, we refer to Chapter 5, which gives an overview of the hardware and software used and outlines some reasons why they were chosen.

At last, I would like to thank my family for their great support during my master thesis. They have encouraged me along the way, also on bad days where things looked darker.

Øystein Volden
Trondheim, January 2020

Contents

List of Tables

List of Figures

Abbreviations

AI	=	Artificial Intelligence
AP	=	Average Precision
AUC	=	Area Under Curve
CCD	=	Charge-coupled Device
COCO	=	Common Objects in Contexts
CMOS	=	Complementary Metal Oxide Semiconductor
CNN	=	Convolutional Neural Network
CUDA	=	Compute Unified Device Architecture
DOF	=	Degrees Of Freedom
DP	=	Dynamical Positioning
ECEF	=	Earth-Centered Earth-Fixed
EO	=	Electro Optical
FC	=	Fully Connected
FPN	=	Feature Pyramid Network
FPS	=	Frames Per Second
FOV	=	Field Of View
GNSS	=	Global Navigation Satellite Systems
GPIO	=	General Purpose Input Output
GPS	=	Global Positioning System
GPU	=	Graphical Processing Unit
IOU	=	Intersection Over Union
IR	=	Infrared
LIDAR	=	Light Detection and Ranging
OBS	=	On-Board System
POE	=	Power Over Ethernet
PR	=	Precision Recall
RANSAC	=	Random Sample Consensus
R-CNN	=	Region-based Convolutional Neural Network
ReLU	=	Rectified Linear Unit
ROI	=	Region of Interest
ROS	=	Robot Operating System
RTK	=	Real-Time Kinematic
SA	=	Situational Awareness
SGD	=	Stochastic Gradient Descent
SPP	=	Spatial Pyramid Pooling
TOF	=	Time-of-Flight
UDP	=	User Datagram Protocol
USV	=	Unmanned Surface Vehicle
YOLO	=	You Only Look Once

Chapter 1

Introduction

This is a master thesis done in collaboration with Maritime Robotics (MR), a Trondheim-based company working on unmanned solutions in the air and on the surface. This chapter is mainly based on my project thesis, with some adaptions to fit into this report. It is a case study on how a stereo camera setup can be used as a redundant positioning system for auto-docking operations with computer vision and deep learning methods. More specifically, the goal is to measure the relative distance and orientation between the quay and the camera system on-board an Unmanned Surface Vehicle (USV) with high accuracy in real-time. In addition to the cameras, several sensors such as Global Navigation Satellite System (GNSS) and radar already exist on most ships today, which deliver position measurements. In that sense, the camera system produces redundant measurements. However, the need for redundancy (e.g. duplication of critical measurements) increases as an autonomous ship needs to be extremely reliable and operate well at all times. This is crucial in order to succeed business of autonomous ships and maintain trust among investors, the crew, passengers and public opinion.

GNSS is used as the main positioning system onboard most ships today. The technology is well-established and GNSS onboard ships often holds a high standard. However, in docking operations without a human operator presented, the need for position estimates with centimeter precision is desirable, and the GNSS may fail to produce such high precision estimates. The consequences of ignoring this problem can be dramatical: The vehicle can hit the quay with a great amount of force, expensive damages may occur and the safety for passengers is in danger. Therefore, it is proposed to apply a local vision-based navigation system during the last part of the docking phase and optimize the position of the ship based on both systems. The key idea is to have a low-cost backup positioning system specialized for auto-docking environments that can prevent the ship from unfortunate maneuvers based on inaccurate or lack of GNSS measurements.

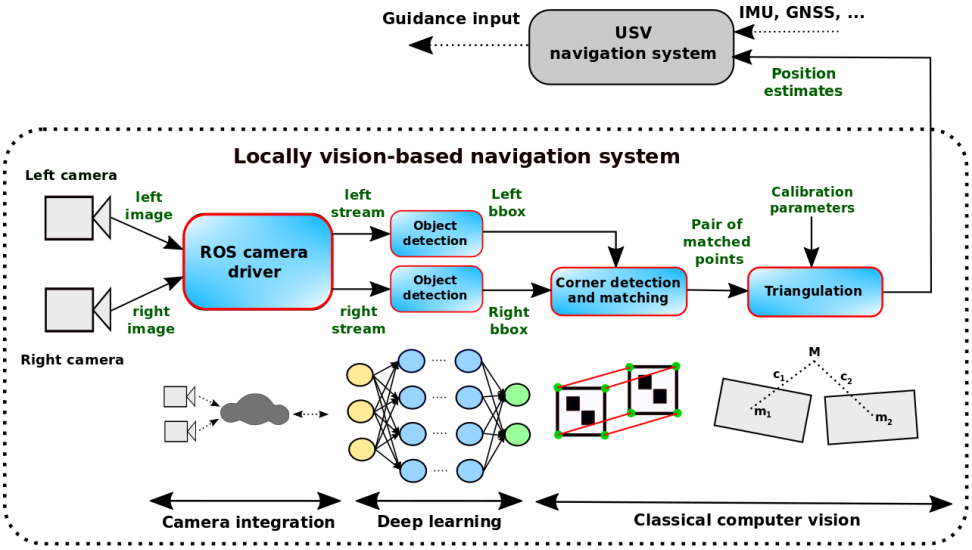


Figure 1.1: The proposed vision-based positioning system for the overall project. The main intention is to feed the USV navigation system with accurate position measurements when the USV approach the dockside.

1.1 Motivation

The main motivation behind the project is to explore how computer vision and deep learning-based methods can be used to develop a new positioning system. The idea is to place cameras on a portable and flexible test platform such as the *Otter* USV (see Figure 1.2). This allows for flexible choices of docking environments as well as the opportunity for testing and recording of data at different outdoor conditions. The *Otter* USV also contains a RTK GPS with centimeter precision which can be used as ground truth to evaluate position estimates from the camera system. Further, it is proposed to place the cameras in a pair to achieve stereo vision view towards the dockside (see Figure 2.2). Figure 1.1 illustrates (on the high level) how the vision-based navigation system is intended to work and how the position estimates may be used by the USV together with other navigation measurements. For now, the intention is that the reader gets some context of the "end-product". Further details around this system will be reviewed in Chapter 7.

On the hardware side, the focus will be on cameras and lenses suited for auto-docking operations. For instance, wide-angle lenses are used to cover much of the quay during the docking operation. This allows for a small number of cameras installed. As the focus is not on existing sensors on older ships (like ferries), it is hard to compare directly how the cameras mounted on a small USV performs in comparison. For instance, the camera views and baseline are completely different. In addition, one can expect the USV to oscillate more at sea due to a less stable construction, compared to a ferry or a cargo ship. However, there are reasons to believe that many of the ideas and principles from the proposed vision-based position system can be transferred to bigger ships.

Figure 1.2: The Otter is a small and portable Unmanned Surface Vehicle (USV) suited for mapping and monitoring of its surroundings at sea. Image courtesy by Maritime Robotics [1].

The development of autonomous ships have proceeded fast and in 2016, Kongsberg Maritime (earlier Rolls-Royce Marine) sold their first auto-crossing system to Fjord1 [2]. Auto-crossing is a big step towards fully autonomous ships as it controls the ship autonomously between the quays, and the captain only takes the control when it's docking (and undocking) the quay. The system is also optimized to be energy-efficient and new fully electric ferries like MF *Gloppefjord* take great advantages of this, as it reduces charging time at the quay. However, to complete the autonomous pipeline from quay to quay, the auto-crossing system needs to be extended and include the ability for autonomous docking. Lately, the first public version of autonomous docking has been demonstrated in a fully autonomous ferry transit in Finland by KM in December 2018 [3]. The demonstration was done on the ferry *Falco* by Finferries as shown in Figure 1.3.

One may question the motivation behind a thesis about autonomous docking when it is already demonstrated. The fact is however that some challenges remain before a robust autonomous docking system can fully enter the commercial market.

One challenge is the strict requirements for redundancy. Unexpected events such as loss of signal/error in the GNSS measurements due to satellite signal blockage can occur and put the ship in a dangerous position. Other GPS limitations include multipath effects, interference, jamming and occasional high noise content [4]. For instance, the multipath phenomenon is a well-known challenge in the Norwegian fjords, where satellite signals tend to reflect via the mountainsides and confuse the GNSS receiver as it receives multiple signals from the same source (i.e. the satellite) [5]. Autonomous cars also suffer from related problems: Self-driving cars in the city can lose GNSS signals due to satellite signal blockage from the buildings. And obviously, indoor applications suffer as well.

High-speed applications (e.g. a car) with the lack of position estimates and the absence of a human operator can be an extremely dangerous combination. Therefore, it has been done extensive research on how sensors like camera and lidar can benefit from their strengths, both standalone and in a sensor fusion scheme. The goal is to use such sensors in a local navigation system to prevent vehicles from dangerous situations where life may

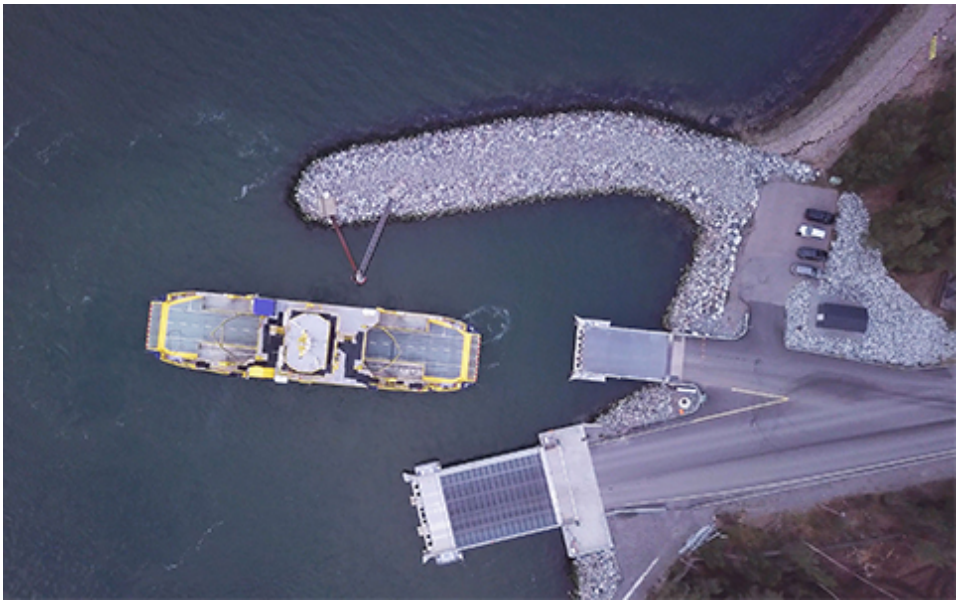


Figure 1.3: The ferry *Falco* during the world's first fully autonomous ferry transit December 2018. Image courtesy of Teknisk Ukeblad [3].

get lost. Although the auto-docking operation is not considered as a high-speed application, the redundancy problem is still serious for ships in general. A completely redundant positioning system is therefore desirable. Research on autonomous cars aims to solve similar problems, which can be beneficial for solving problems relating to autonomous ships. There are several relevant sensors that may be used and they are all reviewed in Chapter 2.

Another challenge is cybersecurity threats. Navigation and sensory systems are vulnerable for several cyber-physical attacks such as jamming, spoofing, and bitstream manipulation. Lately, Russia is accused of jamming the GPS signals in the air-craft traffic in the northern part of Norway and Finland. The signals from the satellites are considered weak and can easily be manipulated by sending signal noise from the ground and thereby jamming the GPS signals [6]. Although air-crafts have several navigation systems to use in case one fails, this is considered to be a serious security risk in general. If jamming signals reach the ground, it can cause problems for a lot of vehicles including ships executing risky navigation maneuvers. This motivates the need for other systems than GNSS to address this issue. In addition, sensor measurements should be encrypted to increase the resilience of autonomous vehicles.

The economical aspect is also important, especially for older ships. New installations on older vessels may not be beneficial. For a shipowner, it is desirable to use the equipment already installed and may reduce it to software installation. New ships however should be built with sensors on a flexible infrastructure that enables and simplifies the possibility for auto-docking (if not already installed). In addition, it is a significant cost for shipowners to

subscribe for differential corrections on GNSS signals (i.e. decimeter precision) or Real-Time Kinematic (RTK) GPS (e.g. centimeter precision). Research on sensor fusion may change this in the future, and work on a vision-based localization system is definitely an important part of this.

1.2 Contributions

This project explores different methods that are needed for estimating the position of an USV during docking operations. In the following, the main contributions will be described in bullet points, starting with the project thesis work followed up by contributions during the master thesis.

- **Project thesis**: In the project thesis, which this master thesis is partly based on, a lot of attention was given to the object detection model. A data-driven method was investigated to achieve robust, fast and accurate detections in various outdoor conditions. To achieve this, a lot of hours were used to collect and prepare the data. This includes evaluating data relevant for the detection tasks and label ground truth examples for the learning tasks. Further, the deep learning model was trained, validated and tested. Look into Section 7.3 for more information about these procedures. The model produced quite promising results which can be seen here:

  `https://youtu.be/8-3frymRwdk`

- **Camera rig design**: The master thesis includes a lot more practical work as there are many components to consider, with respect to hardware and software, for the final perception pipeline to work reliably on an embedded system in real-time onboard an USV. This includes a selection of suitable equipment for a camera rig. The camera rig needs to be waterproofed for outdoor use and fit the test platform (e.g the USV). Also, the embedded computing system should be housed to be protected against seawater during experiments. For more information about the hardware setup, see chapter 5.

- **Camera integration**: Further, many working hours are related to the stereo vision system, a GigE PoE camera pair (including wide-angle lenses). Camera integration into the perception pipeline was time-consuming. Some aspects are exploring the camera driver, bandwidth and network requirements to maintain stable camera streams, and tuning camera parameters for final configuration. Further details are described in Section 7.2.

- **Software development and experimental testing**: Also, a lot of time was spent on software development related to pose estimation algorithms. The deep learning model YOLOv3 was integrated into ROS (Robot Operating System), mainly by using the git repository

`https://github.com/pushkalkatara/darknet_ros`

as a starting point. This is a flexible and general framework for running YOLOv3 in ROS. ROS is a flexible framework for writing robot software and supports many machine vision cameras which offer bridging between OpenCV and ROS. My contribution is to extend and further develop the code for 3D localization tasks. That is, how we can extract 3D information from bounding box predictions. More specific, both monocular and stereo vision techniques for pose estimation are developed, tested extensively and compared. See Section 7.4 for more details.

- **Literature study**: In addition to the practical work, a considerable literature study was done within the field of computer vision and deep learning as well as related works for autonomous positioning systems in the maritime industry. The aim is to reflect the motivation and theory behind the methods used in the thesis.

1.3 Structure

The master thesis has now been introduced. Several challenges related to fully autonomous ships are discussed with a special emphasis on auto-docking. In the following, the structure of the thesis will be presented.

- **Chapter 2** presents literature references to related work within autonomous systems, positioning principles and relevant sensors and markers for localization tasks.
- **Chapter 3** introduces theory within the field of deep learning related to the implementation. It covers theory related to the structure of a convolutional neural network as well as how it can be trained and evaluated. In addition, different state-of-the-art object detection models will be reviewed and compared. At last, the chosen model for the implementation, YOLOv3, will be given some extra attention.
- **Chapter 4** introduces theory about traditional computer vision techniques related to the implementation. Basic concepts to perform 3D reconstruction from an image will be presented. That is, how 3D position and orientation measurements can be obtained from single-view and stereo-view geometry. A short introduction to cameras is also given.
- **Chapter 5** reviews the choice of software and hardware components for the master thesis. We will especially focus on the hardware design to build the vision pipeline and outline some reasons why the different components were chosen.
- **Chapter 6** focuses on the various data sets used for a data-driven learning method with special emphasis on the construction of a dataset designed for docking operations. That is, a custom dataset used to fine-tune the CNN for auto-docking operations.

- **Chapter 7** focuses on the implementation of the camera system. It is separated into three modules. First, how the cameras is integrated into the perception pipeline. Then the object detection model is reviewed. That is, how the datasets are prepared and how the model can be trained, validated and tested within the framework. At last, the 3D reconstruction pipeline will be reviewed. E.g. how we can extract 3D information from bounding box predictions.
- **Chapter 8** contains details about how the experiments are prepared. This includes for instance which assumptions we did on the way to achieve meaningful results. E.g. which assumptions that were necessary to compare how well the camera system performs compared against a ground truth lidar sensor. It also describes specific details about each experiment. Hopefully, this gives the reader a better overview of the objective of each experiment.
- **Chapter 9** introduces the results followed up by a discussion for each experiment.
- **Chapter 10** concludes the thesis. That is, summarizing the most important findings and presenting some future challenges that must be faced. In addition, an overview of the core of the thesis is given.

Chapter 2

Related Work

This Chapter presents literature references to related work within autonomous systems, position principles, and sensors and markers for localization. This chapter is mainly based on my own project thesis.

2.1 Autonomous Systems

We start this Chapter by mentioning a couple of autonomous systems and their history, in order to get some context within the field of autonomy. Systems that can change their behavior in response to unanticipated events during operation are called "autonomous" [7]. The *Johns Hopkins Beast* is considered the world's first autonomous system purely based on feedback control, a theory that was formed in the field of cybernetics. In the 1960s, the mobile automation drove around in the hallway, and when the batteries ran low it was able to search for a black socket and plug itself in to charge, all on its own. The robot did not use a computer and was purely based on transistors controlling analog voltages, and it used sonar and photocell optics to navigate itself [7].

Since the 1960s, the development of autonomous systems has proceeded a long way. Today, autonomy within the car industry has received a lot of attention. Several companies like Tesla and Uber among many others are working hard on autonomous solutions for the commercial market. However, the strict security requirements make it challenging to turn vision into reality and the skepticism among people will not disappear as long as accidents related to autonomous vehicles still occur.

The last years, autonomy has also been given more attention by the maritime industry, and companies like Kongsberg Maritime, Volvo Penta, Wärtsila, DNV GL, and Maritime Robotics are all working on solutions for autonomous ships. Kongsberg Maritime is, in collaboration with DNV-GL and Yara, working on the autonomous cargo ship *Yara Birkeland*. It is estimated to be launched by the first quarter of 2020 and gradually move from manned operations to fully autonomous operations by 2022 [8].

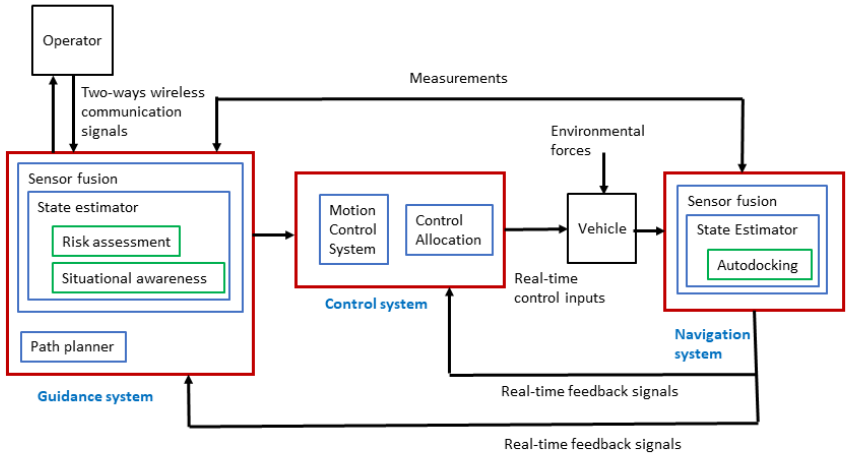


Figure 2.1: A classical guidance, navigation and control system with auto-docking proposed as a local navigation system nearby the quay. Other autonomous applications, like situational awareness and risk assessment, can be used for human-decision support as shown the guidance block. Unlike situational awareness and risk assessment, auto-docking is intended to work more independently of a human operator.

2.1.1 Applications for autonomy in the maritime industry

The rise of autonomous ships opens up for several new applications using different sensors, most likely in a sensor fusion system. Among these, you find applications such as situational awareness, risk assessment, and auto-docking and the use of each application is shown in a control system perspective in Figure 2.1.

Auto-docking

The travel of a vessel from quay to quay normally consists of three different modes: un-docking, transit and docking. Since the un-docking is more or less the inverse operation of docking, one can reduce the problem into two modes, namely docking and transit. As mentioned in Chapter 1, Rolls-Royce Marine has already delivered its auto-crossing product and thereby making the transit mode autonomous. One of the most critical parts of the transit of a maritime vessel is the docking operation. It is the last operation of the transit when a vessel is slowly approaching and then connects to the quay. Then, to maintain its static position, the vessel thrust against the dockside and connects a rope or some other robust connections between the vessel and the quay. In general, the ship should connect to the quay using fine-tuned maneuvers. Figure 2.2 illustrates the docking area where the ship may need several sensors with high precision to achieve a safe docking without human interruptions. How the docking operation is performed in detail will differ

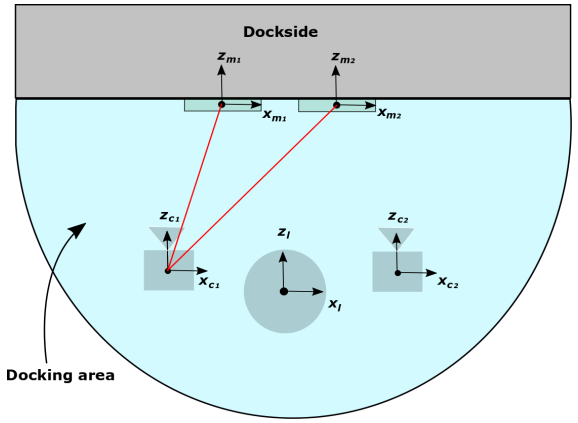


Figure 2.2: From a top-down view, the dockside is illustrated together with sensors on-board a ship to localize itself during the docking operation. The blue area illustrates a constrained area where the proposed vision-based positioning system is intended to work. That is, the area where the ship requires accurate localization measurements from several sensors (e.g. redundancy) enabling a safe, autonomous docking. The markers are used to obtain the relative localization between the ship and the docking station (e.g. the red lines between the markers and the left camera).

from vessel to vessel. For instance, a small flexible USV will have many more ways to dock compared to a ferry or a cruise ship. Also, the consequences of a docking failure will be severe for a large scale ship as it may cause damage on passengers, the dockside or the ship itself, while a small USV may only inflict small damages on itself. Aside from the consequences, the main objective for the vessel (independent of the size) is to be maneuvered in a slowly, energy-efficient and precise manner. And such operations may be challenging, even for a trained crew.

The strict requirements for precision and redundancy in an autonomous docking make it challenging to develop. As already mentioned, a lot of companies are working on solutions for automatic docking systems these days and they are choosing different approaches to solve it. Volvo Penta's system is heavily dependent on sensors mounted on the quay, and can therefore not be used unless the quay is configured for automatic docking [9]. Wärtsila develops an auto-docking system which, in contrast to Volvo Penta's system, relies heavily on the ship's dynamical positioning (DP) system based on GNSS and inertial measurement units (IMU) to avoid drift in the position estimates [10]. This also makes it very dependent of satellites signals. Both systems are just prototypes and are not ready for commercial use yet. It is also worth to mention that Volvo Penta focusing on auto-docking for private yachts and the size of a yacht makes it more comparable with an *Otter* USV, which is the intended test platform for this thesis.

Situational awareness

Situational Awareness (SA) is crucial for maritime operations where the goal is to identify dangerous threats as soon as possible to maintain safe operations [11]. Autonomous ships must be able to handle a lot of complex situations and in order to do so, the ship needs

to be aware of the situation in the first place. Different sensors can be used to monitor the surroundings and detect possible threats. For instance, cameras are already installed at most ships today and are typically used for manually monitoring by the crew. However, this can be quite unilaterally in the long run. Most of the time, no unexpected events occur. But this "safe picture" can change in a moment. Therefore, it is desirable to have a system that detects threats early, and reports about it to the crew immediately. The rise of machine learning algorithms makes it possible to detect and classify various threats in real-time. It doesn't necessarily need to handle the final navigation decisions on its own. Instead, it can be used for human decision-support to give the crew better insight in difficult and unexpected situations that must be handled quickly.

Risk assessment

Risk assessment involves analyzing what can go wrong, how likely it is to happen, what potential consequences are and how tolerable identifiable risks are [12]. To be able to perform a risk analysis, one needs sensors that can gather information about each individual risk and visualize them in an intuitive way. The risk assessment may be complex and consists of many individual risks. Depending on the level of autonomy, there are several approaches to how autonomously risk assessment can be. One approach is to gather and visualize raw sensor data and let the human operator handle a lot of information about every single risk for decision-making. On the other end, one may let the Artificial Intelligence (AI) algorithms process the sensor data and make decisions on its own (e.g. end-to-end solution). Both approaches obviously have weaknesses. Bringing too much complex information to a human operator will lead to human mistakes. On the other hand, fully autonomously solutions (i.e. neural networks) may not be desirable. Even with excellent performance, a black-box approach is not suitable for applications with safety-critical and/or economical-impactful issues. The deep layer structure makes it hard to achieve trustworthy and understandable predictions as the algorithms have no "consciousness" and thereby no answer on how they arrive at the final decision. As with SA, there is a solution in between using a human-machine interface. The algorithms detect each risk, visualize and make an intuitive overview of them, but leave the final decision to the operator. With this approach, the burden of heavy manual monitoring is gone, while it is ensured that the final (critical) decisions are based on someone that can defend and interpret for their choices (e.g. are reliable).

Now, some background on autonomous systems is presented. In addition, several hot applications within the maritime industry are presented. One of them is auto-docking. With auto-docking in mind, we will further introduce and explore the principles of positioning in order to develop a new localization system.

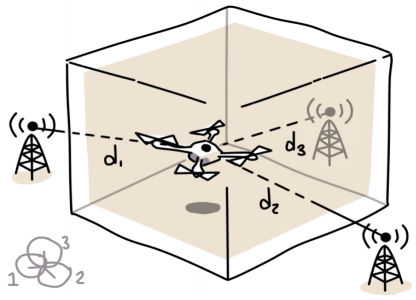


Figure 2.3: Trilateration principle in three dimensions. Image courtesy of [13]

2.2 Positioning

Positioning is the process of determining and describing the position of an object with respect to a coordinate system. When navigating, it is common to use an "Earth-Centered Earth-Fixed" (ECEF) coordinate system that rotates with the earth. The position in earth coordinates is given in latitude, longitude and height. Further, a positioning system should be able to estimate the location of some object based on sensor measurements. In this context, the principles of trilateration and triangulation are very central and are widely used by satellites and GNSS systems today. Trilateration involves measuring distances between the robot itself and objects of interest around it. The spheres (with measured radius) around these objects will intersect in one or several points. With at least three such points, it is possible to localize the robot itself with respect to these objects as shown in Figure 2.3. If distance measurements are not available, it is possible to localize the robot with direction measurements (i.e. heading). If the direction from a robot to an object is known, the object could be anywhere on this line [13]. Therefore, a second direction measurement to another object is needed to triangulate and find the robot.

In this project, it is assumed that the absolute position (ECEF) of some static marker at the quay is known. Thus, the focus relies on estimating the relative position and orientation between the marker and the camera. Since the cameras are mounted to the rig, they are assumed to be fixed with respect to the USV and one can perform simple coordinate transformations to obtain the relative pose of some other point of interest at the USV (i.e. the front of the boat).

2.3 Sensors

To make a positioning system, one needs to know which sensors to use. Some weaknesses of GNSS are discovered in Chapter 1, and it is proposed to find alternative sensors completely redundant to the GNSS system. Optimally, an autonomous navigation system should cover all scenarios without the intervention of a human operator. Therefore, the

ultimate navigation system may use sensor fusion to benefit from the strengths of each sensor and provide robustly and reliable measurements compared to what one sensor can provide alone. For this thesis, the focus is limited to auto-docking (or places nearby land) with visual landmarks. With this application in mind, several sensors have been considered and compared against each other. We will now present some sensors both already in use and some good alternatives that could be installed.

Electro Optical Camera

An Electro-Optical (EO) camera is a passive sensor that can record images of the scene in front of it. Optical cameras are passive as they measure the reflected light emitted from the sun. They provide a very defined way of determining the resolution. By counting pixels, often in vertical and horizontal (i.e. 1920 x 1200, 1280 x 720, etc), one can measure the resolution with a high level of certainty. Furthermore, images of the scene can be combined with geometrical calculations and prior information of some object to estimate the relative pose between the camera and the object in the scene. How it is obtained, depends on the number of cameras in use (i.e. monocular or stereo vision). By using a feature detector, one can extract geometrical information from detected features of interest in the scene. Both classical feature detectors and learning-based approaches (i.e. deep neural networks) can be used to achieve this.

Images provide rich and meaningful information compared to many other sensors and can be used for a lot of applications such as object recognition and localization tasks. It has been done extensive research on visual-based localization systems lately, especially for indoor and outdoor vehicle navigation such as drone inspection, autonomous cars and now autonomous ships. This may be due to the fact that visual-based positioning systems are considered more robust and reliable compared to other sensor-based localization systems [4]. In addition, cameras are considered inexpensive compared to laser sensors and GPS and provide high localization accuracy as well.

However, there are some drawbacks. Vision-based algorithms are highly sensitive to environmental conditions such as light conditions, illumination changes, shadows, motion blur, textures and presence of heavy rain, snow and fog [4]. Therefore, one may expect that such algorithms do not perform well under certain outdoor conditions. This could, for instance, be an auto-docking operation. One approach to deal with this problem is to use active markers like red fog-light (which already exists on many docksides) or similar. Today they are used by the captain to navigate in bad weather, but they can obviously be used for visual-based positioning systems as well. Another approach is to widely represent examples of different environmental conditions in the dataset, specially those occurring under real outdoor operations. With such a dataset, a machine learning approach could potentially learn how to recognize objects in different contexts, even under extreme outdoor conditions. Another drawback is the great amount of memory an image contains, which makes image processing generally a computationally expensive task. For visual-based localization tasks, computations often involve several steps like the acquisition of the camera images, extraction of detected features, matching features between frames and calculate position via pixel displacement between frames.

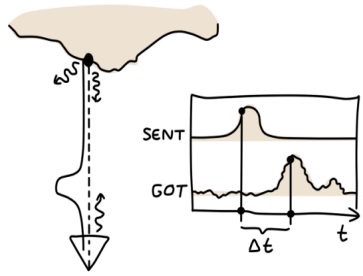


Figure 2.4: Time-of-flight principle. Image courtesy of [13].

IR Camera

While optical cameras provide images based on light reflecting off the object, infrared cameras measure heat energy emitted by the object [14]. They perceive the light of wavelengths, both inside and outside the visible spectrum [13]. That means they cannot provide real-world colors the same way optical cameras do. Instead, they can be used to identify colors outside the visible spectrum. Since infrared cameras measure heat transmitted from objects, one can use fiducial markers to produce colors corresponding to temperatures higher or lower than its surroundings. This makes it easier to create a marker that stands out from the scene and can be detected in the dark. Of course, one can also use an external light source to light up the marker to reduce the problem of darkness. Just like optical cameras, infrared cameras are passive as they do not send any signals out.

Another interesting aspect is the way IR cameras measure resolution. While optical cameras count pixels to measure resolution, infrared cameras usually follow the Johnson Criteria which estimates the number of line pairs across a target. However, there are some problems with this criterion. For example, thermal cameras can sometimes detect objects at a further distance than optical cameras because a hot object emits a lot of heat energy relative to its surroundings. However, if the hot object is cooled down, it may be impossible to detect the same small object far away [14]. Since optical cameras have a more defined way of determining what details we can see, it is hard to compare resolution directly.

Lidar (Light detection and ranging)

Another promising sensor used for many localization tasks is lidar. It is especially popular in the development of fully autonomous cars nowadays (except Tesla and particularly Elon Musk [15]) and is widely used for object detection, obstacle avoidance and 3D mapping. It is a laser-based technology that uses Time-Of-Flight (TOF) to measure the distance to surrounding objects. In TOF systems, a short laser pulse is sent out and the time until it returns is measured (see Figure 2.4). In addition, some lidars spin a beam in a circle, emit a pulse at regular intervals and measure how long it takes to return. Hence, it returns a

360 degree point cloud containing information about objects it hits nearby. Because lidars use a fine laser-beam, they can estimate distance with high-resolution [13]. And they can target a wide range of materials, but also remove some of the interest. For instance, one can apply filtering methods to denoise even thick snow [16]. These capabilities make the lidar very robust to different outdoor conditions.

In comparison with cameras, lidars provide a much wider Field of View (FOV) as well as a greater range. This enables possibilities for measuring longer distances. One can, of course, discuss how important longer distances are if only the last couple of meters are critical for the docking operation.

One drawback of lidar is the price, especially for high-resolution versions. However, the increasing use of it, especially in the automotive industry, pushes the price down every year and it is expected to be less expensive in the near future [17]. Moreover, utilizing lidar data for real-time applications may be challenging due to a high computational cost caused by the high-resolution point cloud.

Another aspect is the placement of a lidar. For a small USV, one may only need one single lidar on top of it to map its surroundings. For bigger ships, like a ferry, one has to install multiple lidars at appropriate positions to cover the entire area around the vessel, including the radar dead zone. It is also worth to mention that lidars, in comparison to cameras, are not usually installed at older ships. And installation of lidars involves certification for maritime use which makes them a bit more expensive than regular lidars. However, this drawback is not emphasized so much as the main focus of this project is to explore new innovative methods that can be used for future ships and not rely too much on what sensors are installed already.

Radar (radio detection and ranging)

The radar is widely used in the maritime industry. It is an active sensor that sends out radio waves in all directions by rotating 360 degrees and receives reflected signals in return, and map its area around based on this. Indeed, radar is the lidar of radio waves.

The radar is considered very robust to disturbances and varying environmental conditions, which makes it ideal for outdoor applications (most ships have radar installed today). Now, given a threshold depending on the magnitude of returned radio waves, it is possible to filter out objects of interest such as snowfall, leaves and birds. Furthermore, radio waves have the ability to more easily penetrate materials, and this makes radar a better choice (than a lidar) in the presence of fog/smoke [13].

Until now, radar sounds like the perfect sensor for outdoor localization. As with the other sensors, it also has weaknesses. Radio waves normally reflect off solid surfaces meaning that objects located behind such surfaces will not be detected by the radar. The area that cannot be seen by the radar is called the dead zone. One approach to deal with this problem is to install antennas (radar reflectors) near the docking in height such that the radar can detect them (i.e. out of the dead zone). This requires, of course, more installation on the quay as well as solid constructions to avoid oscillations due to windy conditions. In addition, the angular precision is lower than laser-based methods like lidar. This is due to

the fact that radio waves cannot make as narrow beams as lasers [13].

Comparison

As seen, all sensors have benefits and drawbacks. The objective of the project is to make a simple, low-cost positioning system that delivers accurate estimates and is robust with respect to outdoor applications such as autodocking. With this objective in mind, electro-optical cameras are a natural choice due to its price, flexibility, accuracy and amount of information stored in every image. However, optical cameras are highly sensitive to environmental changes and may be useless if the lens is covered by heavy rain or in extreme presence of fog and snow. In such scenarios, a laser-based method (i.e. lidar or radar) will most likely deliver more reliable information of its surroundings. No optimal sensor covering all scenarios exists today and a sensor fusion system is most likely needed to provide a better picture for extreme outdoor scenarios.

2.4 Markers

In order to locate the quay relative to the ship under autodocking operations, there should be some features on the quay for the computer vision system to detect. In this context, some kind of markers is proposed to detect such features (inside the markers) easily.

Fiducial markers

A lot of markers exist out there and among these, we find 2D barcode systems such as QR codes. A QR code can store hundreds of bytes (e.g. a lot of information) and works typically great with a stable, slow-varying camera capturing high-resolution images nearby. In contrast, a visual fiducial marker contains a small information payload (perhaps 12 bits) but is designed to be detected even at very low resolutions which allow for long-range detections. They provide camera-relative position and orientation of a tag, and are also designed to detect multiple markers in a single image. In addition, they are easy to recognize and distinguish from one another [18]. This allows for fast, accurate and robust detections, which is the first step of the pose estimation. Therefore, they are, not surprisingly, a popular choice for several pose estimation applications.

Several fiducials markers exist and among these, you find bar-codes, data matrix, QR codes and special 2D markers made for localization tasks such as ArUco marker, AR-Toolkit, ARTag and April Tag. Since this project focuses on outdoor localization, markers such as ArUco markers and April tags are especially interesting.

In addition, a clever choice of colors for the fiducial markers may help it to stand out from the scene, depending on how the background looks like. And obviously, a sufficient marker size must be chosen depending on the range from the marker the vehicle is supposed to localize itself. One approach to solving this may be to place a marker inside

another marker so that different markers can be detected at different ranges.
The markers should obviously be placed nearby the quay where the USV is supposed to dock. One configuration could be several pairs of ArUco markers at the waterfront with a fixed distance between each pair. A similar configuration is shown in Figure 2.5 where the robot operates at a close range with high localization precision.

Natural landmarks

In addition, it might be possible to use naturally-occurring landmarks. One benefit is that we use what is already in the scene. This could be buildings or some static features at the quay. Furthermore, it obviously increases the operational use as it allows for a much higher level of perception in undiscovered environments. However, it is generally harder to provide ground-truth position trajectories with naturally-occurring landmarks, and they also store a great load of memory compared to simple fiducial markers [18]. Another drawback is that there are not too many common features for quays in general. This makes it more challenging to generalize and a machine learning approach would require specific datasets for each quay.

Storing GPS coordinates

It is possible to store information about the position and orientation of the marker (in a global inertial reference frame) inside the marker itself. This makes it possible for a computer vision system without hardcoded position information about each marker at each quay on-board. However, it may be desirable with a simple, distinct marker rather than a sophisticated marker filled with a lot of information in order to simplify the detection. This trade-off should be considered when making a marker for robust marker detection for outdoor applications.

Maintenance of outdoor markers

2D barcode systems also need to be well maintained. If they are changed or distorted or simply just snow covers it, it may be useless as a reference point in a positioning system. Other environmental factors like fog may reduce the camera sight and make the markers useless as well.

CNN for marker detection

Detecting and locating fiducial markers in complex backgrounds is a challenging step. A lot of research has been done in this field. Zhang et al. [19] propose a method to detect non-uniformly illuminated and perspectively distorted 1D barcode based on textual and shape features, while Xu et al. [20] developed an approach for detecting blur 2D barcodes based on coded exposure algorithms. The mentioned methods show high detection rates

Figure 2.5: Boston Dynamics robot *Handle* combines depth cameras with ArUco markers to localize itself accurately and thereby handle different logistics tasks with a mobile manipulator. Image courtesy of Boston Dynamics [22].

on certain barcodes, but their performance may be affected by environmental conditions. The mentioned methods are based on handcrafted features using prior knowledge of specific conditions. However, convolutional neural networks (CNN) have shown outstanding robustness in terms of detecting objects in arbitrary orientations, scales, blur and different light conditions with complex backgrounds as long as such examples are widely represented in the data set [21]. Therefore, a machine learning approach may be recommended to achieve robust outdoor detections of fiducial markers.

How many markers are needed?

Another question is how many markers are needed to get a pose estimate. Since we assume the position and the orientation of the marker in a global inertial reference frame are known, the range from the marker and its attitude, respectively, is known. Therefore, only one marker is needed to estimate the position and orientation of the camera in a global reference frame. However, it may be recommended to place several markers nearby the quay to achieve redundancy. If one single pose estimate differs significantly from the other pose estimates of other markers at the same time or simply just differ more than a certain threshold compared to the previous pose estimate, it is natural to reject it. Therefore, placing several markers can potentially increase the robustness of pose estimates. From Figure 2.5, one can observe that the depth camera will detect two ArUco markers in a pair for close-range operations where the robot manipulator needs to know its pose accurately in order to successfully operate safe and effective.

Markers for relative motion

In this project, it is assumed that the marker is static and fixed to the quay. However, it is also possible to estimate the relative motion between two objects (i.e. none are static with respect to its surroundings). In [23], a camera system is combined with two Motion Measurements Units (MRUs) in a sensor fusion algorithm to estimate the relative motion between two vessels. By placing one MRU at the first vessel and the other inside an ArUco cube at the second vessel, the camera system can measure all 6DOFs between the camera body-fixed coordinate system and the secondary MRU's body-fixed coordinate system (inside the ArUco cube). By assuming both MRUs are fixed with respect to each vessel (as well as the camera), the absolute orientation and position offset between the two body-fixed coordinates of the two MRUs can be calculated quite easily. Together with MRU measurements, relative motion between the vessels can be obtained which allows an offshore crane onboard the first vessel to pick up the ArUco cube onboard the second vessel safely as shown in [23].

Chapter 3

Convolutional Neural Networks

This Chapter aims to introduce the theory related to some of the techniques used in the implementation in Chapter 6. This includes theory related to the structure of a convolutional Neural Network (CNN) as well as how it can be trained and evaluated. In addition, different state-of-the-art object detection models will be reviewed and compared. At last, the chosen model for the implementation, YOLOv3 [24], will be given some extra attention. This Chapter is mainly based on my project thesis.

3.1 Convolutional Neural Networks

On the highest level, CNN can be seen as a black-box. The input data can be one or several images, and the output is typically the predicted class score for each class it has been trained for. The black-box name is frequently used in the context of CNNs with deep layer structure. It has been proven that the interpretability of a CNN tends to decrease as the number of hidden layers increases (which again means the CNN tends to be more black-box-like). In other words, when a CNN is treated as a black-box, the humans can only control the input and observe the output of the model but have no idea on why the model arrived at some specific decision.

Figure 3.1: Black-box view of a CNN for image classification.

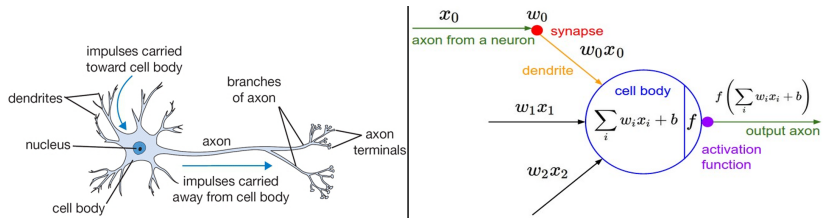


Figure 3.2: The biological neuron to the left and its mathematical model to the right. Image courtesy of Stanford University [25].

3.1.1 Neurons and layers

The field of Neural Networks (NN) has originally been inspired by biological neural systems but has since diverged and become a matter of achieving good results for machine learning tasks. To understand how a CNN is built up we first take a look at how *neurons* work (on high level) and its connection to biological systems.

Each neuron receives input signals from its *dendrites* and produces output signals along its (single) axon, as seen in Figure 3.2. The axon typically branches out and connects via synapses to dendrites of other neurons (e.g. connects neurons to the next layer). The magnitude of the input signal traveling along the axons (x) depends on the synaptic strength (w). The basic idea is that the synaptic strength w is learnable and control the influence of one neuron on another. In the basic model, the dendrites carry the signal to the cell body where they all get summed. If the final sum is above a certain threshold, the neuron can fire, sending a spike along its axon. And the firing rate (out of each neuron) is modeled with an activation function f [25].

Now, with some background on neurons and its biological inspiration we further define the most fundamental terms of any neural network, which is *neuron*, *activation function* and *layer*. These terms give the basis for any neural network.

Definition 3.1.1. *Neuron: A single instance of one layer of a neural network. A neuron receives one or multiple inputs and sums them together to produce a single output that is passed through an activation function.*

Definition 3.1.2. *Activation function: An activation function decides whether a neuron will be pushed forward into the next layer in the neural network or not. In addition, the activation function aims to introduce non-linearity into the output of a neuron.*

Definition 3.1.3. *Layer: A set of neurons, each receiving the same input instances. The input instance(s) can vary as they may be weighted differently.*

The input x of the network in Figure 3.3 is a 1-dimensional input which is weighted by some of the weights in $\boldsymbol{W}^1$. In context of image classification, it could be a single, monochromic pixel. But how is the activation output of a neuron in a certain layer l computed? For a specific example, the output of the upper neuron $\boldsymbol{z}_1^2$ in Figure 3.3 is obtained by multiplying the input signal x (a pixel value) with the weight $\boldsymbol{w}_{11}^1$, and summing it with

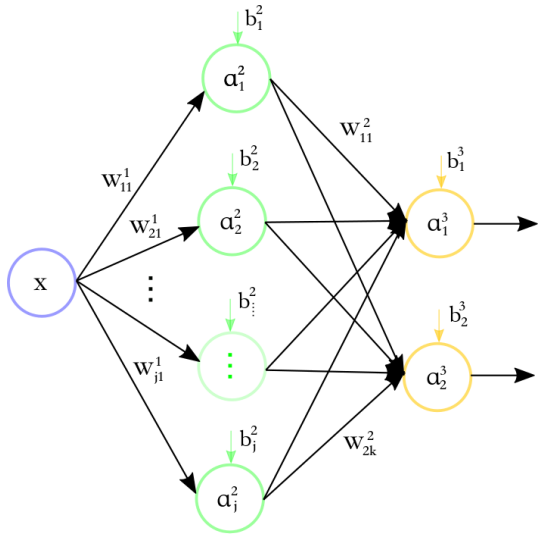


Figure 3.3: Single layer, fully connected neural network.

Table 3.1: Model parameters of a neural network.

Model parameter	Description	Vectorized form
w_{jk}^l	Weight parameter entering layer l a neuron j from neuron k	$\boldsymbol{W}^1 \in \mathbb{R}^{j \times k}$
b_j^l	Bias parameter of layer l to neuron j	$\boldsymbol{b}^1 \in \mathbb{R}^{j \times 1}$
a_j^l	Activation function (e.g. output) of layer l to neuron j	$\boldsymbol{a}^1 \in \mathbb{R}^{j \times 1}$

the bias b_1^2. The single result, $\boldsymbol{z}_1^2 = w_{11}^1 x + b^1$, is then processed through an activation $a_1^2 = \sigma(\boldsymbol{z}_1^2)$ which becomes an input signal for neurons in the next layer. Note that the next layer receives input from several neurons and hence, it needs to sum these. Now, these computations for a specific layer can be generalized for an arbitrary layer l. That is, the linear operations of a single neuron l can be obtained as

$$\boldsymbol{z}_j^l = w_{jk}^l a^{l-1} + b_j^l \,. \tag{3.1}$$

The operation can obviously be expanded from a single operation to include all activations spanning the whole layer such that

$$\boldsymbol{z^l} = \boldsymbol{W^l}\ \boldsymbol{a^{l-1}} + \boldsymbol{b^l}. \tag{3.2}$$

This linear operation is passed through the activation function $\boldsymbol{a^l} = \sigma(\boldsymbol{z^l}) = \sigma(\boldsymbol{a^{l-1}} + \boldsymbol{b^l})$ introducing a nonlinear output value. Observe the model parameter $\boldsymbol{b^l}$ which is usually referred to as the bias. The bias is used to adjust the triggering delay of the activation function. Further details on activation functions will be presented later in this Chapter. Finally, Table 3.1 summarizes the inputs and output in a neural network layer (except the input layer of pixel values from the original image).

3.1.2 Building blocks of a CNN

Convolutional layer

The core of the CNN is the convolutional layer. It consists of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. A filter kernel slides (convolves) the input image by multiplying the values in the filter with the original pixel values of the image (e.g. element-wise multiplication). These multiplications are all summed up to a single number. After sliding the filter over all the locations, all these single numbers (sorted into an array) represents a feature map.

The network treats these filters as model-parameters, e.g. they can be learned. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge [26]. As an example, see Figure 3.4 where a region of the image (the receptive layer) is convolved with the filter. This filter will indeed be activated as the element-wise multiplication and summation will result in a very big number. However, when the same filter is slided over other locations in the image, the same significant number will not be produced and in that sense not be activated. Every filter will initially be small spatially and look for low-level features such as in Figure 3.4. As the feature maps are passed through the layers in the CNN, the number of filters will decrease but also increase in size spatially. In other words, all the filters representing activation of low-level features in the first layers will be transformed to less, but more complex, high-level features. Thus, many small filters are needed to build representations of complex classes (even for a cartoon mouse).

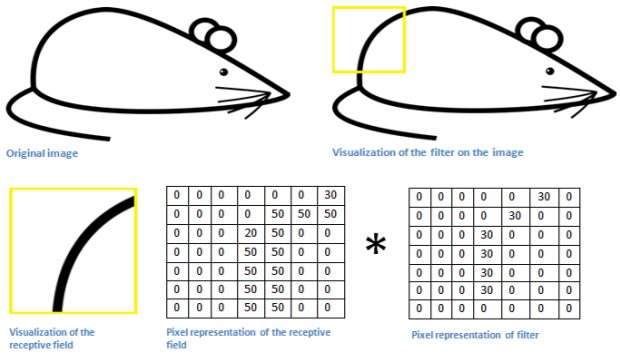


Figure 3.4: Convolutional filter visualized. A specific filter slides over the original image and activates when it recognize the low-level feature it is looking for. Image courtesy by [27].

With some intuition on how the convolutional layer works, we can describe it more generally. What characterizes any respective convolutional layer are hyperparameters. Note that the hyperparameters are not related to any optimization, in contrast to the model parameters. A convolutional layer can be described by 4 hyperparameters:

1. **K** number of filters: It controls the number of low-level features to learn from the input image in CNNs.
2. Spatial extent **F**: The region in the input image that a CNN is sliding over.
3. Stride **S**: It specifies how many pixels we slide the filter for each time. E.g. when the stride is 2, the filter jump 2 pixels at a time.
4. Amount of zero padding **P**: It is used to pad the input volume with zeros around the border. It allows us to control the spatial size of the output volumes.

Given some input volume of size $\boldsymbol{W_1} \times \boldsymbol{H_1} \times \boldsymbol{D_1}$ (i.e. RGB input image have 3 channels such that $\boldsymbol{D_1} = 3$), it can be shown that the spatial size of the output is given by

$$\boldsymbol{W_2} = 1 + \frac{\boldsymbol{W_1} - \boldsymbol{F} + \boldsymbol{2P}}{\boldsymbol{S}} \tag{3.3a}$$

$$\boldsymbol{H_2} = 1 + \frac{\boldsymbol{H_1} - \boldsymbol{F} + \boldsymbol{2P}}{\boldsymbol{S}} \tag{3.3b}$$

From equation (3.3), the size of the convolutional layer output will be $\boldsymbol{W_2} \times \boldsymbol{H_2} \times \boldsymbol{K}$. That is, the amount of neurons equals $\boldsymbol{W_2} \times \boldsymbol{H_2} \times \boldsymbol{K}$ and for large input images results in a large amount of weights. Fortunately, CNNs take advantage of parameter sharing. That is, parameters are shared between neurons throughout the depth of the layer and results in

$\boldsymbol{K} * \boldsymbol{F} * \boldsymbol{F} * \boldsymbol{D_1}$ unique weights. In contrast, if every neuron had a unique weight as in a fully connected layer, the total would be $\boldsymbol{W_2} * \boldsymbol{H_2} * \boldsymbol{K} * \boldsymbol{F} * \boldsymbol{F} * \boldsymbol{D_1}$ model-parameters [28].

Rectified linear unit

To determine if a neuron has been activated (e.g. fired), the batch of input images normally process through an activation function to create an activation map. Rectified Linear Unit (ReLU) is one popular activation function used between layers in a CNN in the last years. ReLU is specially designed for scale invariance and efficient computations, which makes it a suitable candidate for activations of a convolutional layers. It is a non-linear activation function that thresholds output of the convolutional layer such that $\sigma(\boldsymbol{Z}^1) = max(0, \boldsymbol{Z}^1)$. E.g., it returns 0 if it receives any negative input, but for any positive value $\boldsymbol{Z}^1$ it returns that value back.

Many activation functions struggle with the vanishing gradient problem. That is, unless the input value is within a narrow range, the flat derivative makes it difficult to update (and hopefully improve) the weights with gradient descent. And the problem increases with the number of layers in the CNN. In comparison with other activation functions like sigmoid and tanh, ReLU faces the vanishing gradient problem in less degree. This is because the derivative is 1 for positive inputs (e.g. half of the range) which allows gradient descent to keep progressing. In contrast, the derivative of tanh converges towards zero for input values outside the range $(-2, 2)$.

Pooling layer

A limitation of the feature map output of a convolutional layer is that they produce the exact position of features in the input. This makes the feature map very sensitive, e.g. small movements in the position of the features in the input image will result in a different feature map. A common approach to address this problem is to apply a downsampling technique called pooling. It creates a lower resolution version of an input image that still contains the important structural elements, but without the fine detail that may not be useful for the learning task (e.g. it controls overfitting). Hence, the resulting pooled feature map represents a summarized version of the features detected in the input image [29].

In neural networks, it is common to periodically insert pooling layers between the convolutional layers. Beside controlling overfitting, its main function is to reduce the amount of parameters and computations in the network. Further, different pooling operations exist and shortly said they determine how the output feature map (from each convolutional layer) is filtered. The two common pooling operations are average pooling and max pooling. Average pooling calculates the value for each patch of the feature map, while max pooling calculates the maximum value for each patch of the feature map (see Figure 3.5). The pooling layer requires two hyperparameters: Stride **S** and Spatial dimension **F**. Note that max pooling is mostly used, and especially with 2×2 pixel size. A bigger pixel size

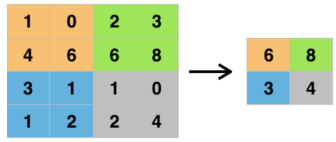


Figure 3.5: Max pooling operation.

will tend to loose more information.

Fully connected layer

In a Fully Connected (FC) layer, each neuron has full connectivity to all activations in its previous layer, as seen in Figure 3.3. In classification tasks, this is normally the final layer. That is, after feature extraction is done the model outputs the data into one or several classes, which can be done using a FC layer. It is usually the most computationally expensive layer.

From Figure 3.1, we can observe the black-box view of the CNN outputting activations in the final FC layer, which are the final classification scores. A classification score represents a measure of the relative activation strength of a class compared to the other classes, and the scores typically lie in the interval $(0, 1)$. To obtain a relative classification score, the output activations for each layer need to be related to each other in a meaningful way. In this context, a normalized softmax function is typically used to achieve this. It is obtained as

$$\sigma(\boldsymbol{z^l})_k = \frac{e^{z_k}}{\sum_{i=1}^{j} e^{z_i}} \quad \text{for} \quad k = 1, ..., j \tag{3.4}$$

The function normalizes the input $\boldsymbol{z^l}$ into a vector of values that follows a probability distribution whose sums up to 1 in total. Hence, the output vector $\boldsymbol{a^l} = \sigma(\boldsymbol{z^l})$ produce values in the range $(0, 1)$ for each class k. The number of classes k are defined by the number of neurons in the final FC layer. Or more generally, when the softmax function is used in intermediate layers, the number of output activations are denoted by the number of neurons j in the l^{th} layer.

3.1.3 The big picture

Now, with some intuition on how the most commonly used building-blocks of any CNN works, we can finally take a look at a typical (high-level) CNN architecture. The general architecture follows the recipe: INPUT-CONV-RELU-POOL- ... -CONV-RELU-POOL-FC. That is, a repetition of CONV- RELU-POOL between input and output. Here, the pooling layer ensures that the spatial size is reduced throughout the network. Figure 3.6 shows an example architecture for image classification of a car. It consists of 6 convolutional layers, each with a ReLU layer in between. It is also down-sampled by 3 max-

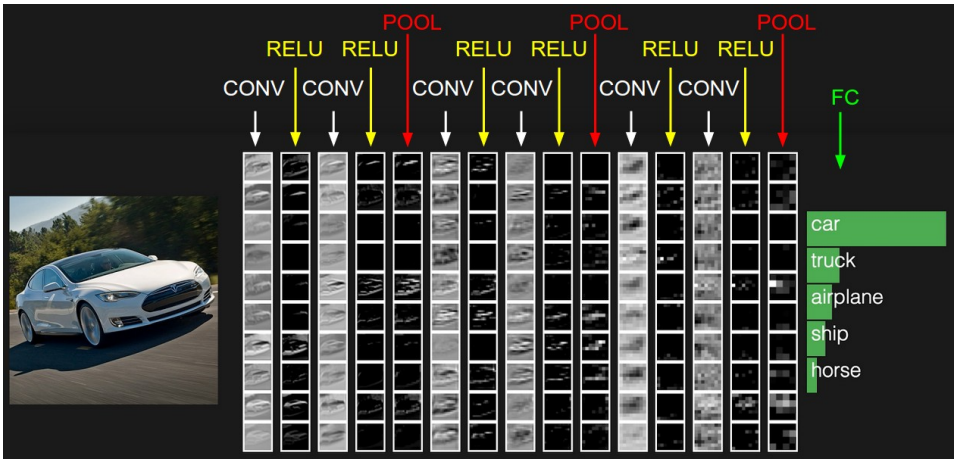


Figure 3.6: Activations of an example CNN architecture for car recognition. Image courtesy of Stanford university [26].

pooling layers after some of the ReLU activations. The final classification scores are obtained in the FC layer using some kind of activation function (i.e. softmax).

3.1.4 Training a CNN

The process of determining the optimal model parameters for a CNN is called *training*. Ground truth classes (made by an annotation program) are compared with model predictions to minimize an objective function. For each input, the weights and biases of the network are adjusted to minimize this objective function. The reason is to correct the probability for the image instance with respect to ground-truth annotations. Usually, the mentioned objective function is backpropagated and then, the weights and biases are updated using a Gradient Descent (GD) algorithm (e.g. an optimizing procedure). This section will look into details regarding the most commonly used training procedures for CNNs.

Backpropagation

During a forward-pass of a batch of annotated training images through a CNN, the CNN outputs a vector of activation outputs $\boldsymbol{a}^L$ as described in 3.4. Further, the ground truth for each class can be obtained as

$$\boldsymbol{y} = \{\boldsymbol{y} \in \mathbb{R}^{j \times 1},\ 0 \leq y_j \leq 1\}. \tag{3.5}$$

When tuning the model-parameters to achieve better classification the euclidean distance distance between the CNN output, $\boldsymbol{a}^L$, and the desired (ground-truth) output, $\boldsymbol{y}$, is subject to a quadratic cost function

$$C = \frac{1}{2N} \sum_{x} \left\| \boldsymbol{y}(\boldsymbol{x}) - a^{L}(\boldsymbol{x}) \right\|^{2} \tag{3.6}$$

where a single sample from a batch is denoted $\boldsymbol{x}$, N is the batch-size of the training examples and L is the total amount of layers in the neural network. Now, this cost function often referred to as the loss function, is what we want to minimize. That is, we want to minimize the term $\left\| \boldsymbol{y}(\boldsymbol{x}) - a^{L}(\boldsymbol{x}) \right\|^{2}$ for all samples $\boldsymbol{x}$ in a batch.
Now, having defined the cost function, the next step is to investigate the impact the weights and biases in the neural network have on the cost function. E.g. in order for backpropagation to work we need to compute the partial derivative of the cost function with respect to any weights and biases denoted as

$$\frac{\partial C}{\partial \boldsymbol{w^{l}}}, \frac{\partial C}{\partial \boldsymbol{b^{l}}}. \tag{3.7}$$

To do this, we need to make some assumptions about our cost function, C, in order to apply backpropagation. The first assumption is that the cost function can be written as an average over cost functions C_x for individual training examples x

$$C = \frac{1}{N} \sum_{x} C_x \tag{3.8}$$

where

$$C_x = \frac{1}{2} \left\| \boldsymbol{y} - \boldsymbol{a^{L}} \right\|^{2}. \tag{3.9}$$

This assumption lets us compute the partial derivative $\partial C_x / \partial \boldsymbol{w^{l}}$ and $\partial C_x / \partial \boldsymbol{b^{l}}$ for a single training example. Then, $\partial C / \partial \boldsymbol{w^{l}}$ and $\partial C / \partial \boldsymbol{b^{l}}$ are obtained by averaging over training examples. Secondly, we assume that the cost function can be written as a function of the activation outputs from the neural network such that $C = C(\boldsymbol{a^{L}})$. Note that the desired output $\boldsymbol{y^{L}}$ is a fixed parameter and in that sense, not a parameter we would like to change by adjusting the weights and biases. Intuitively, it does not make sense to change the ground truth to be learned during training and C should therefore be a function of the output activations $\boldsymbol{a^{L}}$ only.

To understand how the weights and biases in a network change the cost function, we define a small linear change to the activation function, $\sigma(z_j^l + \Delta z_j^l)$, to determine its effect on the outcome of the cost function. This change then propagates through the layers causing the overall cost to change by $\frac{\partial C}{\partial z_j^l} \Delta z_j^l$. Further, we introduce an intermediate quantify, δ_j^l, which defines the error in the j^{th} neuron in the l^{th} layer such that

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \tag{3.10}$$

and subsequently in vectorized form δ^l is the error in the layer l. Now, backpropagation

gives a way to find this error, δ^l for each layer, and relate them to $\partial C \partial \boldsymbol{w^l}$ and $\partial C \partial \boldsymbol{b^l}$. Four equations describe the backpropagation procedure:

$$\delta^L = \nabla_a C \circ \sigma^{'}(z^L) \tag{3.11a}$$

$$\delta^l = ((w^{l+1})\delta^{l+1}) \circ \sigma^{'}(z^l) \tag{3.11b}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \tag{3.11c}$$

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{3.11d}$$

First, equation (3.11a) describes the error in the output layer of the network which are necessary (initially) for computation of equation (3.11b). That is, to compute the error in the previous layer with respect to the last output layer. Or more generally, as the equations backpropagate throughout the network, the error in the l layer with respect to the next layer $l+1$. In addition, equation (3.11c) describes the rate of change of the cost with respect to any bias in the network, while equation (3.11d) describes the rate of change of the cost with respect to any weight in the network. As the reader may observe, these equations are solved from top to bottom making a chain of computations for each layer. By this, backpropagation involves to solve this chain of computations from the output layer to the previous layer and repeat this process throughout the network until it reaches the first layer. This means we can finally measure the impact the weights and the bias in the neural network have on the cost function, as earlier mentioned. For more information on how the backpropagation equations are derived, see [30].

Gradient descent

With some knowledge on how the cost function C is computed for a given input, it is time to review a popular optimization technique called Gradient Descent (GD), which is used broadly when training a CNN. It is used to adjust the model-parameters such that the cost function is minimized. Specifically, the weights and biases in the network can be updated according to

$$\boldsymbol{W^l} \rightarrow \boldsymbol{W^l} - \eta \frac{\partial C}{\partial \boldsymbol{w^l_{jk}}} \tag{3.12a}$$

$$\boldsymbol{b^l} \rightarrow \boldsymbol{b^l} - \eta \frac{\partial C}{\partial \boldsymbol{b^l}} \tag{3.12b}$$

where η is the learning rate controlling the step-size for each computation. Indeed, a low learning rate will cause smaller steps towards the local minima of the cost function and thereby slowing the learning process. On the other hand, a too high learning rate can cause divergence in training as the steps are too big and may never reach the local minima at all. Also, pay attention to the minus sign in front of η as it is desirable to move towards

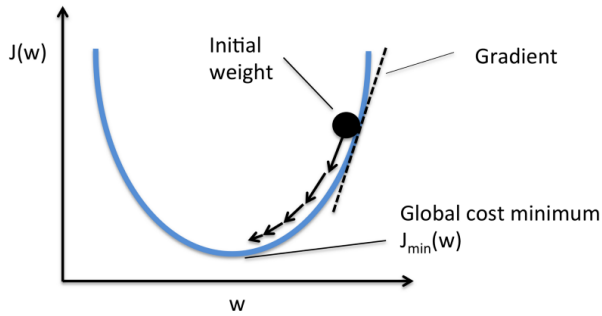


Figure 3.7: Gradient Descent procedure illustrated in 2 dimensions. For simplicity, the cost function, here denoted as $J(\boldsymbol{w})$, is only optimized with respect to the weights $\boldsymbol{w}$. Image courtesy of hackernoon [31].

the local minima (using small steps along the gradient) and not away from it! From Figure 3.7, we can see how some weight proceeds step-wise along the gradient of the cost function C given some initial starting point. Also, notice how the steps tend to decrease as the weights get closer to the optimal minimum. Further, the weights and biases are only updated after each batch of training images is evaluated. The batch size determines the size of the subset of training images used for one iteration and also, how many images are averaged over at the same time. Therefore, in order to reduce the sensitivity from noisy outliers in the cost function, it may be desirable to pick a batch size which is not too small.

At last, it is worth to mention that the gradient descent algorithm may be infeasible when the training data is huge. That is, the computational cost of gradient descent scales linearly with training data set size n meaning that if n is high, the computational cost of gradient descent is also high [32]. Stochastic Gradient Descent (SGD) is used to address this problem. The main idea is to use randomly (e.g. stochastic) selected training examples to evaluate the gradients. This smaller subset of the whole training set is still representative due to the randomly selected images. The path to reach the minima is usually noisier due to its randomness. However, SGD algorithm does still reach the minima in shorter training time if the training parameters are tuned correctly.

3.1.5 Evaluating a CNN

To evaluate a CNN, some metrics are needed to measure its accuracy. Concretely, the accuracy of a CNN is determined by calculating the Precision-Recall (PR) curve for the CNN given a set of annotated test-images. This means the model evaluation metrics precision and recall are needed. Shortly said, the precision represents the relevant instances of detection, while the recall is the amount of relevant instances that are retrieved [33]. To compute these metrics it is necessary to introduce the four distinct output instances (i.e. possible outcomes of a binary classification). Table 3.2 describes these four possible outcomes given a data set with labeled ground truth objects. Note that the ground truth objects in the test images are annotated, usually manually by a human evaluator.

Table 3.2: Relevance representations.

Full name	Abbrevation	Description
True Positive	TP	The amount of positive classifications correctly classified with respect to ground truth.
False Positive	FP	The amount of positive classifications wrongly classified with respect to ground truth.
False Negative	FN	The amount of ground truth not identified by the classifier.
True Negative	TN	The amount of correct rejections by the classifier relative to ground truth.

With these four possible outcomes, the precision and recall of a CNN can be computed as

$$p = \frac{TP}{TP + FP} \tag{3.13a}$$

$$r = \frac{TP}{TP + FN} \tag{3.13b}$$

where p denotes the precision and r denotes the recall. Given the definition of precision and recall, the Average Precision (AP) can be computed. According to VOC2012 documentation [34], AP can be computed using the Area Under Curve (AUC) method, that is:

1. Compute a version of the measured precision/recall curve with precision monotonically decreasing, by setting the precision for recall r to the maximum precision obtained for any recall $r' < r$.

2. Compute the AP as the area under this curve by numerical integration.

The AUC method is used for evaluation in ImageNet as well, a popular data set reviewed in Chapter 6. Note that although the principle of measuring AP follows the same procedure, the exact calculation may vary from dataset to dataset. Further, for the multiclass scenario, the term mean Average Precision (mAP) is used. This is simply the mean of AP computed for all classes. These definitions of AP and mAP will be used throughout the thesis.

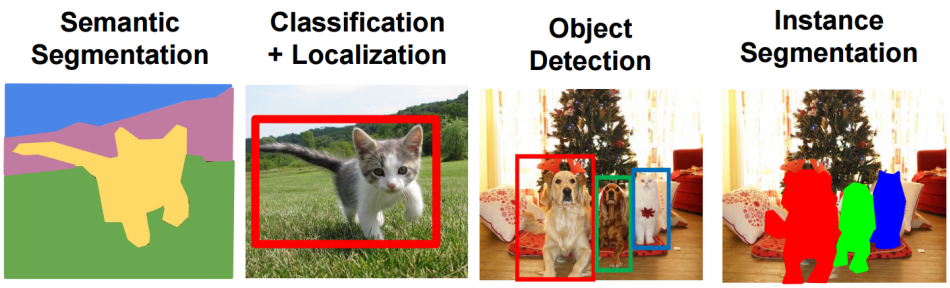


Figure 3.8: Object detection compared to closely related concepts. Image courtesy of [35].

3.2 Deep CNNs for Object Detection

Now, some theory related to CNNs have been presented and it is time to look into what object detection is and review some state-of-the-art object detection models within the deep learning field.

3.2.1 Object detection

Object detection involves detecting and classifying multiple objects and where they are in the image marked by rectangle-shaped bounding boxes. In this context, closely related concepts like classification and segmentation exist. First of all, classification simply focuses on whether a single object exists in an image or not, but its pixel coordinates (e.g. localization) are left unknown. Further, there are mainly two types of segmentation. Semantic segmentation is able to distinguish between classes (but not between objects within a class) at pixel-level. Usually, colors are used to illustrate this as shown in the left image of Figure 3.8. Instance segmentation is a slightly harder task as it aims to not only distinguish between classes but also distinguish between objects at pixel-level. In fact, instance segmentation only differs slightly from object detection. That is, object detection use bounding boxes to assign classes, while instance segmentation assigns each class at pixel-level (which is more computationally expensive). Mask R-CNN is a known model for instance segmentation.

3.2.2 State-of-the-art models

Here, several state-of-the-art models are presented. Both single-shot detectors and region-based detectors will be reviewed. Note that at the time this thesis was written, Faster R-CNN may not represent state-of-the-art anymore (in context of real-time performance), but it represents together with its predecessors how fast the development of CNNs for object detection progress.

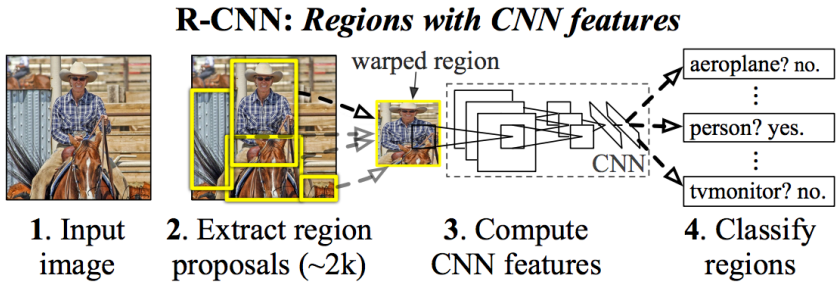


Figure 3.9: R-CNN framework. R-CNN takes input image, creates a set of regions, computes features for each proposals using a deep CNN and classifies each region using class-specific linear SVMs. Image courtesy of [36].

Faster R-CNN

Before discussing the Faster R-CNN, it is necessary to review its predecessors, R-CNN and fast R-CNN, to get some context of the basics for these models as well as how they are improved step by step.

It all started with an early application of CNN to object detection called Region-based CNN (R-CNN). The overall goal of R-CNN is to take in an image and correctly identify where the main objects in an image are located. This is done by four steps (see Figure 3.9):

1. Generate a set of region proposals for an image.
2. Process the region proposals using a CNN.
3. Compute task-specific output.
4. Linear regression: Improving the bounding boxes.

The first step, region proposal, creates bounding boxes using selective search. Selective search is a method that looks at the image through windows of different sizes (also called sliding window) and for each tries to group together pixels in same region by texture, color or intensity for identification of objects. Next, the region proposals are transformed to standard square size and pass it through the CNN as shown in the slide. Overall, the convNet works as a feature extractor and classifier, e.g. it transforms the input region layer by layer to final class scores. The third step is the final layer of the CNN where a support vector machine (SVM) classifies whether this is an object or not, and if so, what object it is. Finally, we tighten the box to fit the true dimensions of the object by running a simple linear regression on the region proposal [36]. To sum up, the R-CNN takes in sub-regions of the image corresponding to objects and outputs new bounding box coordinates for the object in the sub-region.

Now, although R-CNN works fine, it is quite slow as it:

1. Requires forward pass of the CNN for every single region proposal for every single image.

2. Have to train three different models separately: The CNN to generate image features, the classifier that predicts that class and the regression model to tighten the bounding boxes. This results in a quite slow and ineffective pipeline.

The improved version, Fast R-CNN, address problem 1 (which introduced slow performance) by applying ROI (Region of Interest) pooling. ROI pooling refers to cropping a part of a feature map and resizing it to a fixed size [37]. Specifically, this means to run the CNN once per image and find a way to share computations for sub-regions. For one image, it shares the forward pass of the CNN across its subregions. The CNN features are obtained by selecting a corresponding region from the CNN's feature map. Then the features in each region are pooled (often max-pooled). By potentially running thousands of regions through the CNN with R-CNN, you only need one per image with Fast R-CNN which dramatically speeds up the performance.

Problem 2 is solved by using a single network to compute all three modules (extract image features, classify with SVM and tighten bounding boxes with regressor) at the same time. Notice that the SVM classifier is replaced with a softmax activation on top of the CNN to output classifications. It also adds a linear regressor in parallel to output bounding box coordinates [37]. In other words, all outputs needed come from one single network.

Even with the improvements in Fast R-CNN, there was still a bottleneck with the region proposer. In the first step, selective search was applied to generate a set of region proposals which is fairly slow. A team of researchers at Microsoft realized that the convolutional feature map used by the region-based detector can also be used for generating region proposals. Therefore, by reusing the same CNN computations instead of running a separate selective search algorithm increased the performance greatly [37].

Mask R-CNN

Briefly said, Mask R-CNN extends faster R-CNN for pixel level segmentation. The basic idea of Mask R-CNN is to go one step further and locate the exact pixels of each object instead of just bounding boxes. This is known as instance segmentation. It adds a binary mask to the Faster R-CNN that outputs whether or not a given pixel is a part of an object. However, the binary mask leads to some challenges. The regions of the feature map selected by the ROI Pool becomes slightly misaligned. As an example, an 128×128 image with 15×15 region reduced to a feature map of 25×25 pixels will give a corresponding region of 2.93×2.93 pixels. ROI pool will round these numbers to 2 pixels. However, using ROIAlign (Realigning ROl pool) such roundings are avoided by using bilinear interpolation [37]. With this improvement, Mask R-CNN can generate more precise segmentation.

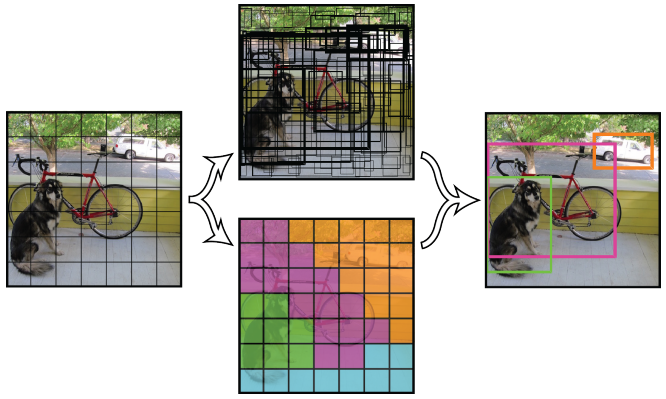


Figure 3.10: YOLO framework. Given an input divided into $S \times S$ grids, YOLO produce bounding box predictions (upper image) and a class probability map (lower image). These predictions are merged to final detections as seen in the right image. Image courtesy of [38].

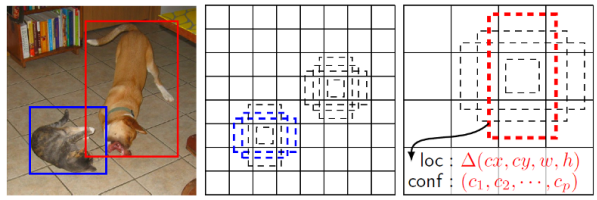
YOLO (You Only Look Once)

YOLO (You Only Look Once) is a popular single shot detector used for object detection tasks. The main difference between YOLO and the R-CNN models is that R-CNN use different region proposals and run those proposals through a CNN, while YOLO passes the (whole) image just once.

As seen in Figure 3.10, YOLO splits the input image into smaller grids. And each grid cell predicts a constant number of bounding boxes. If the center of an object falls into a grid cell, that grid is responsible for detecting that object [38]. Each bounding box contains a confidence scores on how certain it is that an object exists or not within the bounding box. Notice that many of the bounding boxes (in the upper image in Figure 3.10) will have very a low confidence score and based on a threshold, most of them disappear. Actually, there are only 3 bounding boxes (representing the dog, the bike and the car) left based on the threshold. Every grid also predicts a class (visualized by colors) as seen in the lower image in Figure 3.10. This works as a classifier and gives a probability distribution over all the possible classes that the network has been trained on. More details regarding YOLO will be explained in the next section.

SSD (Single Shot Detector)

The SSD (Single Shot Detector) approach is based on a feed-forward convolutional network that produces a fixed-size collection of bounding boxes and scores for the presence of object class instances in those boxes, followed by a non-maximum suppression (NMS) step to produce the final detections [39]. As with YOLO, SSD only uses a single shot to detect multiple objects within the image. In addition to the early layers inspired by the standard architecture *VGG-16* for high-quality image classification, SSD is characterized by several interesting features. Among these, we find multi-scale feature maps that allow



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 3.11: SSD framework. There are 4 default boxes of different height-width ratios at each location in the feature maps with different scales ((b) and (c)). For each default box, we predict both the shape offsets and the confidences for all object categories. At training time, we match these default boxes (e.g. anchor boxes) to the ground truth boxes in (a). Image courtesy of [39].

predictions of detections at multiple scales. Another feature is default boxes and aspect ratios. That is, we associate a set of default bounding boxes with each feature map cell (as seen in Figure 3.11), for multiple feature maps at the top of the network. At each feature map cell, we predict the offsets relative to the default box shapes in the cell. As seen from Figure 3.11, we evaluate 4 default boxes of different aspect ratios at each location in several feature maps with different scales (e.g. 8×8 in (b) and 4×4 in (c)). For each default box, we predict both the shape offsets and the confidences for all object categories [39].

Comparison

It is somehow hard to have a fair comparison among the different object detectors. A bunch of different papers uses different datasets for benchmarking state-of-the-art object detection models. And besides the choice of model, several other things impact the performance such as the choice of feature extractor (VGG16, ResNet, etc), input image resolution and other training parameters like batch size, learning rate and so on.

Therefore we will only discuss briefly some key characteristics for the models we have reviewed. An important aspect to discuss when comparing object detection models is the speed-accuracy tradeoff, where speed is denoted as inference time (e.g. the time it takes for the input data to pass through all the layers in the CNN). Since both YOLO and SSD are in fact single shot detectors, they provide an effective pipeline, which allows for low inference time. In comparison, a region-based method like Faster R-CNN uses different region proposals and run each region through the CNN.

An improved model, R-FCN (Region-based Fully Convolutional Network), is used for benchmark testing on COCO dataset along with YOLOv3 and SSD. From Figure 3.12, one can observe that all the different configurations of YOLOv3 (320×320, 416×416 and 608×608 input resolution) is faster than the region-based CNNs (FPN FRCN and R-FCN). However, FPN FRCN is slightly more accurate than YOLOv3-608 (but more than $3\times$ slower compared to YOLOv3-608). The SSD models have slightly more equal characteristics to YOLOv3, but they are all achieving lower mAP and inference time compared to YOLOv3.

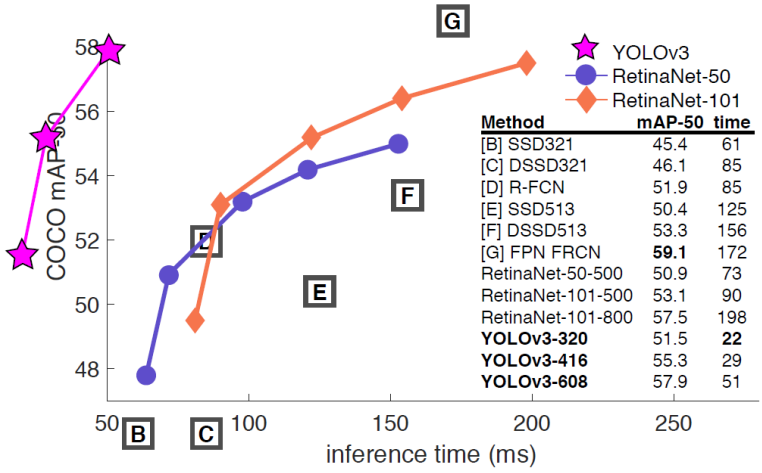


Figure 3.12: Inference time and corresponding mAP for different CNNs. The CNNs is tested at COCO dataset (80 classes) with 0.5 IOU threshold. Image courtesy of [24].

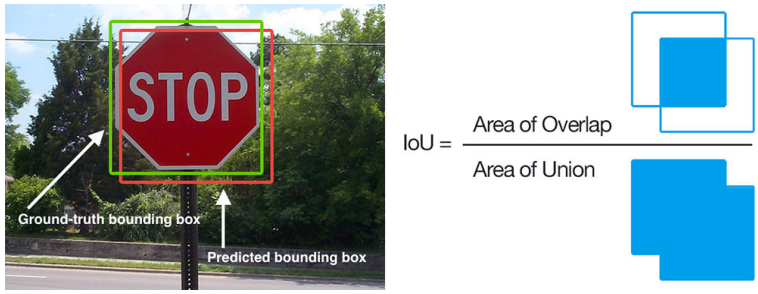


Figure 3.13: Intersection over Union (IoU) metric visualized.

3.3 YOLOv3

So far, different object detectors have been reviewed (including YOLO) as well as theory related to CNNs. As seen from the benchmark test in the previous section, YOLOv3 provides a good tradeoff between mAP and inference time compared to other (reviewed) models. YOLOv3 is especially characterized by its low inference time enabling for detections in real-time. This section aims to present a more detailed description of YOLOv3, which will be used in the implementation.

Bounding box predictions

It turns out that most bounding boxes have a certain height-width ratio. So instead of directly predicting a bounding box, YOLOv3 predicts offsets from a pre-defined set of boxes with a particular height-width ratio. These predefined boxes are referred to as anchor boxes. In [40], a k-mean clustering algorithm is run to obtain a number of anchor

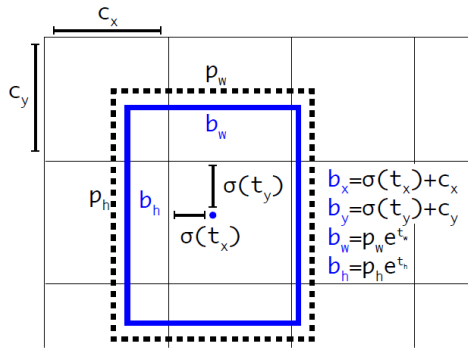


Figure 3.14: Bounding box prediction. The width and height of the blue box is predicted using offsets from the prior known anchor boxes. In the figure, only the closest anchor box with width p_w and height p_h is shown. The other anchor boxes is illustrated in Figure 3.15. The center coordinates of the box are predicted using a sigmoid function. Image courtesy of [24].

boxes, which give a good tradeoff for recall vs. complexity of the model. In YOLOv3 9 anchor boxes are applied, while YOLOv2 have 5 anchor boxes (see Figure 3.15).

The network predicts coordinates for each bounding box, t_x, t_y, t_w, t_h. Now, given that the cell is offset from the top left corner of the image (c_x,c_y) and the bounding box prior has width p_w and height p_h (e.g. the anchor box), then the predictions correspond to

$$b_x = \sigma(t_x) + c_x \tag{3.14a}$$
$$b_y = \sigma(t_y) + c_y \tag{3.14b}$$
$$b_w = p_w e^{t_w} \tag{3.14c}$$
$$b_h = p_h e^{t_h} \tag{3.14d}$$

.

That is, we predict location coordinates relative to the location of each cell. During training, sum of squared error loss is used to improve bounding box coordinate predictions [24].

Furthermore, YOLOv3 predicts an objectness score for each bounding box using logistic regression. The score is 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. This means only one bounding box prior is assigned for each ground truth object. If the bounding box prior is not the best but overlap a ground truth object more than a threshold of 0.5, the prediction is ignored [24].

Class predictions

As seen in Figure 3.10, YOLO also predicts a class probability map. This section will go into detail on the class predictions.

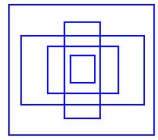


Figure 3.15: The 5 different anchor boxes used in YOLOv2 providing different height-width ratios. They are applied at different scales. In comparison, YOLOv3 provides 9 anchor boxes, 3 for each scale.

So how do YOLO figure out if a bounding box contains a specific type of object? Recall that YOLO divides the input image into $S \times S$ smaller grids (see Figure 3.10). And each grid cell predicts C conditional class probabilities, Pr(Classi$_i$|Object). That is, given that an object exists, what is the probability that this object has a specific class. We only predict one set of class probabilities per grid cell, regardless of the number of boxes B. At test time, we multiply this conditional class probability with the confidence value for the individual box confidence prediction, Pr(Object) (e.g. how confident the model is that the box contains an object). In addition, we multiply with IOU, which takes the union of ground truth bounding box and the predicted bounding box (see Figure 3.13) to obtain the following equation:

$$\text{Pr(Class}_i\text{|Object)} \times \text{Pr(Object)} \times \text{IOU}_{pred}^{truth} = \text{Pr(Class}_i\text{)} \times \text{IOU}_{pred}^{truth} \tag{3.15}$$

This equation gives us the class-specific confidence scores for each box. In other words, both the probability of that class appearing in the box and how well the predicted box fits the object [38].

At last, the output activation for classification is slightly changed. In earlier versions of YOLO, class scores where produced using a softmax function and the class with the maximum score was assigned to be the class of the object contained in the bounding box. However, softmax rests on the assumption that classes are mutually exclusive (e.g. if one object belongs to one class, it cannot belong to another). As an example to illustrate its weakness, classes like Person and Women are not mutually exclusive. Now, YOLOv3 addresses this problem by using a logistic classifier instead, which is able to perform multilabel classification for detected objects [41].

Predictions across scale

In order to detect small objects in the image well, YOLOv3 predicts boxes at 3 different scales. YOLOv3 has 9 anchors, which are grouped into 3 different groups according to their scale. Each group is assigned to a specific feature map for detecting objects (depending on their pixel size). We now introduce some technical details related to how the features are extracted from these scales.

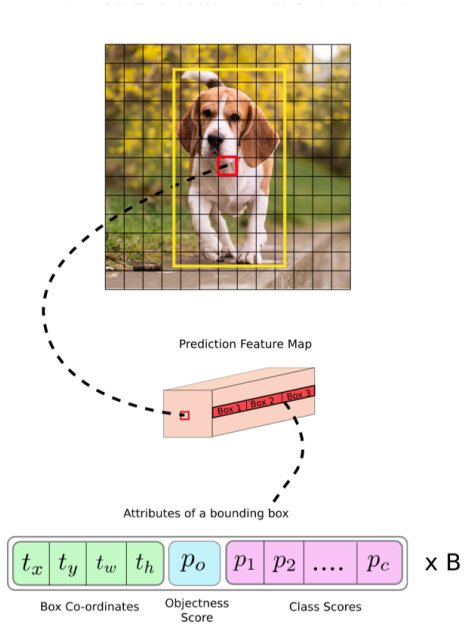


Figure 3.16: The content of a feature map for a single grid cell. B refers to the number of bounding boxes a grid cell can predict. Image courtesy of [41].

Features from these scales are extracted similar to a feature pyramid network (FPN). From our base feature extractor, we add several convolutional layers where the last represents a 3D tensor with bounding box prediction, objectness and class predictions (see Figure 3.16). Next, we take the feature map from 2 layers previously and upsample it by a factor of 2. We also take a feature map from earlier in the network and merge it with our upsampled features using concatenation. With this method, the resulting feature map provides meaningful semantic information and finer-grained information from the earlier feature map (e.g. better spatial information on object locations). We then process the combined feature map through a few more convolutional layers to predict boxes for the final scale [24].

Network architecture

The network architecture used for the YOLO model (independent of version) is referred to as Darknet. For YOLOv3, a new network for performing feature extraction is used. In YOLOv2, Darknet-19, an originally 19-layer network supplemented with 11 more layers, is used. However, YOLOv2 often struggled with small objects due to the loss of fine-grained features. As obtained in the previous section, YOLOv3 is now able to detect smaller objects with the use of feature maps with finer-grained information. Now, YOLOv3 uses a variant of Darknet (Darknet-53), an originally 53 layer network trained on ImageNet. See Figure 3.17 for details. For the detection task, 53 more layers are

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.17: Original Darknet-53 architecture used by YOLOv3. Image courtesy of [24].

stacked onto it, giving us a 106 layer architecture for YOLOv3. As there exist several variants slightly different from the original Darknet-53 architecture (i.e. YOLOv3-spp, YOLOv3-tiny, etc), it will not be given more detailed information about the specific architecture(s).

Chapter 4

Vision-based 3D Reconstruction

This Chapter introduces the theory about traditional computer vision techniques related to the implementation. Basic concepts to perform 3D reconstruction from an image will be presented. That is, how 3D position and orientation measurements can be obtained from single-view and stereo-view geometry. A short introduction to cameras is also given.

4.1 Camera

A camera is a passive sensor used to capture visual information about the scene. It collects the light in the scene during a short time interval (e.g. exposure time) and represents the information, usually in a digital format (i.e. bytes). The camera mainly consists of three components; an imaging sensor to capture the visual information in the scene, a lens to collect and direct the light towards the imaging sensor and a camera housing to ensure the imaging sensor is not affected by other light sources. The digital imaging sensor is by far the most used nowadays, although analog imaging sensors still exist.

4.1.1 Digital image acquisition

Digital image acquisition is the digital encoded representation of the visual characteristics of an object [42]. Today, two types of sensor units are used in digital cameras; complementary metal oxide semiconductor (CMOS) and charge coupled device (CCD). Both units have one main feature in common; they both consist of a grid of pixels that measures the light strength in the scene, which is proportional to the brightness of the light. They do however measure light intensity and convert the captured electrical data into an image file very differently [43]. In terms of computer vision methods, it is not important how they work in detail, and further details on these sensor units will not be given.

Each pixel in a digital image usually consists of parts that measure the intensity of red, green and blue light, unless a greyscale sensor is used, which only measures the total light intensity. Each pixel is assigned a value between 0 and 255. Hence, each color pixel

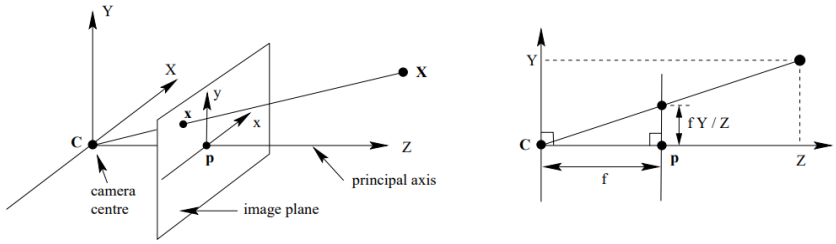


Figure 4.1: Pinhole camera geometry. The left part shows the pinhole camera model. The right part shows mapping of points to the image plane by similar triangles. Image courtesy by [44].

consumes one byte to store the information, which is three times the storage amount of greyscale pixels.

4.2 Single-view Geometry

A camera can be seen as a transformation between the 3D world and a 2D image plane. This transformation can be approximated with a camera model. This section introduces the pinhole camera model, the relationship between the coordinates of a point in 3D space and its projection onto the image plane, assuming an ideal pinhole camera. Further, a model for radial and tangential distortion is presented. The theory in this section is largely based on [44].

4.2.1 The pinhole camera model

The pinhole camera model is a *central projection* of points in space onto an image plane. Let the centre of projections be the origin of an Euclidean coordinate system, and consider the plane $Z = f$, which is called the *image plane*. In Figure 4.1, a 3D point with coordinate $\boldsymbol{X} = (X, Y, Z)^T$ is mapped to the image point where a line joining the point $\boldsymbol{x}$ to the centre of projection meets the image plane. By similar triangles, the point $(X, Y, Z)^T$ can be mapped to the point $(fX/Z, fY/Z, f)^T$ on the image plane (see right part of Figure 4.1). Since the final image coordinate represents a constant distance between the camera center and the image plane, we ignore this coordinate and see that

$$(X, Y, Z)^T \mapsto (fX/Z, fY/Z)^T \tag{4.1}$$

describes the central mapping from 3D world to 2D image coordinates. With homogeneous coordinates, (4.1) can be written in terms of matrix products as

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \mapsto \begin{bmatrix} fX \\ fY \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \text{diag}(f, f, 1) \begin{bmatrix} \boldsymbol{I}^{3\times 3} & \boldsymbol{0}^{3\times 1} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.2}$$

where $\text{diag}(f, f, 1)$ is a diagonal matrix, $\boldsymbol{I}^{3\times3}$ is the 3×3 identity matrix and $\mathbf{0}^{3\times1}$ is a 3×1 zero vector. We now introduce the notation $\boldsymbol{X}$ for the world point given by the homogeneous coordinate vector $(X, Y, Z, 1)^T$ and $\boldsymbol{X}_c$ for the corresponding point in the image plane represented by a homogeneous coordinate 3-vector. Further, we introduce $\boldsymbol{P}$ for the homogeneous 3×4 camera projection matrix, so (4.2) can compactly be written as

$$\boldsymbol{X}_c = \boldsymbol{P}\boldsymbol{X} \tag{4.3}$$

where

$$\boldsymbol{P} = \text{diag}(f, f, 1) \begin{bmatrix} \boldsymbol{I}^{3\times3} & \mathbf{0}^{3\times1} \end{bmatrix}. \tag{4.4}$$

Principal point offset and pixel density

The transformation in (4.3) assumes that the origin of the image plane coincides with the principal point. E.g the point where the principal axis intersects the image plane (see Figure 4.1). In practice, it may not be, so the general mapping between world and image plane coordinates is

$$(X, Y, Z)^T \mapsto (fX/Z + p_x, fY/Z + p_y)^T \tag{4.5}$$

where $(p_x, p_y)^T$ are the coordinates of the principal point. With a principal point offset, this equation can be expressed in homogeneous coordinates as

$$\begin{bmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{bmatrix} = \begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{I}^{3\times3} & \mathbf{0}^{3\times1} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{4.6}$$

where

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & p_x \\ 0 & f & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

is the *camera calibration matrix*.

The last step involves a change of coordinates, from meters to pixels, which is performed with pixel density of the imaging sensor. We define the number of pixels per unit distance in image coordinates along the horizontal and vertical direction as $m_x = n_x/s_x$ and $m_y = n_y/s_y$. Here, n_x, n_y, s_x and s_y represents the number of pixels in the imaging sensor and the physical size of the sensor in meters, respectively. By multiplying (4.7) with $\text{diag}(m_x,\ m_y,\ 1)$, we obtain

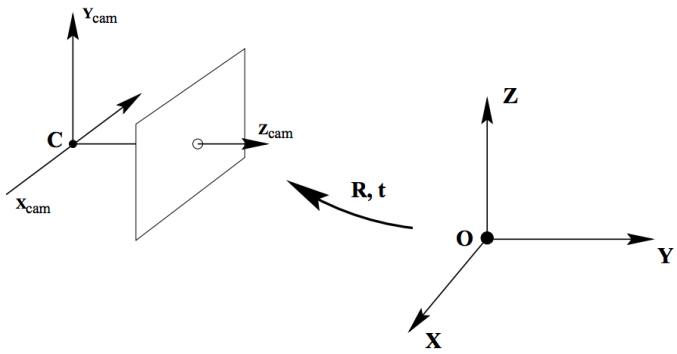


Figure 4.2: The Euclidean transformation between the world and camera coordinates. Image courtesy by [44].

$$\boldsymbol{K} = \begin{bmatrix} \rho_x & 0 & c_x \\ 0 & \rho_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

where $\rho_x = fm_x$ and $\rho_y = fm_y$ describes the focal length in pixel units along the horizontal and vertical direction, respectively. Since the pixels (often) are non-square when measured in image coordinates, the scale factors along the x and y axis, m_x and m_y, are usually not the same. $c_x = m_x p_x$ and $c_y = m_y p_x$ represent the coordinates of the principal point with respect to pixel dimensions. A more general model will also include the skew-symmetric parameter s, which represents non-rectangular pixels. Now, the parameters of the matrix $\boldsymbol{K}$ are known as the *intrinsic parameters*.

Camera rotation and translation

In general, points in space will be expressed in terms of a different Euclidean coordinate frame, known as the *world coordinate frame*. As seen in Figure 4.2, two coordinate systems are related via a translation $\boldsymbol{t}$ and rotation $\mathcal{R}$. Homogeneous coordinates are however convenient for compact representations of a geometric mapping. Consider an arbitrary point $\boldsymbol{X}$ in some right-handed coordinate system. The mapping between two arbitrary Euclidean coordinate frames a and b can be expressed as

$$\boldsymbol{X}^a = \mathcal{R}_b^a \boldsymbol{X}^b + \boldsymbol{t}_b^a \tag{4.9}$$

where $\mathcal{R}_b^a$ is a 3×3 rotation matrix and $\boldsymbol{t}_b^a$ is a 3×1 translation vector relating the two coordinate system. In contrast, the same transformation can be obtained with homogeneous coordinates written as

$$\boldsymbol{X}_b^a = \mathcal{T}_b^a \boldsymbol{X}^b, \quad \text{where} \quad \mathcal{T}_b^a = \begin{bmatrix} \mathcal{R}_b^a & \boldsymbol{t}_b^a \\ \boldsymbol{0}^{1\times 3} & 1 \end{bmatrix} \tag{4.10}$$

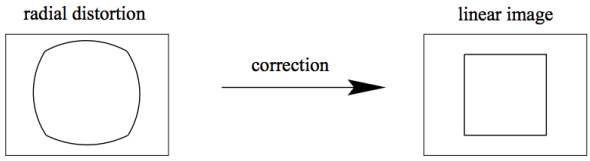


Figure 4.3: A visual illustration of how radial distortion affect an image and how it is corrected. From a real-world perspective, it is easy to observe when straight-line shaped objects tend to bend in the image due to radial distortion. Image courtesy by [44].

and $\boldsymbol{X}^a$ and $\boldsymbol{X}^b$ denotes 4-vectors on the form $[x,\ y,\ z,\ 1]^T$, and $\mathbf{0}^{1\times 3}$ is the 1×3 vector. In contrast, the nonhomogeneous vector $\boldsymbol{X}$ in (4.9) is a 3-vector $[x,\ y,\ z]^T$. Revisiting Figure 4.2, the camera coordinate system and the world coordinate system can be related via a rotation matrix R and translation vector $\boldsymbol{t}$ such that

$$\boldsymbol{X}_{cam} = \mathcal{R}\boldsymbol{X}_{world} + \boldsymbol{t}. \tag{4.11}$$

By combining (4.6) representing the image to camera transformation and (4.11) representing the camera to world transformation, we obtain the *camera matrix* $\boldsymbol{P}$. Thus, matrix $\boldsymbol{P}$ relates a point expressed in the world coordinates with pixel coordinates, which can be written as

$$\boldsymbol{P} = \boldsymbol{K}[\mathcal{R}\ \ \boldsymbol{t}]. \tag{4.12}$$

The parameters of $\mathcal{R}$ and $\boldsymbol{t}$ are called the *extrinsic parameters*. They represent the camera position and orientation with respect to the world coordinate system. Both the intrinsic parameters and the extrinsic parameters are usually found through a calibration procedure.

4.2.2 Distortion model

So far, an ideal pinhole camera model has been introduced. It assumes a linear mapping between the image plane and the world coordinate system. For cameras with real (non-pinhole) lenses, this assumption does in general not hold. Here, radial distortion may have a great impact, especially when wide-angle lenses are used. If the camera is modeled as a pinhole camera, the error becomes more significant as the focal length of the lens decreases. Radial distortion is produced when light rays bend more near the edges of a lens compared to its optical center. The objective is to remove nonlinearities induced by radial distortion, to again obtain a linear relationship between the image and the real world. Figure 4.3 illustrates this.

Now, consider the (radial) distorted image coordinates $[x_d,\ y_d]^T$ related to the ideal (pin-hole) image coordinates $(\tilde{x},\ \tilde{y})$ by a radial displacement, which can be approximated as [44]

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x}(1\ +\ k_1 r^2\ +\ k_2 r^4\ +\ \dots) \\ \tilde{y}(1\ +\ k_1 r^2\ +\ k_2 r^4\ +\ \dots) \end{bmatrix} \tag{4.13}$$

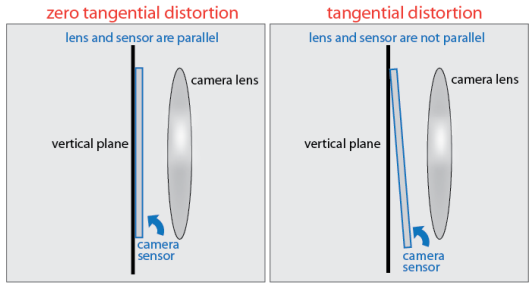


Figure 4.4: Illustration of the ideal zero tangential camera setup to the left, and the less fortunate camera setup causing tangential distortion to the right. Image courtesy by [46].

where $(1 + k_1 r^2 + k_2 r^4 + \dots)$ is a distortion factor that influences the distorted image coordinates $[x_d, y_d]^T$. The parameters k_1, k_2,... are the radial distortion parameters. Normally, two coefficients are sufficient to obtain a satisfactory calibration result. They do not depend on the scene reviewed and remain the same regardless of the captured image resolution [45]. Thus, they belong to the intrinsic camera parameters.

A lens can also produce tangential distortion. This normally occurs when the lens and the image plane are not parallel as seen in the right part of Figure 4.4. The tangential distortion can be approximated by [47]

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x} + 2p_1\tilde{x}\tilde{y} + p_2(r^2 + 2\tilde{x}^2)) \\ \tilde{y} + p_1(r^2 + 2\tilde{y}^2) + 2p_2\tilde{x}\tilde{y}) \end{bmatrix} \tag{4.14}$$

where p_1 and p_2 are the tangential distortion parameters. By combining (4.13) and (4.15), the distorted image coordinates in the normalized image plane can be written as

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} \tilde{x} + \tilde{x}(1 + k_1 r^2 + k_2 r^4 + \dots) + 2p_1\tilde{x}\tilde{y} + p_2(r^2 + 2\tilde{x}^2)) \\ \tilde{y} + \tilde{y}(1 + k_1 r^2 + k_2 r^4 + \dots) + p_1(r^2 + 2\tilde{y}^2) + 2p_2\tilde{x}\tilde{y}) \end{bmatrix}. \tag{4.15}$$

And finally, the distorted pixel coordinates (u_d, v_d) can be obtained by multiplying the camera calibration matrix with the normalized distorted pixel coordinates such that

$$\begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix}. \tag{4.16}$$

This model specifies the forward projection from scene to image plane. Sometimes it is however desirable to solve the inverse problem, where we want to recover 3D points given its corresponding image coordinates. Unfortunately, no analytic solution to the inverse model exists, if distortion is taken into consideration. However, computer vision software such as Matlab and OpenCV provide functions for undistorting an image if the distortion parameters are known. That is, they provide methods to perform a nonlinear search to recover undistorted image coordinates from the distorted image coordinates [47].

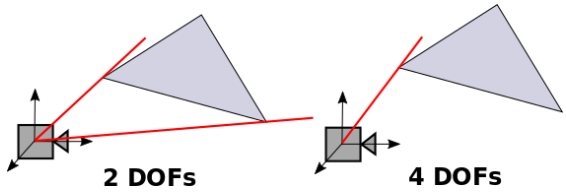


Figure 4.5: The Figure visualizes the P1P and P2P problem; the degrees of freedom (DOFs) in case of one or two camera rays, respectively. Each 2D-3D correspondance eliminates two DOFs.

4.2.3 Image plane to scene

If we revisit the inverse problem and for now, ignore the computer vision software solving it, the back-projection provides the transformation from pixel coordinates to the projective plane. In the scene, this corresponds to the camera ray. Thus, the 3D point along this camera ray needs to be located, and in order to do so, at least one of three scene coordinates has to be known.

To solve this problem with a single camera, the pose estimation algorithms require some point correspondences between the image plane and the scene. This is referred to as *model based pose estimation*. For this thesis, the AruCo marker is the model to be considered. Thus, an ArUco marker is used as a reference object to locate the vehicle with respect to the dockside (e.g. where the markers are located). The pose estimation is performed by moving the pose of the camera so that image distance between the projected point and the measured (e.g. detected) one is minimized. This is called the *reprojection error*. The problem of estimating the pose of a camera-based on n known points is called the *PnP problem*, which will be introduced in the next section.

4.3 Pose Estimation

Pose estimation is the process of computing the pose relative of an object with known structure, with respect to a calibrated perspective camera [48]. The pose of an object is defined as the position and orientation relative to some world coordinate system, usually the camera coordinate system. Usually, point or line correspondences between the object and the image are used to estimate the pose, given a calibrated camera with known intrinsic parameters [48].

Initially, the model has six degrees of freedom (DOF), three along x, y and z axis representing the position, and three about the corresponding axis representing the orientation. The goal of the pose estimation procedure is to reduce it to zero DOF. If we refer to Figure 4.5, the model is reduced to four DOF if one single point correspondence between object and image is available. From a geometric perspective, the P1P problem still provides three DOFs for orientation. In addition, the object can be moved exactly along the perspective projection, in both directions, meaning that the object has one single DOF for translation along this projection. However, since the object has been fixed by a pixel in the image, it cannot be moved in the horizontal or vertical direction, seen from the image plane. The P2P problem provides two DOFs, interestingly. From Figure 4.5, the first DOF is the easy

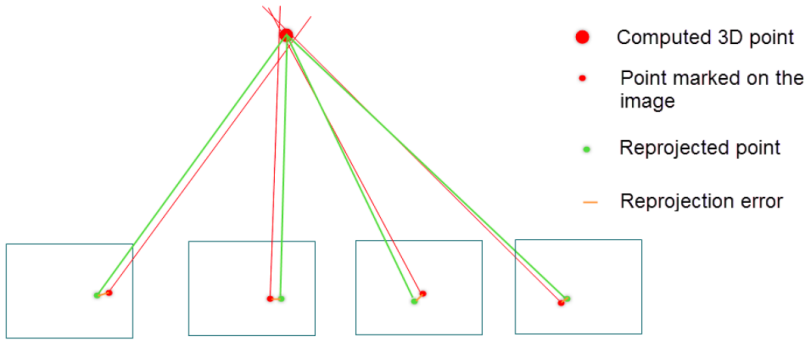


Figure 4.6: The Figure shows the reprojected 3D point on all images it appears together with the marked/detected point, and the resulting reprojection error in between.

to observe. If we rotate the object about the line segment between the edges where the camera rays intersect with the object, the point correspondences are obviously maintained. In addition, the object can be moved back and forth, such that the angle between the two camera rays stays fixed if some rotation is applied at the same time. With three point correspondences, the pose of the camera is reduced to zero DOF. Thus, each feature that is visible eliminates two DOFs. However, three point correspondences do not necessarily give an unique pose; it can actually give up to four solutions [49].

PnP

The Perspective-n-Point (PnP) problem refers to the pose estimation problem based on n known 2D-3D correspondences between image and object. The problem assumes the position of the points in the scene relative to the image plane is known. The problem itself boils down to minimizing the reprojection error. Thus, the position and orientation of the camera should be determined so that the image distance between the projected points and the detected points is minimized. A visual description is given in Figure 4.6.

In practice, small errors are introduced due to aspects such as sensor noise, image quantization, and manufacturing tolerances. This results in ambiguities and errors in the estimated pose [50]. Therefore, optimization techniques such as *bundle adjustment* may be applied so that feature point locations can be more accurately assigned before pose estimation is performed. Bundle adjustment can be defined as "the problem of simultaneously refining the 3D coordinates describing the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera(s) employed to acquire the images, according to an optimality criterion involving the corresponding image projections of all points" [51]. I.e. the images in Figure 4.6 can be used in a bundle adjustment scheme. The model may also include a robust estimation scheme, such as RANSAC (Random Sample Consensus), where each of the potential solutions is benchmarked against the other point matches to remove outliers [48], especially in cases where the points are influenced by a lot of disturbances.

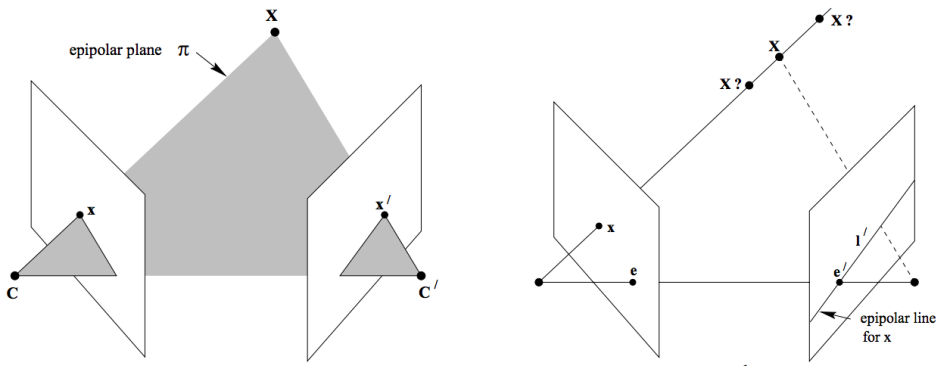


Figure 4.7: Point correspondence geometry. To the left, two camera centers are represented by $\boldsymbol{C}$ and $\boldsymbol{C}'$, with their corresponding image planes. The camera centres, the 3D point $\boldsymbol{X}$, and its image points $\boldsymbol{x}$ and $\boldsymbol{x}'$ all lie in a common plane $\boldsymbol{\pi}$. To the right, a camera ray produced by the camera center $\boldsymbol{C}$ and image point $\boldsymbol{x}$ are imaged as a line $\boldsymbol{l}'$ in the second camera view. The 3D point $\boldsymbol{X}$ must lie on this ray, meaning it must also lie on $\boldsymbol{l}'$. Image courtesy of [44].

The mathematical background on PnP solvers is in general quite complex, and we consider a derivation of the PnP problems as out of scope for this thesis. In addition, we did not find space for a detailed derivation of the calibration procedure.

4.4 Stereo-view Geometry

In this section, multiple-view geometry using stereo vision will be reviewed, much inspired by how humans recover depth from two eyes. That is, how the 3D position of an object can be reconstructed from a stereo vision setup. The theory in this section is largely based on [44]. If we revisit the problem of reconstructing a 3D point from a single camera view, it is evident that the back-projection only provides a camera ray where the 3D point could be placed arbitrarily along with this projection. Now, with an additional view of the scene, the 3D point can be recovered directly since the second camera ray (representing the same scene point) can be used to triangulate and reconstruct the 3D point in the scene. To be able to achieve this, some important questions need to be addressed. These can be summarized in the following.

- **The correspondence problem**: Given a detected 2D point in the first camera view, how do we find the corresponding 2D point in the second camera view?

- **The reconstruction problem**: Given a 2D stereo pair obtained from the correspondence problem, how do we recover the corresponding 3D point in the scene?

These two problems do in fact summarize the main procedure of a stereo reconstruction. Now, a detailed derivation of how 3D points can be recovered from stereo vision will be introduced.

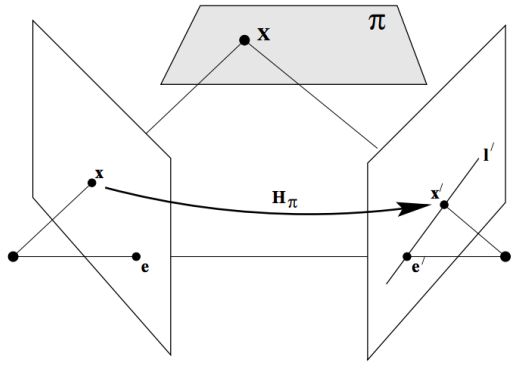


Figure 4.8: A point $\boldsymbol{x}$ in the left image is is transfered via a plane $\boldsymbol{\pi}$ to a corresponding point $\boldsymbol{x}'$ in the right image. The line segment joining epipole $\boldsymbol{e}'$ and x' is the epipole line $\boldsymbol{l}'$. The mapping from $\boldsymbol{x}$ to $\boldsymbol{x}'$ is denoted $\boldsymbol{H}_{\boldsymbol{\pi}}$. Image courtesy by [44].

4.4.1 Epipolar geometry

Epipolar geometry is usually motivated by considering the search for corresponding points in stereo matching [44]. It is the intrinsic projective geometry between two views, and is independent of the scene structure; it only depends on the cameras' intrinsic parameters and their pose relative to each other.

Consider a 3D point $\boldsymbol{X}$ imaged from two views, at $\boldsymbol{x}$ in the first image, and $\boldsymbol{x}'$ in the second image. Furthermore, consider two camera centres $\boldsymbol{C}$ and $\boldsymbol{C}'$ in 3D space, representing the two camera views. As three points are always coplaner, we define the plane $\boldsymbol{\pi}$ which includes the point $\boldsymbol{X}$, $\boldsymbol{C}$ and $\boldsymbol{C}'$. Clearly, $\boldsymbol{x}$ and $\boldsymbol{x}'$ also lies in the plane $\boldsymbol{\pi}$ as the camera rays (back-projected from $\boldsymbol{x}$ and $\boldsymbol{x}'$) intersects in $\boldsymbol{X}$. This can be seen in the left part of Figure 4.7, where the plane $\boldsymbol{\pi}$ is the grey epipolar plane.

Now, assume that we only know where $\boldsymbol{x}$ is located and the relative pose between the cameras. The first assumption results in a line segment corresponding to the camera ray of the left starting at $\boldsymbol{C}$, while the second assumption results in a line segment between the camera centers (e.g. the baseline), also starting in $\boldsymbol{C}$. Hence, the plane $\boldsymbol{\pi}$ is determined by the baseline and the camera ray. As already discussed, the corresponding point in the second image $\boldsymbol{x}'$ is assumed to lie in $\boldsymbol{\pi}$. Obviously, the $\boldsymbol{x}'$ also lies in the second image plane. Therefore, it is evident that $\boldsymbol{x}'$ lies on a line $\boldsymbol{l}'$ resulting from the intersection of $\boldsymbol{\pi}$ and the second image plane. We denote this line the *epipolar line* corresponding to $\boldsymbol{x}$ (see Figure 4.7). These findings clearly simplify the correspondence problem as the search is restricted to the line $\boldsymbol{l}'$ instead of the whole image.

The fundamental matrix

Before we move on, we will summarize the previous findings. That is, a compact algebraic representation of the mapping between a point and its epipolar line will be discovered.

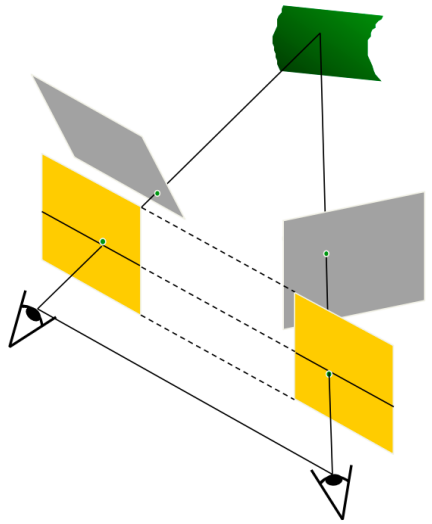


Figure 4.9: Stereo image rectification visualized. The grey original image planes are projected onto the yellow image planes to obtain horizontal scan lines of the images. Image courtesy by [52].

Thus, there is a map

$$\boldsymbol{x} \mapsto \boldsymbol{l}' \tag{4.17}$$

from a point in one image to its corresponding epipolar line in the other image. This projective transformation from points to lines is represented by the *fundamental matrix* $\boldsymbol{F}$.

In the first step, the point $\boldsymbol{x}$ is mapped to some point $\boldsymbol{x}'$ lying on the epipolar line $\boldsymbol{l}'$, by transferring via the plane $\boldsymbol{\pi}$. That is, $\boldsymbol{x}$ maps to $\boldsymbol{X}$ where the first camera ray meets the plane $\boldsymbol{\pi}$. Then, $\boldsymbol{X}$ is projected to a point $\boldsymbol{x}'$ in the second image. In the second step, the epipolar line $\boldsymbol{l}'$ is obtained as the line joining $\boldsymbol{x}'$ to epipole $\boldsymbol{e}'$. Both steps can be seen from Figure 4.8. The epipolar line can be defined as $\boldsymbol{l}' = \boldsymbol{e}'_x \times \boldsymbol{x}' = [\boldsymbol{e}']_\times \boldsymbol{H}_{\boldsymbol{\pi}} \boldsymbol{x}'$, where $[\boldsymbol{e}']_\times$ is a 3×3 skew-symmetric matrix. Further, there is a 2D homography $\boldsymbol{H}_{\boldsymbol{\pi}}$ mapping $\boldsymbol{x}$ to $\boldsymbol{x}^{'}$ such that $\boldsymbol{x}' = \boldsymbol{H}_{\boldsymbol{\pi}} \boldsymbol{x}$. By this, the epipole line can be written as

$$\boldsymbol{l}' = [\boldsymbol{e}']_\times \boldsymbol{H}_{\boldsymbol{\pi}} \boldsymbol{x} = \boldsymbol{F} \boldsymbol{x} \tag{4.18}$$

where

$$\boldsymbol{F} = [\boldsymbol{e}']_\times \boldsymbol{H}_{\boldsymbol{\pi}} \tag{4.19}$$

is the fundamental matrix.

4.4.2 Rectification for stereo vision

Although the search for corresponding points is limited to a search along the epipolar line $\boldsymbol{l}'$, there is still room for improvement. If the stereo images are parallel with respect to each other and the baseline, the epipolar lines fall along with the horizontal scan of the images, which simplifies the search even more. The procedure to re-project image planes onto a common plane parallel to the line between optical centers is referred to as *stereo image rectification*.

Some matrix manipulation is needed to re-project the image planes onto a common parallel plane. First, consider a point in the image $\boldsymbol{x} = \boldsymbol{PX}$, where $\boldsymbol{P} = \boldsymbol{K}[\mathcal{R} \;\; \boldsymbol{t}]$ (defined in (4.12)). Then, assume the calibration matrix $\boldsymbol{K}$ is known, and we allow ourselves to remove the effect of the known calibration matrix by defining a new point

$$\hat{\boldsymbol{x}} = \boldsymbol{K}^{-1}\boldsymbol{x}. \tag{4.20}$$

Then, by substituting $\hat{\boldsymbol{x}}$ into $\boldsymbol{x} = \boldsymbol{K}[\mathcal{R} \;\; \boldsymbol{t}]\boldsymbol{X}$, we obtain

$$\hat{\boldsymbol{x}} = [\mathcal{R} \;\; \boldsymbol{t}]\boldsymbol{X}. \tag{4.21}$$

Here, $\boldsymbol{P}_n = \boldsymbol{K}^{-1}\boldsymbol{P} = [\mathcal{R} \;\; \boldsymbol{t}]$ is called a *normalized camera matrix*. Now, consider a pair of normalized camera matrices, $\boldsymbol{P}_n = [\boldsymbol{I} \;\; \boldsymbol{0}]$ and $\boldsymbol{P}_n^{'} = [\mathcal{R} \;\; \boldsymbol{t}]$. Thus, the camera matrices of a calibrated stereo system with the world origin located at the first camera. The fundamental matrix corresponding to this pair of normalized cameras is denoted the *essential matrix*.

With the given assumptions, the essential matrix from a geometric perspective can be derived. Recall that the epipolar plane $\boldsymbol{\pi}$ is a plane containing the baseline, and intersects the image planes in corresponding epipolar lines $\boldsymbol{l}$ and $\boldsymbol{l}'$. Hence, the epipolar plane can be spanned out by two 3D-vectors, $\boldsymbol{t}$ and $\mathcal{R}\hat{\boldsymbol{x}}'$, representing the translation vector between the camera centers and the camera ray passing through $\hat{\boldsymbol{x}'}$, respectively. Consequently, the cross product $\boldsymbol{t} \times \mathcal{R}\hat{\boldsymbol{x}}'$ is a 3D-vector perpendicular to the epipolar plane. Since $\hat{\boldsymbol{x}}^T$ is perpendicular to the obtained cross product, it follows that

$$\hat{\boldsymbol{x}}^T \cdot [\boldsymbol{t} \times (\mathcal{R}\hat{\boldsymbol{x}}')] = \hat{\boldsymbol{x}}^T \cdot [[\boldsymbol{t}]_\times \mathcal{R}]\hat{\boldsymbol{x}}' = 0 \tag{4.22}$$

where

$$\boldsymbol{E} = [\boldsymbol{t}]_\times \mathcal{R} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t_x \\ 0 & t_x & 0 \end{bmatrix}. \tag{4.23}$$

is the essential matrix. For a more comprehensive algebraic derivation, we refer to equation (9.2) in [44]. Equation (4.23) shows that the essential matrix $\boldsymbol{E}$ maps corresponding points in rectified stereo images by a horizontal shift t_x. Simply said, both image planes are transformed such that they are parallel with respect to each other. Hence, both z and y-coordinates of corresponding points are the same, and the scan can purely be

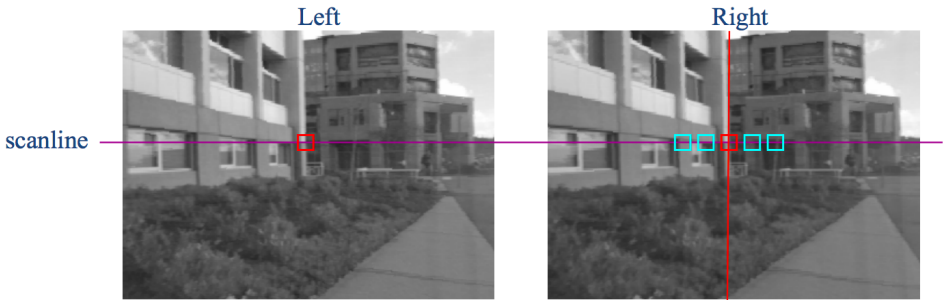


Figure 4.10: Stereo matching performed by sliding a window along the horizontal image axis to find the corresponding point in the right image. Image courtesy by [52].

performed along the horizontal image axis. Note that the essential matrix $\boldsymbol{E}$ is a special case of the fundamental matrix $\boldsymbol{F}$.

4.4.3 Correspondence problem

With the epipolar lines transformed into scanlines, the next step is to find stereo correspondences in a stereo pair. In case of a depth map, each pixel in the first image should have a corresponding match in the second image. We will now look into aspects relevant for the matching strategy.

Since sophisticated stereo matching strategies are not applied directly in the thesis, algorithm-specific details will not be given. The reason is simply that OpenCV provides a specialized ArUco marker detector, which provides the same order of detected corner points every time it is performed. Hence, the corner points were matched directly by their index in the implementation. We will however discuss some characteristics relevant in the stereo matching scheme, structured in bullet points.

- **Similarity constraint:** Intuitively, corresponding regions in a stereo pair should be similar in appearance, and non-corresponding regions should be different. Thus, the order of objects in the scene is maintained from one camera view to the other (assuming the objects have approximately the same depth). There are however several limitations of the similarity constraint. For instance, textureless surfaces will indeed produce repetitive regions of points, which makes the search along the scanline harder. Occlusions also make it harder to recover 3D points from a stereo camera as the scene may have some object blocking for another object of interest.
- **Window size:** From Figure 4.10, observe the sliding window along the scanline. In this context, the size of the search window introduces some effects. A smaller window will produce a more detailed disparity map, but also noisier. On the other hand, a larger window produces smoother disparity maps, but also less detailed. A disparity map refers to the pixel displacements between a pair of stereo images.
- **Baseline:** The baseline also plays an important role. In the next section, we will discover that the baseline is inversely proportional to the depth. Intuitively, if the

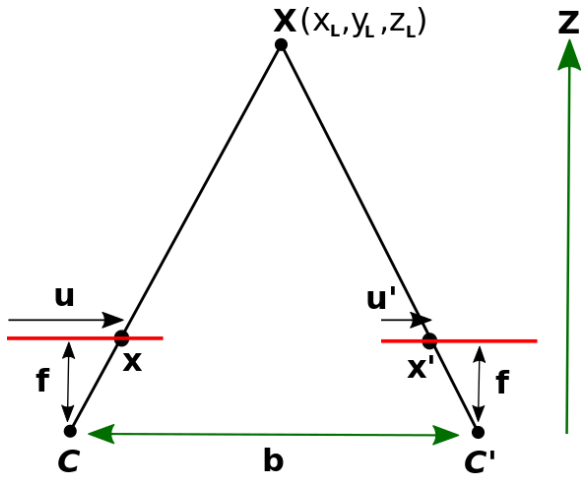


Figure 4.11: The 3D reconstruction problem illustrated from the x-z plane, where the goal is to recover the scene point $\boldsymbol{X}$ from the stereo pair $(\boldsymbol{x}, \boldsymbol{x}')$.

baseline is small and the depth (between the cameras and the object in the scene) is relatively larger, the disparity can get very small. As the depth increases, the stereo camera will slowly, but surely converge towards a monocular camera. Thus, a small baseline makes the stereo system "near-sighted" suitable for close-range detections, but lack the ability to recover depth accurately from large distances. On the other hand, a large baseline reduces the depth error for larger distances. The drawback is however that fewer scene points are visible in both camera views, which makes the matching strategy harder.

4.4.4 Reconstruction problem

Assuming the correspondence problem is solved, giving us a stereo point correspondence, the last step involves reconstructing the corresponding scene point. This step can be summarized in Figure 4.11, where the goal is to reconstruct a scene point $\boldsymbol{X}$ from the stereo pair $(\boldsymbol{x}, \boldsymbol{x}')$ with pixel displacement u and u' along the horizontal image axis. The red lines in Figure 4.11 illustrates the parallel image planes, assuming the stereo setup is rectified.

In order to recover the depth z_L, some geometric derivation is needed. First, consider the point $\boldsymbol{X}$ with coordinates (x_L, y_L, z_L) relative to the left camera origin $\boldsymbol{C}$. Further, denote u and u' the horizontal component of the image points, for left and right camera view, respectively. Similarly, v and v' represents the vertical component of the image points. u and v are usually defined relative to the upper left corner of the 2D image. Both cameras have the same focal length distance f between the camera origins and the image planes along the z-axis. By using similar triangles, the following relationship can be obtained

$$u = f\left(\frac{x_L}{z_L}\right). \tag{4.24}$$

By introducing $\boldsymbol{X}$ with coordinates (x_R, y_R, z_R) relative to the right camera, a similar relationship can be obtained such that

$$u' = f\left(\frac{x_R}{z_L}\right). \tag{4.25}$$

It is evident that the baseline length b relates the two camera centers such that $x_R = x_L + b$. Further, the disparity is defined as $d = u' - u$, a pixel shift along the horizontal image axis. From this, together with (4.24) and (4.25), it follows that

$$d = u' - u = f\left(\frac{x_R}{z_L}\right) - f\left(\frac{x_L}{z_L}\right) = f\left(\frac{x_L + b - x_L}{z_L}\right) = f\frac{b}{z_L}. \tag{4.26}$$

By rearranging (4.26), z_L is obtained, and interestingly inversely proportional to disparity d. x_L and y_L can be obtained by the same principles (e.g. similar triangles), such that the final 3D reconstruction can be obtained from

$$\boldsymbol{X} = (x_L, y_L, z_L) = \left(u\frac{z_L}{f}, v\frac{z_L}{f}, f\frac{b}{d}\right). \tag{4.27}$$

Here, we assume the baseline b, the disparity d and the focal length f to be known such that z_L can be recovered. In addition, the image point $\boldsymbol{x} = (u, v)$ is assumed to be known. By this, both x_L and y_L are obtained naturally from z_L. The more general case for non-parallel cameras will not be investigated as it is not used directly in this thesis.

Chapter 5

Hardware and Software Choices

This Chapter represents the software and hardware components chosen for the master thesis.

5.1 Hardware Setup

Overall design

We start this chapter with an overview of the hardware components chosen for the overall system which will be used for the final experiments. As seen in Figure 5.1, there are some components involved in the vision system. This includes

- A stereo vision system consisting of a camera and lens pair, in addition to GPIO cables for synchronization between the cameras.
- PoE interfacing for fast and reliable data and power transmission in one cable for each camera.
- DC/DC converters for power interfacing between the on-board battery system and the hardware components.
- An embedded PC for fast computations on-board.
- A waterproofed camera rig for protection of the hardware components under docking operations.
- A lidar for verification of camera measurements.

From the beginning, the intention has been to build a decoupled system that can easily be mounted and configured to an arbitrary USV with a minimal number of cables. For the system to work, only two cables are required. This makes it easy to unplug the system from an USV and interface with a power supply and an ethernet cable for testing at the

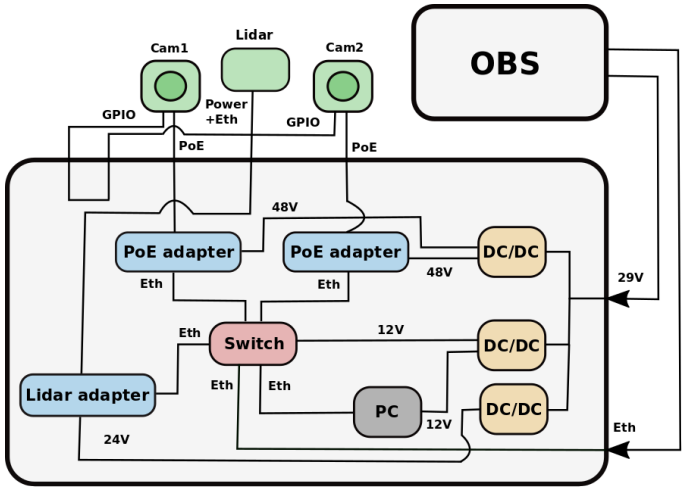


Figure 5.1: The overall design of the hardware components involved in the vision system. It also shows the power and ethernet interface with On-Board System (OBS).

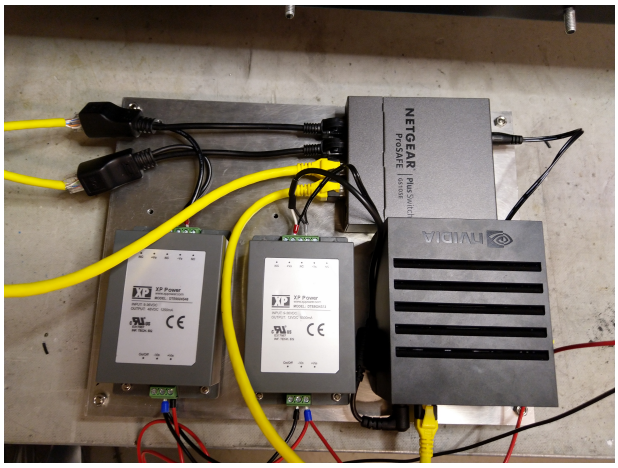


Figure 5.2: The actual hardware components (inside the box) that were used in the master thesis. In addition, a lidar adapter and another DC/DC converter were placed inside the box in the last phase of the work for verification of the camera measurements.

Figure 5.3: To the left, a FLIR Blackfly S GigE PoE machine vision camera. To the right, a wide angle lens from Edmund optics. Both are used throughout the master thesis.

office as well. A sketch of the overall hardware design and the actual system used for the experiments are shown in Figure 5.1 and Figure 5.2, respectively.

To get an overview of the hardware design, it is natural to start with the interface between On-Board System (OBS) and the proposed vision system. OBS provides the hardware components with power and communication via ethernet. Also, note that OBS itself has a battery power supply. If we look at the components inside the vision box, three DC/DC-converters are used since the different devices require different voltage levels (see Figure 5.1). Since the cameras apply PoE (Power over Ethernet), only one cable is needed to power and transfer data for each camera. This reduces the number of cables and hardware complexity. The GPIO (General Purpose Input Output) cables which connect the left and right camera are used for camera synchronization. The ethernet switch is also used to interface the embedded computer with the cameras and the lidar, enabling the computer to receive sensor data with high bandwidth.

A brief description of each component will be given with reasoning on why they were chosen.

Camera system

The heart of the vision system is the camera itself. For this project, a lightweight and cost-effective GigE PoE camera (see Figure 5.3) made for different industrial machine vision applications is chosen. PoE cameras have been a well-known standard for industry cameras for a long time. PoE capabilities allow for effectively combining data transfer and power supply in one cable. It uses global shutter CMOS (Complementary Metal Oxide Semiconductor) as its digital imaging sensor which delivers clear and (nearly) distortion-free images at high speed. Further, wide-angle lenses are chosen for identification of landmarks more often. The lenses from Edmund Optics provides a fixed 3.5 mm focal length with wide FOV (Field of View) and low distortion (less than 0.4 %). For more details about the specification, see table 5.2.

In contrast to lenses with fixed focal lengths, there also exist "all-in-one lenses" with optical zoom (e.g. multiple focal lengths). These lenses allow a photographer to take al-

Table 5.1: Camera Specifications for the master thesis.

Camera Specifications	
Camera model	BFS-PGE-13Y3C-C
Resolution	1280×1024
Maximum FPS	84
Sensor type	CMOS
Sensor format	1/2"
Interface	GiGE PoE
Dimensions	$29 \times 29 \times 30$ mm
Operating temperature	0-50 $^{\circ}$

most any type of image (portrait, landscape, telescope etc). However, there are some limits to these lenses. Even though image quality can be good, they are generally not as sharp as other lenses across focal lengths. In addition, to keep size, weight, and costs down, all-in-one lenses are unlikely to be very fast (e.g. they won't have a large aperture). On the other side, a prime lens has only one focal length, which can be wide-angle, telephoto or normal. Those who choose prime lenses are looking for either ultra-high image quality or very fast f-stops. Now, there are some modest prime lenses, such as a 50mm f/1.8, 35mm f/2.8, or 135mm f/3.5. These lenses are smaller, lighter, and not nearly as expensive as the premium prime lenses, and yet can still deliver high-quality images [53]. Therefore, to achieve high-quality images with fast f-stops at a reasonable price that fits our application, prime lenses were chosen for the experiments.

Ideally, the camera sensor format should match the lens for utilization of the whole camera view. If the camera sensor format is bigger than the maximum sensor format provided by the lens, the phenomena of vignetting occurs. That is, the lens is not able to cover the whole camera view and consequently, black shadows appear in the corners. Also note that with a lower sensor format, one needs to compensate with a shorter focal length to achieve the same FOV. This gives more distortion and a camera suitable for short-range detections. Depending on your machine vision tasks and specifications, it is important to be aware of the size of the sensor format when picking a camera/lens combination.

At last, it is desirable to synchronize the left and right camera. Normally, the cameras are triggered via software or hardware to achieve synchronization. Since we want to achieve stereo vision, it is desirable to minimize the latency between when the left and right image were taken to achieve multi-view of the scene at the same time. Therefore, hardware triggering with GPIO cables is applied as it provides considerable lower latency between left and right image compared to software triggering. E.g. the GPIO connectors provide a more reliable synchronization. In our case, it means that the right camera is triggered by the left camera in a master-slave setup (see Figure 5.4).

Table 5.2: Lens specifications for the master thesis.

Lens Specifications	
Focal length	3.5 mm (fixed)
Aperture	f/2.4
Field of View	82.4 °
Distortion	< 0.4 %
Mount	C-Mount
Working distance	100 mm - ∞
Maximum sensor format	1/2"
Length	38.2 mm

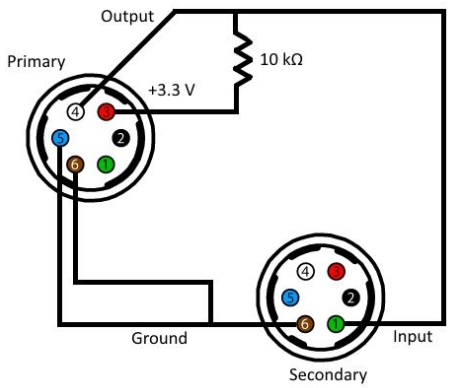


Figure 5.4: A master-slave setup between the left and right camera to achieve hardware triggering. In our implementation, the left camera is the primary (the master) and the right camera is the secondary (the slave). Image courtesy by [54].

Power management and data transmission

This section covers how the different devices are powered and how the sensor data is transmitted. DC-to-DC converter is an electromechanical device that converts a source of direct current (DC) from one voltage level to another. Since the battery provides power with 29V and none of the hardware components accept this voltage level, DC/DC-converters can be used to meet the required input voltage range for each component. Both the Nvidia Jetson Xavier and the ethernet switch accept 12V input, while the two cameras can be powered with 48V PoE. At last, the lidar adapter accepts input 22-26V. Therefore, three separate DC/DC-converters with output 12V, 24V and 48V are used to fulfill the power requirements for each component. All the DC/DC-converters accept input voltages between 9V and 36V to ensure that they are suitable for the battery (which outputs 29V).

The chosen cameras apply Power over Ethernet (PoE) adapters which allows for fast and reliable power and data transmission. In addition, they use the well-known GigE (Gigabit Ethernet) vision standard which is popular for industrial machine vision tasks due to its high bandwidth capabilities. That being said, the switch and the computing system should match with highspeed I/O to take full advantage of the camera streams. The switch supports up to 1000 Mbps of data transmission for each port, which is more than enough for our application. Note that the lidar uses the GigE standard as well.

Computing system

As the experiment is intended to be done on-board a small, flexible USV, it is desirable to use an embedded PC with a small form-factor. In addition, the use of Deep CNNs for object detection with low inference time requires a GPU-computing unit with great performance. To meet these criteria, it was natural to look for what Nvidia could offer as they are world-leading within the market of embedded edge-computing units. After some research, we found the Nvidia Jetson Xavier (Developer Kit) to be a great candidate for our application.

With its 105×105 mm dimensions, it is indeed a compact computer (it can be recognized in the lower right corner in Figure 5.2). Depending on the power mode, it can be adjusted between 10W and 30W. This allows for lower chances of overheating inside the water-proofed camera box (where the Nvidia computer is located).

The heart of the Jetson Xavier is its GPU. Mostly because GPUs have shown amazing parallel computing capabilities and is therefore a natural choice for heavy, AI-computing applications. It delivers up to an unparalleled 32 TeraOPS (TOPS) of peak compute. To be able to utilize full performance, it has to be set in "MAXN" power mode (corresponding to 30W). It is ideal for running deep neural networks. It can even be used to train deep CNNs although it is, in general, recommended to train deep neural networks with a powerful desktop. Since the cameras and the lidar may produce a significantly amount of data, it is desirable with a computer to handle this seamlessly. Fortunately, Jetson Xavier delivers up to 750 Gbps of high-speed I/O, which ease the burden of handling huge amounts of data in operations with strict real-time requirements.

Table 5.3: Specifications for the Nvidia Jetson Xavier (Developer Kit).

Nvidia Jetson Xavier specifications	
GPU	512-core Volta GPU with Tensor Cores
CPU	8-core ARM v8.2 64 bit CPU
Memory	16GB 256-Bit LPDDR4x
Storage	32GB eMMC 5.1
Size	105 mm $\times$ 105 mm

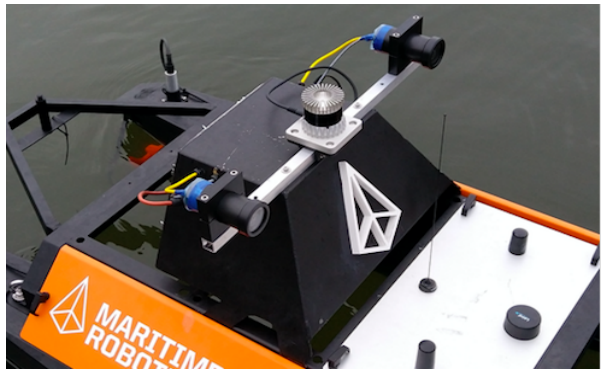


Figure 5.5: The vision box in action on top of an Otter USV. The lidar is placed in between the stereo camera setup. All sensors are fixed to the same bracket.

The physical camera rig

The physical camera rig represents the mechanical components to house and mount the electrical components. On top, the cameras are mounted on a bracket to ensure they are fixed relative to the USV and the bracket itself (see Figure 5.5). This is essential for maintaining the extrinsic camera parameters and perform stereo vision. The fixed baseline between the cameras is approximately 62 cm. Hopefully, a considerable high baseline will provide great stereo vision measurements for the USV at middle ranges and not only for the last couple of meters of the docking operation. Since the camera rig is mounted at the rearmost part of the USV, we can relax the minimum working distance for our stereo vision system. In addition, at very short ranges (e.g. 0.5-1.5 m from the dockside), we can apply monocular pose estimation with approximately the same accuracy. At last, each camera is housed to make sure they are water-proofed during the experiment.

Lidar

To be able to verify the camera measurements, some sensor is needed for ground truth comparison. Different options such as GPS, radar and lidar can be used. In the first place, the intention was to use a RTK GPS for ground truth comparison. But there are some challenges involved when comparing sensor measurements between the camera and the GPS directly. First of all, the GPS provides absolute position and orientation measurements, while the camera system produces a position and orientation relative to some reference

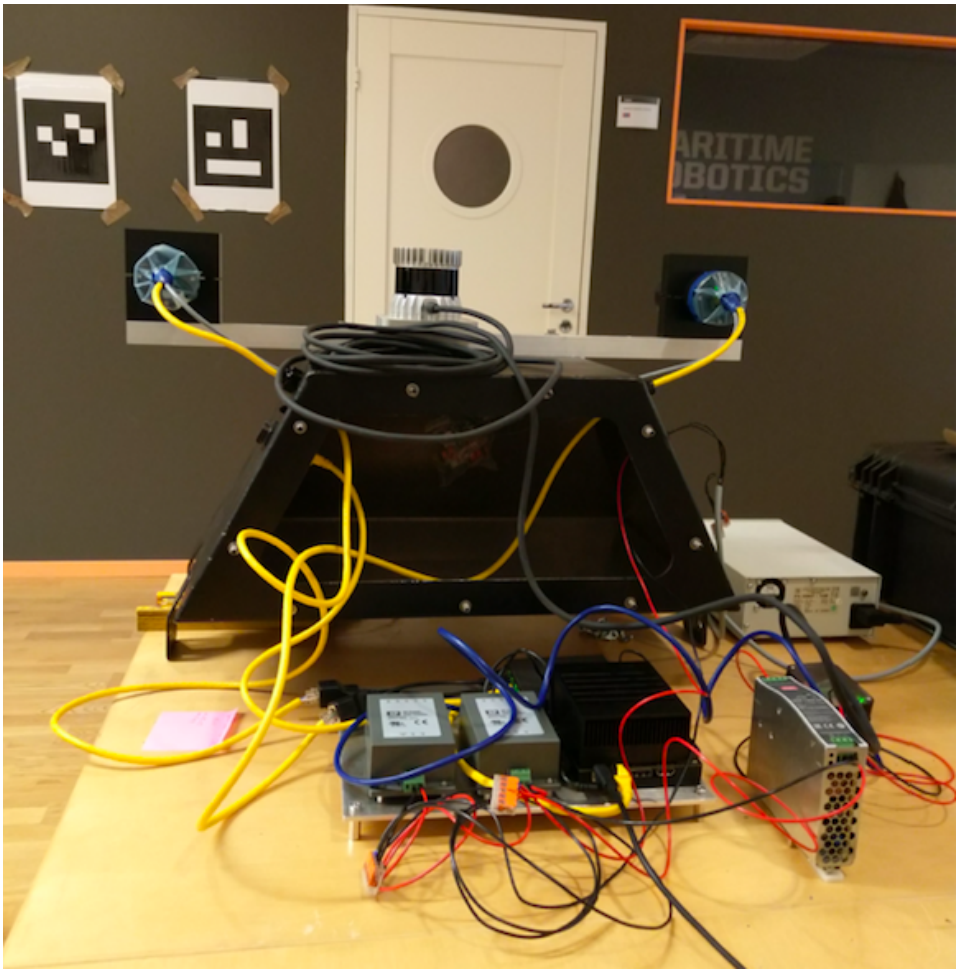


Figure 5.6: Behind the scenes, a visual shot of the self-developed camera rig during the test phase. See Figure 5.1 for the corresponding sketch of the hardware components.

Table 5.4: Lidar specifications for the experiment with focus on optical performance. For more information about the specifications, see here [55].

Lidar Specifications	
Model name	Ouster OS1
Range	0.5-120 m
Range resolution	1.2 cm
Maximum vertical resolution	64 beams
Maximum horizonal resolution	2048
Horizontal Field of View	360°
Vertical Field of View	+16.6° to -16.6°
Rotation rate	10 to 20 hz

point (i.e. the midpoint of the markers). That means the static reference points need to be measured in a global coordinate system, by the GPS, in order to compare the localization measurements between the cameras and the GPS directly. Since the main motivation of the master thesis is to compare how well the cameras measure the relative position, between the camera and the marker, a lidar was chosen instead for verification and comparison. A lidar produces a map of the surroundings nearby and where the lidar is located relative to these surroundings. Since the lidar provides a relative distance between itself and objects it hits nearby, it can be used to compare distance measurements between the camera system and the lidar directly. In addition, lidar is widely supported in the ROS community, which makes the lidar an ideal choice for synchronization and visualization of the sensor measurements.

Figure 5.6 summarize it all; the camera rig mounted on top of the protecting box and all necessary hardware to perform the final experiments.

5.2 Software Choices

This section provides a wide range of software environments used in the project. Some of them are related to deep learning frameworks while others are used for more basic computer vision tasks like image acquisition, image processing and camera calibration. This section is more or less picked from my own project thesis.

OpenCV

OpenCV is an open-source computer vision and machine learning software library. It contains more than 2500 optimized algorithms including both classic and state-of-the-art computer vision and machine learning algorithms. The library has been used frequently through the project and has been especially helpful when designing 3D reconstruction algorithms. OpenCV offers most of its algorithms in both Python and C++.

Matlab

Matlab is one of the most used mathematical software among engineers around the world and provides toolboxes for a wide range of applications. One relevant toolbox used for this thesis is the stereo camera calibrator. It has also been used for visualization of my results.

ROS

Robot Operating System (ROS) is a flexible framework for writing robot software. It is open-source and provides tools for visualization, monitoring and simulation. Further, it simplifies communication and data transfer across multiple systems. Different sub-modules communicate via messages, they receive data and output processed information. For instance, ROS has been used to send data between ROS nodes running simultaneously in this project. ROS also offers bridging between openCV and ROS. And Blackfly S cameras is widely supported by ROS. For this project, YOLOv3 has been integrated into the ROS environment to ensure that bounding box information can be used to produce 3D information.

CUDA

When training and testing deep neural networks, it is desirable to utilize parallel processing to speed up the pipeline. In this context, the Graphical Processing Unit (GPU) has shown amazing parallel computing capabilities and is therefore a natural choice for deep learning applications, which may have hard real-time requirements. In order to utilize the power of parallel processing on a GPU, some software is needed to enable it. CUDA is one such platform developed by Nvidia which allows for thousands of GPU cores to run in parallel and hence, it reduces the training time significantly. CUDA is a prerequisite for running the deep learning model YOLOv3 in (hard) real-time. It supports languages like Python, C/C++ and fortran. Fortunately, CUDA is pre-installed on the Nvidia Jetson Xavier used for this project.

Darknet

Darknet is a software environment for the deep learning object detection model YOLOv3. Briefly said, it contains all modules you need to run YOLOv3. The software is mainly written in C and C++ which allows for effective computing and low runtime (compared to languages like Python). Both openCV and CUDA must be installed and enabled in order to run Darknet in real-time.

Spinnaker SDK

Spinnaker is a Software Development Kit (SDK) for FLIR blackfly cameras. For this project, it was used to configure and test the cameras.

Chapter 6

Data Acquisition

In order to explain the methods explored in the project, it is natural to start with the datasets. The datasets provide a foundation for data-driven learning methods in the computer vision field. This Chapter focuses on the various data sets as well as the construction of a dataset designed for auto-docking operations. The custom dataset is indeed an important step for the preparation of the experiments. With some adaptations, this chapter is mainly based on my project thesis.

6.1 ImageNet

It may be very time consuming to train a CNN from scratch. Therefore, it is common to use a pre-trained model. Objects in general have a lot of low-level features (like corners and edges) in common and training on a big, general dataset lets the network learn these general patterns rather than only be trained on a small dataset with very specific features. This makes the model more general and ensures that it is not overfitted in the first place. For this experiment, YOLOv3 is pretrained on ImageNet which contains 1.2 million images with approximately 1000 different categories. Further, transfer learning is applied by domain-specific fine-tuning on a smaller dataset with different docking scenarios. This custom dataset is reviewed in the next section.

Table 6.1: Camera specifications for the data collection.

Camera settings			
Camera	Resolution	Lens type	Pixel format
FLIR blackfly S	1280 x 1024	Wide FOV	monochrome

Table 6.2: Split between training, validation and test images.

Training, validation and test data			
Dataset	Training	Validation	Test
Custom	683	85	85

6.2 Collecting Data

On November 20th, 2019, data collection was conducted to gather camera data of different markers on a quay at Trondheim Harbour. The experiment consisted of navigating a small USV around the harbor and slowly approaching the dockside from different angles. The camera system, explored in Chapter 5, was mounted at the back of the USV and records where taken with settings as shown in Table 6.1 using another laptop to remotely control it with SSH (Secure Shell).

Obviously, only relevant examples (i.e. images showing at least one marker) from the records were included in the custom dataset. Sampling of the relevant videos resulted in a custom dataset consisting of 853 images. Sampling methods effectively remove redundancy in the dataset as two frames in a row (i.e. a video sequence) share almost the same patterns. This allows for effectively cover more variations in the dataset without spending weeks for labeling images. In addition, YOLOv3 performs on-line data augmentation which improves existing training data.

6.2.1 Train, validation and test data

The custom dataset was randomly shuffled and then, 80 % of it was assigned to training, 10 % was assigned to validation and 10 % was assigned to test data. Table 6.2 shows how many images each set contains. By this, we ensure that the training, test and validation set is independent, which is essential for evaluating the accuracy of the trained model on unseen data. It also ensures that the model is trained on a sufficiently amount of data, while some are left for validation. Based on the validation, we can pick the most optimal weights for testing. At last, some test data were used for verification of how accurate the model performs on unseen data.

The dataset is considered quite small, which sets a limit on how general the model can be. However, a comprehensive custom dataset for ArUco marker detection may not be necessary. Recall that YOLOv3 performs data augmentation online to produce synthetic data from the custom dataset, and thereby improving training. The most significant improvement may therefore be to increase the variations of how the markers look like from different views and orientations as well as under different environmental conditions.

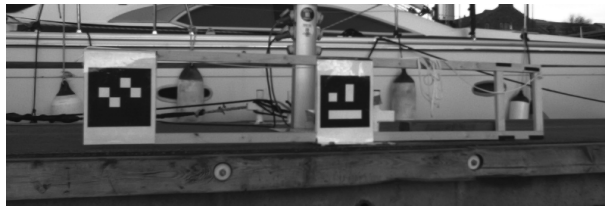


Figure 6.1: Each marker has its own dedicated class. The left marker is named *m1*, while the right is named *m2*.

6.2.2 Classes and Marker configuration

Marker configuration

To make things simple, only one marker configuration is reviewed in this master thesis. In my project thesis, different marker configurations were explored. Some of the findings from the experiments in the project thesis were that simple "black/white" ArUco markers with sufficient size (A3 size paper) lead to high detection quality. Therefore, we will solely focus on a simple marker configuration consisting of two different ArUco markers. The main motivation is to show that the camera system can handle multiple markers in the scene without making things overcomplicated (e.g. the more markers in the scene, the more cases need to be handled by the system). There are several benefits when applying multiple markers. One is obviously that different markers can be placed around the dockside and thereby extending the perception level for the camera system. Recall that no localization measurements are produced without a marker in the camera view. Another reason is that multiple markers in the scene at once may produce more trustworthy measurements (e.g. redundancy).

Another aspect is where to place the marker configuration. Since we assume that the markers have a well known fixed position it is important that the markers are not placed on a floating quay but rather on a fixed dockside (i.e. fixed to the land).

Classes

The custom dataset consists of two classes, where both of them are associated with different ArUco markers. The classes try to sort these markers by their patterns. Figure 6.1 shows the two classes, where the left marker is defined with the class name *m1* and the right marker is defined with the class name *m2* (simply corresponds to marker1 and marker2). As the reader may observe, only one object is included for each class and therefore there are quite small variations within each class. Also, both have the same color and fixed size. Table 6.3 shows that there is a balance between the two classes (i.e. how often they occur in the custom dataset).

The reader may wonder why not ArUco markers with different patterns stay within the same class (as they all could have been defined as a class "markers"). With our definition of classes, we are able to distinguish between the markers which can simplify the pipeline

Table 6.3: Number of ground truth labels for each class in the part of the custom dataset used for training, validation and testing.

Classes in custom dataset		
Class name	Marker type	ground truth labels
m1	ArUco marker	829
m2	ArUco marker	835

of the whole working system. Normally, a real-time pose estimation system with ArUco markers needs some sort of tracking capability to know that the detected markers between consecutive frames in fact corresponds. This is usually done by assigning an object id for each object in a frame and look for small pixel displacements between consecutive frames in order to the update object id for each object.

However, in our case, only one marker for each class shows up in our marker configuration. By this assumption, one can simplify the tracking problem and assume that the class number corresponds to the object id.

A remark about the dataset

The final custom dataset serves some comments. Normally, we would collect data for several days to cover different weather and light conditions. This would make our custom dataset less homogeneous and thereby more robust with respect to environmental changes that can affect the visual scene. For this thesis, constructing a dataset reflecting extreme environmental changes has not been prioritized as it requires considerably more testing days to cover more weather and light conditions. The priority has solely been on building a prototype that outputs accurate and meaningful measurements in normal outdoor conditions. We consider it "the next step" to build a vision system that can handle more extreme environmental changes and it would not make sense to prioritize this challenge before the vision system is proved working satisfactorily under normal conditions. Based on my project report, we found the trained CNN to produce considerable high detection quality, although the training data was quite homogeneous (e.g. only records from the same dockside for some hours). Therefore, we decided to only spend a day for data collection.

Chapter 7

Implementation

This Chapter focuses on the implementation of the camera system and reflects the main contribution. It is separated into three modules. The first module investigates how the cameras are integrated into the perception pipeline. Then the object detection model is reviewed. That is, how the datasets are prepared and how the model can be trained, validated and tested within the framework. At last, the 3D reconstruction pipeline will be reviewed. E.g. how we can extract 3D information from bounding box predictions. We start this chapter with an overview of the whole perception pipeline to get some context of how the work contributes towards a robust, real-time localization system. Sections 7.1 and 7.3 are mainly based on my project thesis, with some adaptions to fit into the master thesis.

7.1 Pipeline Overview

This section aims to introduce how the whole perception pipeline works. The final working system employs a complementary part-based approach that uses a combination of data-

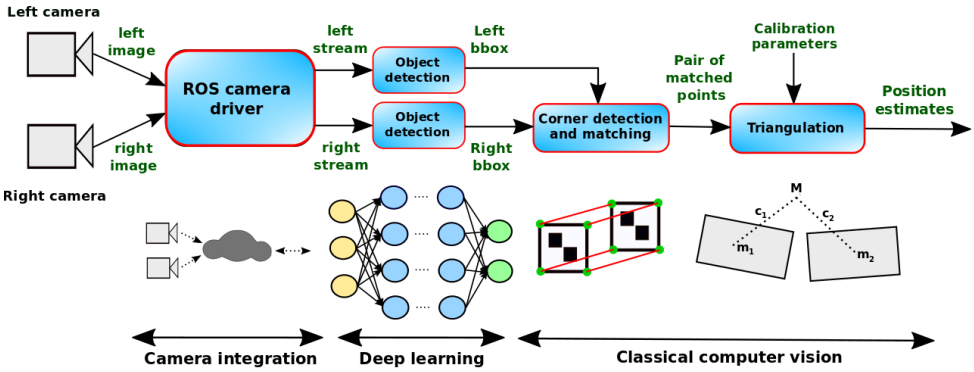
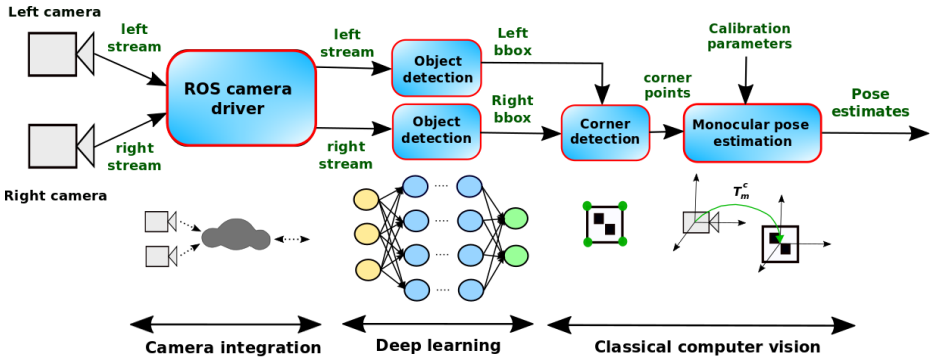


Figure 7.1: Proposed vision-based navigation system for the overall project with stereo vision.



Proposed vision-based navigation system for the overall project with monocular vision.

Figure 7.2:]

driven deep learning methods, utilizing data wherever required, and at the same time using traditional computer vision techniques when the scope and complexity of the task are reduced. These design choices were chosen to make the pipeline more intuitive and easier to debug compared to an end-to-end deep learning pipeline.

Two implementations are proposed, as shown in Figures 7.1 and 7.2. The main difference between them is that one utilizes stereo vision to achieve localization estimates, while the other technique relies on one camera only (e.g. monocular vision) to achieve localization estimates. Implementation details and differences will be reviewed in detail in Section 7.4. If we disregard the differences, the high-level design and functionality is more or less the same. In this section, the motivation is to introduce the reader to the main idea and therefore, we will only explain it with focus on one of the methods. That is, the stereo vision design as shown in Figure 7.1. It consists of three pipeline-based modules (as shown in Figure 7.1) which takes raw images as input and produces position estimates as output. The object detection model is deep learning based, while the ROS camera driver and the stereo matching/triangulation modules belong to the classical computer vision field. A final working system assumes the deep learning based model is trained for its application and that the stereo vision system is calibrated.

In the following, we will explain how it works. The camera driver captures frames from the left and right camera and outputs streams for the object detection model to read and process. Since the camera driver and the object detection model are implemented in ROS, they can easily communicate and interchange data. This way, the object detection model is able to receive a sequence of frames from the left and the right camera. Note that we run two instances of the object detection model in parallel to produce two bounding boxes from different angles at the same time. Now, the trained object detection model outputs predicted bounding boxes around the detected markers and a corresponding confidence score. Each marker with distinct patterns is sorted into different classes such that pairs of predicted bounding boxes around the same marker (from the left and the right camera) can easily be found by its class. These pairs of bounding boxes are fed into a

corner detector specialized for ArUco markers. Since the corner detector outputs corner position (in the image plane) of the detected marker in the same order for each marker, it is in fact a stereo matcher as well. Thus it finds the corresponding corners in the two bounding boxes. Note that the search space is reduced dramatically, which may favor classical computer vision methods. We also utilize image rectification to simplify the problem of finding matching points between images (e.g. the correspondence problem). Thus the search of corresponding points is reduced to one dimension such that we search along a horizontal scanline. Once the pair of points is matched, 3D reconstruction (for each pair of matched points) can be obtained directly using stereo triangulation for calculation of the disparity map. The down-right corner of Figure 7.1 shows how a pair of corresponding points together with the position of each camera is used to construct and intersecting lines to obtain 3-D coordinates of the point. From the disparity map, the relative 3D position between one of the cameras (usually the left) and corner points in the markers can then be obtained for each frame in the sequence. Note that it is essential that the images are taken at the same time (e.g. are synchronized) for the algorithms to produce correct measurements.

The pipeline is partly inspired by a thesis performing 3D pose estimation with an older version of YOLO (v2) and a monocular camera [56]. Instead of using fiducial markers, natural occurring cones on the track are used as reference objects. In comparison with our proposed stereo vision pipeline, the implementation in this thesis is fairly more complicated as it is based on a monocular system and object priors. The reader is encouraged to compare section 6.2 in [56] with our proposed pipeline in this section to understand why. In addition, we propose to use classical computer vision techniques for corner detection and matching (referred to as keypoint regressor in [56]), while a machine learning approach is applied for the same problem in the formula thesis. The reason for this choice is that we believe classical detection techniques are sufficient when the problem is reduced (e.g. only detect points in a bounding box and not in the whole image).

7.2 Camera Integration Pipeline

This section will focus on the first part of pipeline (see Figures 7.1 and 7.2). That is, the components needed to feed raw camera streams from the stereo vision system into the object detection model. This section will mainly focus on practical aspects such as how the camera driver works, bandwidth and network requirements to maintain a stable camera stream and some key camera parameters for the final configuration. In this context, some related challenges will be mentioned as well.

Camera driver

When designing the proposed vision pipeline, it is important that the camera streams are published in a format that the object detection model understands. Since the object detection model is implemented in the ROS ecosystem, the obvious choice was to find a camera

driver that was ROS compatible. The main motivation for using ROS is that it simplifies the communication and data transfer across multiple systems. Later in this chapter, we will see that these capabilities are beneficial for other parts of the proposed vision system as well. Now, we found a ROS camera driver suited for FLIR blackfly S cameras to match our criteria (see here [57]). This driver let us control both task-specific and system configuration parameters. The system configuration parameters let us specify which cameras to connect and which camera to be master for triggering the other camera (e.g. for stereo setup). We refer to Figure 5.4 to show how the GPIO cables were connected to achieve hardware triggering. The task-specific parameters let us control things like color mode, fps (frames per second), exposure time, binning (related to resolution) and so on. If you want to configure other task-specific parameters, it is recommended to use Spinview which is a GUI (Graphical User Interface) to control and set different camera parameters.

Network and bandwidth requirements

Before you can play around with different camera configurations, some network requirements are needed so the PC can reach the cameras. The first step is obviously to connect the devices physically using ethernet cables and a switch (see Figure 5.1). When a physical connection is established, the next step is to reach the cameras by its IP address. The camera assigns its current IP address using one of three options: Persistant, DHCP (Dynamic Host Configuration Protocol) or LLA (Link-local address). After some trial and error, persistent IP was found to be the optimal solution for maintaining a stable connection. With this choice, the camera is always reached on the same IP address, even after a power cycle. We also tried dynamic IP addresses using DHCP with a lower success rate. Especially after the lidar was added to the same switch, the PC struggled to reach the cameras with DHCP. Consequently, we ended up with the safest network solution for our application, namely persistant IP.

Since the cameras produce a considerable amount of data, some adjustments were needed to handle issues related to bandwidth. By default, linux places very restrictive limits on the performance of UDP buffer (25 Mbps). In the same manner, the USB buffer is also quite limited. Therefore, both limits were increased significantly (up to 1500 Mpbs) to meet the bandwidth requirements for our application. To illustrate how low these limits are, the bandwidth through the network interface was measured using the linux tool *iftop*. Measurements showed that the final camera and lidar configuration required 81 Mbps per camera and 126 Mbps for the lidar. In other words, almost 300 Mbps for the three sensors!

Camera configuration

The main objective of our application is to push out high-quality images frequently. In this context, the bandwidth consumed by the cameras is closely related to how the final camera configuration will look like. Bandwidth is the amount of data that can be transmitted in a fixed amount of time and is usually expressed in bits per second (bps) or bytes per second. Given an upper throughput limit for the switch, there is typically a tradeoff between

the image quality for each frame and how frequently it is published (usually measured in frames per second). When it comes to frequency, we were able to deliver images at 8-9 fps with the given resolution. Although it is not considered "hard" real-time, it is sufficient for our application as the docking is assumed to be a slow, fine maneuvered operation. When it comes to image quality, it is desirable to achieve maximum image resolution to enable long-range measurements. In our case, this means 1280×1024 resolution. If we were to achieve measurements at even longer ranges, it would require a camera with higher maximum resolution. In addition, you may need to increase the baseline between the cameras to achieve reasonable measurements from the stereo vision system at ranges above 15-20 m from the dockside. That being said, the monocular camera system could potentially deliver decent measurements given a sufficiently high image resolution. However, this is out of scope for this project as the main objective is to deliver accurate localization measurements in the last (and most critical) phase of the docking.

Now, the image resolution and the fps do not solely determine the amount of data pushed through the system. The pixel format is also of great importance as each pixel can store a different number of bits depending on the pixel format. This is usually referred to as bits per pixel (bpp). Since the ArUco markers to be detected only store two colors (black and white), we can greedily choose the monochrome pixel format to reduce the bandwidth. Also, the chosen machine vision cameras do not deliver a compressed image format (i.e. H.264 or H.265 codec) by itself, which means that the ingredients to produce the raw image are of even greater importance.

7.3 Object Detection Pipeline

Before diving into each sub-part of the process, lets briefly get an overview of the learning-based pipeline. Figure 7.3 summarizes the whole pipeline from how raw data is collected to a fine-tuned object recognition model.

In step 1, data is collected and prepared based on the predefined learning task which is to robustly detect and distinguish between markers placed at some dock-side from different views. Step 2 involves transforming the prepared data into a format that the deep learning model understands. As YOLOv3 is a supervised learning method, it needs ground truth labels. These can be provided using an annotation program to manually label ground truth objects that are supposed to be learned. In step 3, the labeled data is fed into the model together with pre-trained weights. When training looks promising, training is stopped, weights are validated and the final set of weights is produced for testing. In Step 4, the fine-tuned model is tested on new unseen data (e.g. test data). Statistical metrics like mAP, IOU and Recall are central, both for evaluating training and for testing the final model. Finally, the model is tested more extensively in operational use, in step 5. Here, you may encounter challenges which were not presented in the dataset the model was trained for.

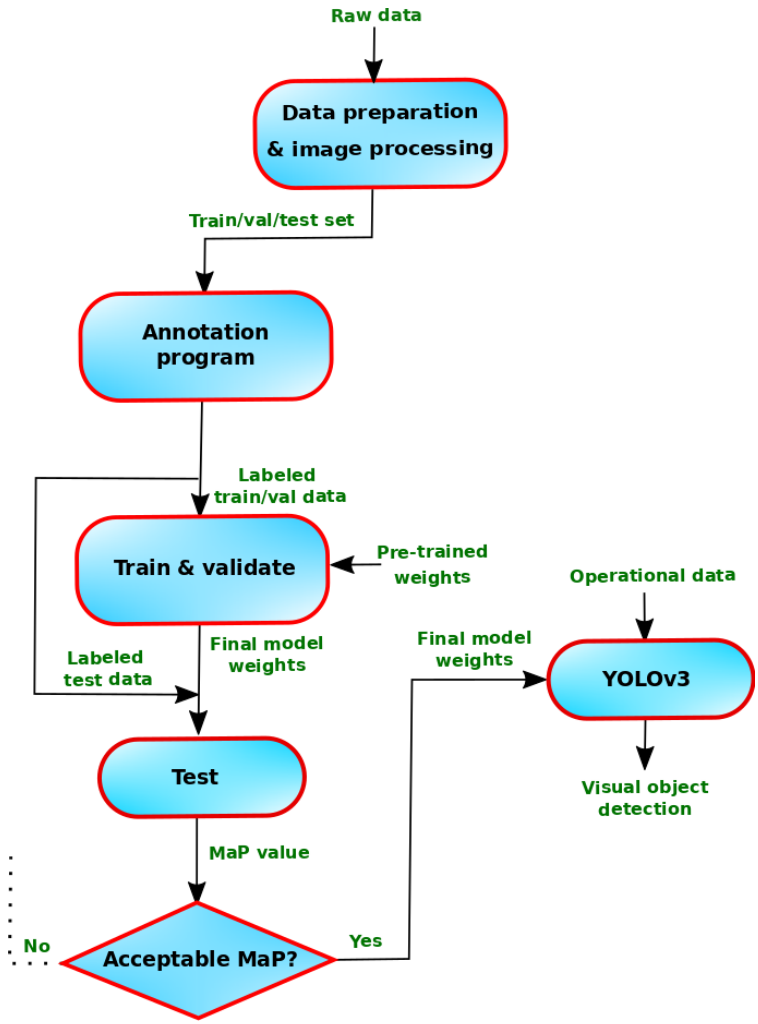


Figure 7.3: Object detection pipeline. A step-by-step illustration of the object detection pipeline from data collection towards a fine-tuned object recognition model. First, images are collected and prepared based on the learning task. Then, relevant images are labeled using an annotation program to obtain a ground truth for the supervised CNN to learn. In the next step, labeled data is fed into the data-driven detection model, together with pre-trained model-weights, to fine-tune the model weights. A validation set is used for model selection, and thereby decide when the model should stop training. At last, the fine-tuned model is tested on unseen data to evaluate its accuracy, normally by the mean average precision (MaP). If results are satisfying, the final model weights are saved and used for operational use in object recognition tasks.

Figure 7.4: The annotation tool Yolo Mark in use during the labeling process.

7.3.1 Step 1: Data preparation and pre-processing

In the field of supervised learning, data is essential for training CNNs. Data is used to give ground truth examples that are relevant to the learning task. But before any such examples can be learned by the model, the data should be prepared and transformed into a format that the model understands. The first step is to gather a custom dataset that represents examples of what the model should learn. Chapter 6 describes in detail how the custom dataset is gathered and which data that is included in it.

In addition, it is often preferred to do some image processing to prepare the input data. It may be to speed up the pipeline (e.g. reduce the image size), to simplify a complex and noisy image or to make it more "correct" (e.g. remove distortion) with respect to the real world. In this context, both camera calibration and downsizing of raw data can be applied. Camera calibration is a widely used technique to remove distortion in the image.

A consequence of using fisheye or wide-angle cameras is that distortion will always occur (at least to some extent). As the final goal is to estimate 3D points accurately, it is desirable to correct input images for distortion so mapping between 3D coordinates and the image plane is correct. In my own project thesis, a GoPro camera with fisheye lenses (and high degree of distortion) was used. Therefore, camera calibration was performed to undistort the input images before they were learned by the object detector. On the other hand, if we ignored this aspect, the objects may have been learned "incorrectly" compared to how they appear in the real world. However, as our (new) wide-angle lenses produce very low distortion ($< 0.4\ \%$), we may not need to remove distortion before 3D reconstruction is performed. In other words, we can pass through undistorted images to the object detector.

7.3.2 Step 2: Labeling process

When the custom data is prepared and necessary pre-processing is done, the next step is to make ground truth labels from each image in the dataset representing what the model should learn. These ground truth labels are used to tell the deep learning model what the correct answer is (i.e. a supervised model). An annotation program called Yolo Mark (suited for YOLOv3) is used to achieve this. In practice, the annotation process goes like this: For each image, simply drag rectangle-shaped bounding boxes around each relevant object (e.g. the markers) and assign its class. Figure 7.4 shows 2 different markers marked by bounding boxes with colors corresponding to each class. In general, precise labeling is very important for the learning process. Often, unexpected learning is a result of inaccurate or lack of labels. For instance, only labeling some parts of the object can be dangerous as the model then learns that this is the whole object.

7.3.3 Step 3: Training and validation procedure

This section explains implementation details relevant for the final training regime.

Transfer learning and fine-tuning

Transfer learning is a widely used technique in the deep learning field. The method aims to transfer the learning outcome of low level features that many object shares in common and also, avoid training the model from scratch which can be very time-consuming, even for GPUs [58]. For this project, model parameters trained on a big and general dataset called ImageNet (reviewed in Chapter 6) was used such that the model already has learned general patterns. The intention is to use these pre-trained model parameters as a starting point and fine-tune them with the custom dataset. In Figure 7.3, these inputs are denoted "pre-trained weights" and "labeled train/val data". The pre-trained model parameters used for this project is called "darknet53.conv.74" and are usually referred to as the pre-trained weights. For more information about transfer learning, see here [58].

Training parameters

Table 7.1 summarizes the final choice of training parameters. The first horizontal row shows the neural network architecture, the second considers parameters related to GPU processing power, while the third focuses on the input image. The fourth controls how the model parameters (e.g. the weights) are updated, the fifth involves parameters relevant for the learning rate and the last serves parameters related to data augmentation. Given the chosen network architecture, all of the parameters can be adjusted in the configuration file. Now, each of the parameters deserves some attention.

- **Architecture**: Different versions of the original YOLOv3 network exist. For this project, the original YOLOv3 network architecture with spatial pyramid pooling (SPP) was chosen as it achieved the best MaP (60.6 %) on COCO dataset using 0.5 IOU threshold. We did also some experiments with tiny YOLOv3, a light-weight version of YOLOv3. It did however not deliver sufficient detection quality for small

Table 7.1: The final choice of training parameters for the YOLOv3-spp architecture.

Training model and parameters	
Architecture	YOLOv3-spp
Batch size	64
Subdivision	32
Width	416
Height	416
Channels	3
Momentum	0.9
Decay	0.0005
Learning rate	0.001
Burn in	1000
Max batches	10000
Policy	steps
Steps	8000,9000
Scales	0.1,0.1
Angle	0
Saturation	1.5
Exposure	1.5
Hue	0.1

objects and we decided to drop it in favor of YOLOv3-spp.

- **Batch size and Subdivision**: The batch size determines the size of a subset of the training images used for one iteration. For a large number of images, it is not practical or necessary to use all in the training set at once to update the weights and hence, a batch of images is used instead. It also shows how many images are averaged over at the same time, which helps to generalize the training. The batch size is set to 64, which is a default value.
  Subdivision involves splitting images in one batch into several mini-batches. The intention is to control how many images are trained for in parallel (i.e. the number of images processed by the GPU simultaneously). With a decent image resolution (416x416), it was necessary to divide the batch into 32 mini-batches so the GPU did not run out of memory (e.g. the Nvidia GTX 1060 in use).

- **Width, Height and Channels**: The input images for training are resized to Width $\times$ Height. Choosing high-resolution increase accuracy during training, but slower the processing time. A decent image resolution (416x416) is chosen as it is desirable to learn detection of small objects during training. Channels are chosen to 3 as the model should be able to process RGB images (e.g. 3 channels) if necessary.

- **Momentum and decay**: Some parameters are used to control how the weight is updated. The CNN is usually updated based on a small batch of images and not the

entire dataset. Due to this reason, the weight updates may fluctuate quite a bit. In this context, the parameter momentum is used to penalize weight updates that differ largely between iterations [59].
A CNN typically has millions of model parameters and can easily overfit to any training data. One way to approach this problem is to use regularization techniques which artificially forcing your model to be simpler. In this context, one such technique is to add a penalty parameter in the cost function (also called the loss function). The decay parameter controls this and aims to penalize large weight values and consequently simplify the complexity of the model and prevent overfitting.

- **Learning rate**: The learning rate controls how aggressively the model should learn. Picking too low learning rate may lead to slow learning process. Picking too big learning rate is far more dangerous as it most likely cause divergence in training loss and never reaches an optimum during training. However, learning rates are all dynamic in modern gradient descent based networks. The initial value is chosen to be 0.001 (default value).

  Burn in: It has been empirically found that the training speed tends to increase if the learning rate is lower for a short time after initialization [59]. This short time is controlled by the burn in parameter. The value is set to 1000, meaning that the learning rate is lowered until it reaches 1000 iterations.

  Max Batches: Max batches determine how many iterations the network is trained for and hence, the total training time. Different tests showed that 10000 iterations were enough to find an optimal model and to confirm overfitting using a validation set.

  Policy, **steps** and **scales**: Policy determines how a learning rate should increase or decrease. For this project, the learning rate is changed stepwise during training. Given a stepwise policy, steps decide at which iterations the learning rate should be changed. At 8000 iterations, the learning rate is scaled by 0.1 and after 9000 iterations, it is once again scaled by 0.1. These scalings are determined by the scale parameter.

- **Angle, saturation, exposure and hue**: As data collection and labeling can be very time consuming, it is proposed to make synthetic data based on the collected data to produce even more and improve the learning outcome. This is called data augmentation and aims to fill the gap between limited data of real images and strict requirements for robustness of outdoor detections. In this context, the angle parameter determines how many degrees a given image can be rotated (+-). The colors in the image can also be transformed by adjusting the parameters saturation, exposure and hue. The default values are used for training.

Given the training regime described above, the network was trained locally using a Nvidia GTX 1060 GPU. Also, the model is trained with the default IOU threshold 0.5. One training with the given training configurations took approximately 48 hours to succeed. This is indeed a matter of hardware. Figure 7.5 shows training loss during training.

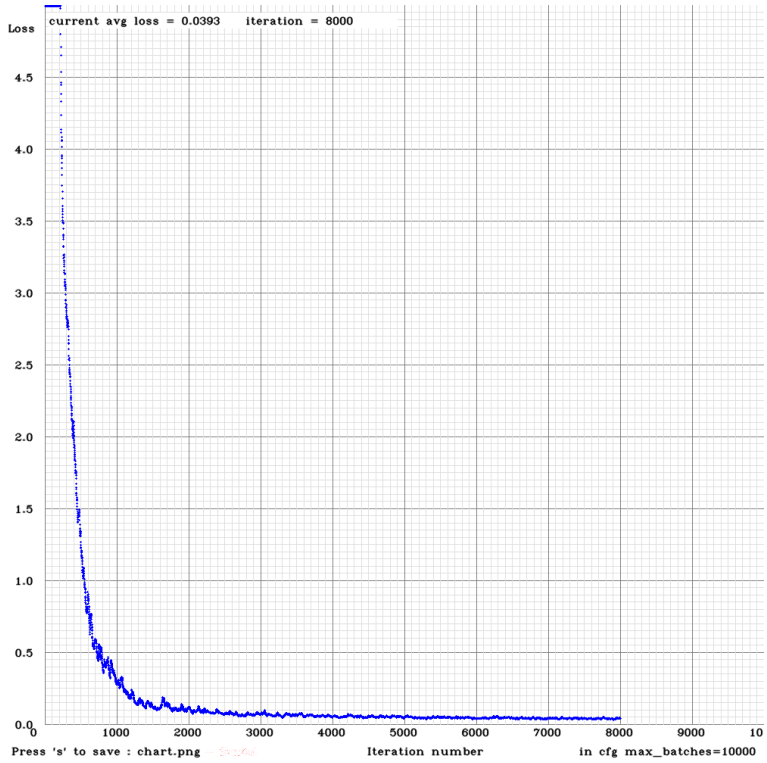


Figure 7.5: Training loss during the final training scheme. After 2000 iterations, the training loss tends to converge. Based on earlier experiences, it was not necessary to train for more than 8000 iterations.

Validate training

When the training period is succeeded, it is desirable to measure somehow which model parameters perform optimally across time (e.g. iterations or epochs). Deep learning models usually learn the optimal model parameters towards the end as the learning is an optimization process if training parameters are tuned correctly. The main problem is however that deep learning algorithms look for deep complex patterns that humans not even understand when the model is trained too long. And consequently, it would no longer fit the expectations of a human evaluator (i.e. overfitted to training data). It is therefore desirable to validate the training using a validation set. The validation set is a sub-part of the custom dataset (left away from training) which is used for model selection. That is, picking the model that performs most accurate on unseen data. Several accuracy measures exist like MaP, Recall and IOU as reviewed in Chapter 3.

Now, the model selection deserves some comments. Among data scientists, the normal practice is usually to save the model during training for each epoch, train for a large number of epochs and retain the value with the lowest loss on the validation set. Unfortunately, the used framework does not offer these capabilities by built-in functions. For this project, validation MaP (instead of validation loss) is used to validate the training data due to its ease of availability in the framework. It is also widely used and accepted in the data science community. In Table 7.2, MaP calculations is performed to validate training for every thousand iteration. Note that MaP is derived from AP of each class (i.e. the mean of the APs across the classes). Also note that AP is usually derived from precision-recall (PR) curve. For this project, the Area Under Curve (AUC) approach is used to compute AP for each class, that is, to compute the area under the interpolated PR curve to obtain the APs for each class. For more information about the method, see subsection 3.1.5.

The final model is therefore chosen as the one with the highest calculated MaP on the validation set. From Figure 7.2, observe that we achieve the highest achievable MaP after 3000 iterations. The problem is however that the MaP value is similar for all iterations between 3000 and 8000 iterations. For this reason, we look for additional metrics such as IOU to give us more insight. From Figure 7.2, we can see that the IOU value tends to decrease after 6000 iterations. This may indicate overfitting. Consequently, we ended up picking the model parameters trained for 6000 iterations. That being said, a small validation set makes it harder to select an optimal model and one may also consider to use other techniques such as cross-validation when working with small validation sets or simply collect more data for validation. Now, in comparison with the normal validation practice described, this is not an optimal model selection. As seen from Figure 7.5, the training loss oscillates a bit and a peak may occur right before some of the saved model parameters (which is saved for each thousand iteration). In other words, instead of retaining the optimal model parameters during training, different model parameters are saved at certain intervals and the optimal model is most likely hidden in between some of the saved models. A workaround could be to sample saved model parameters more frequently, but a laptop may run out of memory as one model parameter file stores more than 250 MB.

Table 7.2: Comparison of MaP and average IOU (threshold = 0.25) for different weights with the chosen training configuration.

MaP comparison on validation set		
Iterations	MaP (%)	IOU (%)
1000	9.20	72.38
2000	9.37	84.55
3000	9.38	85.74
4000	9.38	86.32
5000	9.38	86.15
6000	9.38	87.03
7000	9.38	86.77
8000	9.38	87.28

7.3.4 Step 4: Test procedure

The test set will be used to test the final model (chosen in the validation procedure) on new unseen data and hence, verify how well the model works in reality. As described in Chapter 5, the custom dataset is well mixed together and randomly shuffled before it was split into training, validation and test set. This means that the 10 % of the custom dataset assigned for testing represents a wide range of scenarios from the dataset. As with the validation procedure, MaP will be used as the main accuracy measure. In general, the model can be accepted if the MaP value achieves a considerably high value. This value depends on how big and challenging the dataset is. For this case, the dataset is considered quite simple as the classes contain easily identifiable markers and one can therefore expect the CNN to deliver high MaP values on the test set.

Now, the final model (trained for 6000 iterations) was tested on the test-set (e.g. 96 unseen random images in the custom set) and achieved 99.40 % MaP among all the classes. These model parameters will be used for the final experiments.

7.3.5 Step 5: Operational use

If the results from the test set are satisfying, the next step is to test the object detection model in operational use (i.e. the last step before the model is ready for commercial use). That could be test scenarios (in real-time) that the model is trained for. Of course, the variation and complexity of the operational environments are constrained by the training data and in that context, data collection is essential. Thus, the real-time data for operational use should be reflected by the training data. In other words, the more general and robust detection model you want to achieve, the more comprehensive the data acquisitions are needed.

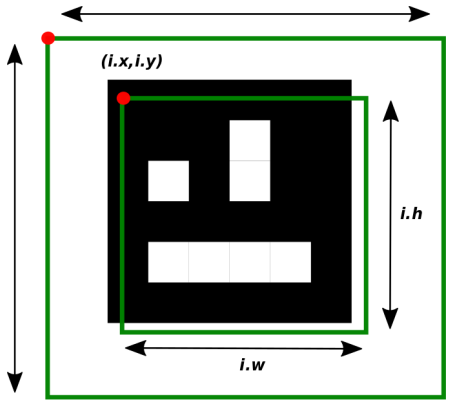


Figure 7.6: The Figure illustrates how the bounding box is extended. Initially, the CNN predicts the inner bounding box which may not include all the marker corners. Therefore, both the red starting point ($i.x$, $i.y$) and the height and the width ($i.h$ and $i.w$) of the bounding box are adjusted as a function of the bounding box size to include all the marker corners.

7.4 3D Reconstruction Pipeline

7.4.1 Overall design

The previous section showed how to produce accurate bounding boxes around the markers for a well-trained CNN. This helps us to reduce the scope and complexity of the 3D reconstruction task. Instead of performing 3D reconstruction directly on the whole image, the task is now reduced to perform this on a small sub-part of the whole image. This also reduces the chances of producing outliers.

In the following, we will introduce some design choices in order to simplify both monocular and stereo vision techniques for 3D reconstruction. As shown in Figures 7.1 and 7.2, we have two YOLOv3 instances running in parallel producing frequently bounding box estimates (assuming the markers are recognized in the camera view). To be able to perform stereo vision, two bounding box predictions from left and right camera view at the same timestamp are needed. As a design choice, the predicted bounding boxes were sent along with its coordinates relative to the whole image from one YOLOv3 instance over to the other with ROS. The bounding box image and its coordinates are published as topics from one of the ROS nodes (e.g. YOLOv3 instance), while the other one subscribes for the corresponding topics. Figures 7.1 and 7.2 illustrate how we gather this information to perform the last part of the pipeline. With a bounding box pair available at every timestamp, several opportunities are available. It allows us to perform both monocular and stereo vision techniques for 3D reconstruction at once. Thus, both techniques are implemented in C++ and run really fast (especially on small bounding boxes). We assume the techniques to be performed approximately at the same time and then, they are directly comparable. Recall that monocular 3D reconstruction only relies on one image view together with some prior knowledge of the marker size.

Since both algorithms are strongly dependent on the whole marker including the corners to be visible inside the bounding box, it is necessary to extend the bounding box slightly. This ensures that the marker detector from the ArUco library is able to recognize the markers in cases where the bounding box prediction does not sit perfectly around the marker. To obtain bounding boxes with meaningful content across any scale, a bounding box extension as a function of the bounding box size was chosen. The following line of code summarizes this.

```
cv::Rect ROI(i.x*(1-i.w*0.001),i.y*(1-i.h*0.001),i.w*2.5,i.h*2.3);
```

Here, `i.x` and `i.y` correspond to the pixel coordinate of the upper left corner of the bounding box relative to the whole image. `i.h` and `i.w` denote the width and height of the bounding box measured in pixels. We moved the red starting point (`i.x`,`i.y`) slightly in addition to extending the width and height of the bounding box as a function of the bounding box size. If we refer to Figure 7.6, this corresponds to the outer (green) bounding box.

7.4.2 Camera calibration

In order to determine the camera location in the scene, we need to perform camera calibration. To estimate the camera parameters, one needs to obtain 3D world points and their corresponding 2D image points. This mapping can be obtained by using multiple images of a calibration pattern (i.e. a checkerboard).

In total, 48 images of a 7x10 checkerboard were taken from different views and orientations using the Blackfly S cameras. That is, 24 images for each camera. Figure 7.7 shows one of the stereo pairs used to find correspondences between the 3D world points and their corresponding 2D image points. By using the stereo calibration app in Matlab [60], 48 images were added and all of them were approved by the calibrator app. Further, the length of the checkerboard square was measured manually and inserted into the calibrator app. A regular camera model (instead of a fisheye camera model) was chosen in order to achieve the correct calibration.

To evaluate the camera calibration, the reprojection error is used as a measure. The original mean reprojection error (of the 24 images) was 0.14 in pixels. To improve the calibration, outliers (images with the highest reprojection error) were removed and a recalibration was done. In total, 2 images where removed and the 22 remaining images gave 0.13 mean reprojection error in pixels as shown in Figure 7.8. With 1280×1024 image resolution provided by the cameras, this is considered as quite good results. Therefore, we decided to save the corresponding calibration parameters and apply them for the final experiment.

Note that, in contrast to the single camera calibrator app, the stereo camera calibration app provides two individual set with camera parameters suited for the stereo pair. In addition, it provides a transformation matrix to map between the camera coordinate systems. As we can easily measure the fixed distances between the cameras (i.e. the offset along the x-axis in Figure 7.9), we can easily verify whether the transformation matrix seems correct or not.

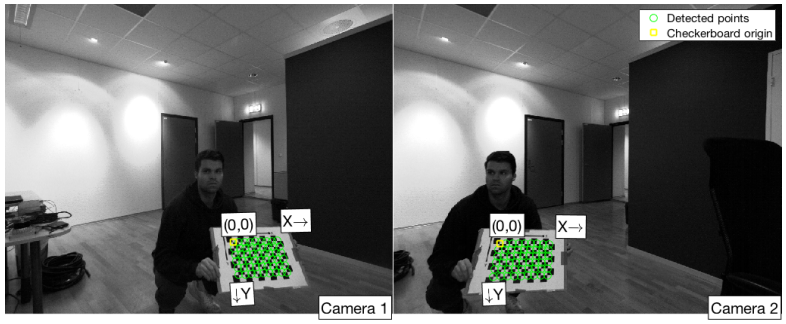


Figure 7.7: The stereo camera calibrator app in Matlab [60] estimates camera intrinsics, extrinsics and lens distortion parameters. Here, the app provides corner detections of the checkerboard pattern for one of the 24 approved stereo pairs during the calibration process.

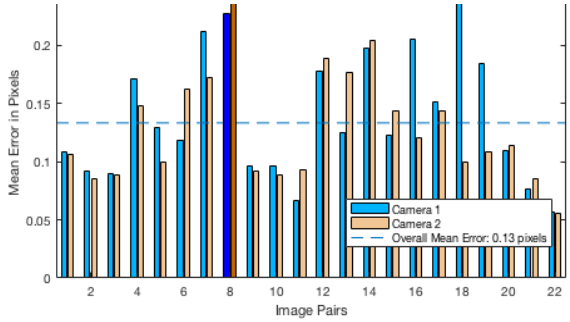


Figure 7.8: Reprojection error after re-calibration.

The lens configuration in this thesis produces less than 0.4 %. However, this does not mean that we can ignore the impact of distortion totally when calibrating the cameras. In fact, we estimate the distortion parameters with three radial distortion coefficients and two tangential distortion coefficients during the calibration process.

7.4.3 Stereo vision design

In this section, the algorithms to perform 3D reconstruction from stereo vision will be presented. A brief overview is given, before implementation details for each algorithm are given. If we refer to Figure 7.10, a pair of bounding boxes around the same marker (detected from left and right camera) are fed into a corner detector specialized for ArUco markers. Fortunately, the corner detector delivers the four corners of the detected marker in the same order every time. This simplifies the matching strategy significantly. It allows us to detect the same marker from different views (using the same corner detector) and match stereo points (the same point seen from two different camera views) directly. The stereo pairs are updated in terms of pixel coordinates relative to the whole image instead of the bounding box to achieve correct results. At last, the stereo pairs are passed into a triangulation algorithm along with projection matrices (obtained from the calibration parameters) to obtain 3D points of the corners relative to the left camera.

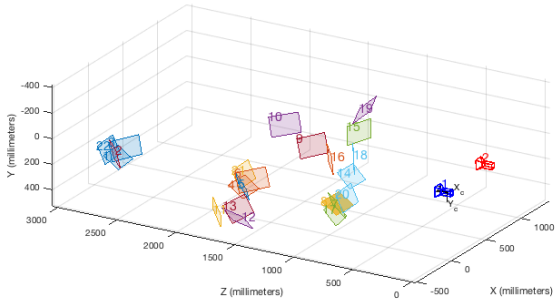


Figure 7.9: Visualization of the extrinsic parameters. That is, how the stereo pair is located relative to the checkerboard patterns. One can easily see the different checkerboard patterns in the 3D space.

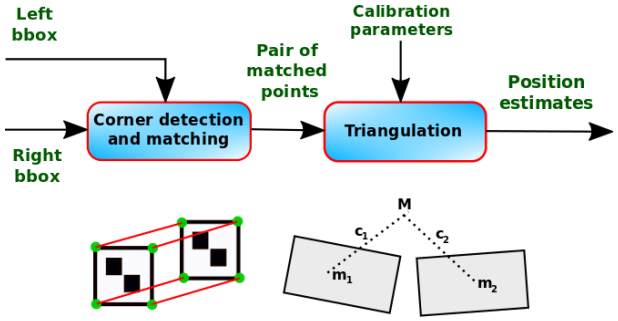


Figure 7.10: 3D reconstruction with stereo vision design.

Corner detection and matching

Implementation details will for the stereo algorithms used in this thesis will now be explained. Firstly, the marker detector finds the location of the four corners of the marker. The following two lines were used in the implementation to achieve this for one of the markers. The first line represents marker detection inside the bounding box relative to the left camera view, while the second line represents marker detection inside the bounding box relative to the right camera view.

```
cv::aruco::detectMarkers(bbox_left_1, dictionary, markerCorners_l_1,
markerIds_l_1, parameters, rejectedCandidates_l_1);

cv::aruco::detectMarkers(bbox_right_1, dictionary, markerCorners_r_1,
markerIds_r_1, parameters, rejectedCandidates_r_1);
```

- `bbox_left_1` and `bbox_right_1` denote the input bounding box image of the marker seen from left and right camera view, respectively. Hence, only a small part of the whole image is passed through this algorithm.
- `dictionary` indicates the type of markers that will be searched. The main properties of a dictionary are the dictionary size and the marker size. Dictionary size 4×4 and marker size 250 mm were used. When the dictionary is specified, the algorithm searches for the marker id within the dictionary it belongs to (given an input image of an ArUco marker).
- `markerCorners_l_1` and `markerCorners_r_1` store pixel coordinates of the four corners of the marker from left and right camera view, respectively. They are returned in their original order which is clockwise starting with the top left. This allows us to simplify the matching strategy as already discussed.
- `markerIds_l_1` and `markerIds_r_1` is the list of ids of each of the detected markers, seen from left and right camera view.
- `parameters` is an object that includes all the options that can be customized during the marker detection process. This includes detector parameters such as thresholding, contour filtering and bits extraction. Default values were used for the final experiment.
- `rejectedCandidates_l_1` and `rejectedCandidates_r_1` return a list of marker candidates. I.e. shapes that were found and considered, but did not contain a valid marker.

Marker detection was only performed when a bounding box prediction was available. Previous bounding box predictions were deleted before the next iteration to avoid using old measurements.

If the ArUco marker detector produces two sets with marker corners (one from each camera view), the next step is to match these corner points. Since marker corners are returned in the same order for both cameras, we can simply match them directly. They are stored in two vectors (one for each camera view). A stereo pair can be found by calling the same index for these two vectors (`keyPoints_left_1` and `keyPoints_right_1`). Furthermore, the coordinates of the stereo pairs are updated to be relative to the whole image (instead of the bounding box). This is done with the following code.

```
for(int i = 0; i < 4; i++){
    pointsMat_l_1.at<double>(0,i) = keyPoints_left_1[i].pt.x+xmin_left_1;
    pointsMat_l_1.at<double>(1,i) = keyPoints_left_1[i].pt.y+ymin_left_1;
    pointsMat_r_1.at<double>(0,i) = keyPoints_right_1[i].pt.x+xmin_right_1;
    pointsMat_r_1.at<double>(1,i) = keyPoints_right_1[i].pt.y+ymin_right_1;
}
```

Here, `xmin_left_1`, `ymin_left_1`, `xmin_right_1` and `ymin_right_1` denote the upper left corners of the two bounding boxes (one per camera view). By using these offsets, we obtain the stereo pair coordinates relative to the whole image.

Triangulation

Given the obtained stereo pairs (e.g. the four matched corner points) and the stereo calibration parameters, all information available to perform triangulation is available. However, before the triangulation is performed, some pre-calculations are needed. The triangulation algorithm requires two projection matrices, `P1` and `P2`, which can be obtained from `cv::stereoRectify()`. `cv::stereoRectify()` computes rectification transforms for each head of a calibrated stereo camera. That is, given the obtained calibration parameters (`K1`, `D1`, `K2`, `D2`, `R` and `T`) and the full image resolution (`img_size`), the function returns 3×4 projection matrices in the new (rectified) coordinate systems for each camera (`P1` and `P2`). Note that `cv::stereoRectify()` only needs to be called once to obtain the static projections matrices, `P1` and `P2`, at least until a new calibration is performed. The following line of code shows the structure of the function call.

```
cv::stereoRectify(K1, D1, K2, D2, img_size, R, T, R1, R2, P1, P2, Q,
CALIB_ZERO_DISPARITY, 0, img_size, &roi1, &roi2);
```

The rest of the function parameters are not relevant for the implementation and will therefore not be discussed in detail.

Now, with `P1` and `P2` available, we can finally perform triangulation. The triangulation algorithm reconstructs 3D points in homogeneous coordinates by using their observations with a stereo camera. The following line of code shows the structure of the function call.

```
cv::triangulatePoints(P1, P2, pointsMat_l_1, pointsMat_r_1, pnts3D_1);
```

- `P1` and `P2` are the projection matrices for the left and the right camera, respectively.
- `pointsMat_l_1` and `pointsMat_r_1` store the stereo pairs obtained in the previous section.

- `pnts3D_1` returns the triangulated points in 3D relative to the left camera (in homogeneous coordinates).

Here, `cv::triangulatePoints()` produces the reconstructed 3D points in homogeneous coordinates (`pnts3D_1`). Thus, cartesian coordinates are transformed since they are easier to relate to, especially when comparing and evaluating the final results. We refer to the OpenCV documentation for 3D reconstruction [45] to see how the points are located relative to its coordinate system.

At last, we transformed the four triangulated points from their corners into the center of the marker. This to compare the measurements directly with measurements from the monocular vision design, which will be reviewed in the next section. In addition, we took the median of the four shifted 3D points. By this, we reduce the four points into one robust estimate. Since the median is more robust against outliers, it makes more sense to apply the median compared to the mean. However, since the median is defined on odd numbers, the measurement is in fact found by taking the mean of the two middlemost measurements.

7.4.4 Monocular vision design

The monocular vision design shares many commonalities with the stereo vision design. Therefore, we will not repeat all the procedures in detail. A quick overview of the algorithms is given. Refer to Figure 7.11, a pair of bounding boxes around the same marker is fed into a corner detector (the same ArUco marker detector as in the stereo vision design). This to obtain the relative pose with respect to each camera individually. As with the stereo design, the marker detector finds the location of the marker corners in the bounding boxes. The corner points are updated with an offset to obtain the corner points relative to the whole image. At last, the detected marker corners are passed into a monocular pose estimation algorithm together with the prior length of the marker size and calibration parameters to output the pose estimation of the marker with respect to each camera individually.

Corner detection

This part shares many commonalities with the stereo vision design. Both designs apply `cv::aruco::detectMarkers()` to detect where the marker corners are located in the bounding box. In contrast to stereo vision, monocular 3D reconstruction only relies on one image view together with the prior known length of the marker size which simplifies the design significantly. That is, there is no need for matching strategies. Further, we do not need to worry too much about how well the cameras are synchronized as the cameras produce two independent measurements. At last, the corner points are updated to be expressed with respect to the whole image.

Pose estimation

Given the obtained calibration parameters, prior knowledge of the marker size and the four detected marker points for each bounding box (e.g. one per camera view), all information is available to perform monocular pose estimation. The monocular pose estimation

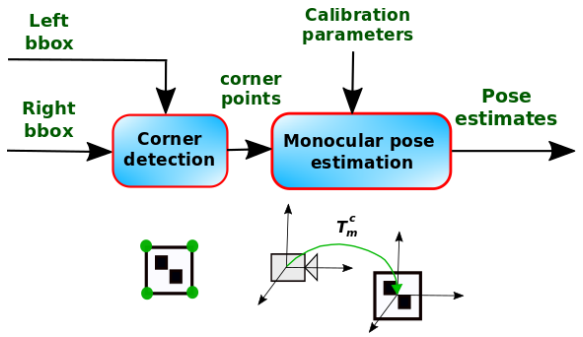


Figure 7.11: 3D reconstruction with monocular vision design.

specialized for ArUco markers is presented. The following two lines of code show the function calls to perform monocular pose estimation from each camera individually.

```
cv::aruco::estimatePoseSingleMarkers(markerCorners_l_full_1, 0.2628, K1,
D1, rvecs_l_1, tvecs_l_1);
cv::aruco::estimatePoseSingleMarkers(markerCorners_r_full_1, 0.2628, K2,
D2, rvecs_r_1, tvecs_r_1);
```

- `markerCorners_l_full_1` and `markerCorners_r_full_1` represent the marker corners seen from left and right camera view, respectively. As before, they are expressed with respect to the whole image and not the bounding box.
- 0.2628 is the measured marker length. The unit is meter.
- `K1`, `D1`, `K2` and `D2` represent the necessary calibration parameters for each camera, individually, to perform monocular pose estimation. `K1` and `K2` represent the 3×3 camera matrix for each camera, while `D1` and `D2` represent the distortion coefficients for each camera. The distortion coefficients can be of 4, 5, 8 or 12 elements depending on the calibration.
- `rvecs_l_1`, `tvecs_l_1`, `rvecs_r_1` and `tvecs_r_1` are the returned translation and rotation vectors for each marker. The returned transformation transforms points from each marker coordinate system to the camera coordinate system. The marker coordinate system is centered in the middle of the marker, with the z axis perpendicular to the marker plane.

Note that the monocular pose estimation algorithm expresses the measurements in the marker coordinate system, while the stereo triangulation algorithm expresses its measurements in the camera coordinate system. Also note that the monocular pose estimation algorithm provides us with orientation measurements, in addition to positioning measurements. For more information and documentation of the chosen algorithm for monocular pose estimation, we refer to the OpenCV library for ArUco markers [61].

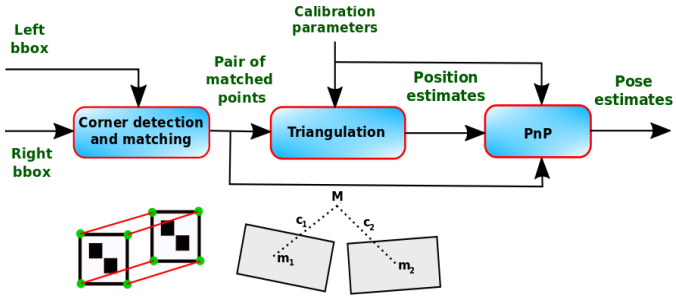


Figure 7.12: The first design proposal use a PnP solver to recover orientation estimates from stereo vision, by using object points and their projected image points.

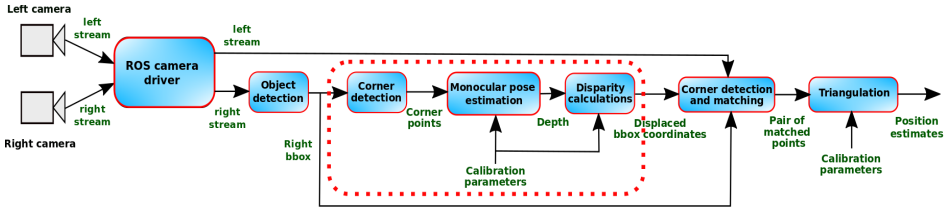


Figure 7.13: The second design proposal use object priors to shift the bounding box coordinates such that it fits the marker seen from the second camera view. By this, only one object detection model is used for a less computationally heavy pipeline.

7.4.5 Future design improvements

This section briefly introduces some candidates for future improvements to the pipeline. The first proposal suggests expanding the stereo pipeline to achieve orientation estimates, while the second proposes a way to achieve stereo reconstruction with one detection model. Both proposals can be fused into the same pipeline, making the stereo vision system able to recover full pose estimation with less computational power.

- **Pose estimation for stereo vision:** Recall from Chapter 4 that an object pose can be obtained from 2D-3D correspondences. Since the stereo vision design provides us with position estimates corresponding to their corner points in the image, all information needed is available to reconstruct the full pose. We propose the PnP solver `solvePnP()` from the OpenCV library [62] to approach this problem. This function calculates an object pose given some object points, their corresponding image projections, as well as the distortion coefficients and the camera matrix. Figure 7.11 illustrates how these inputs are fed into the PnP solver to obtain pose estimates. We assume four point correspondences in each marker are sufficient for the pose estimation scheme.

- **Using one CNN instead of two:** Indeed, running two deep, computationally heavy CNNs in parallel on a small embedded computer can be challenging. As will be discovered in Chapter 9, the embedded system tends to shut off during the testing phase, most likely triggered by the TTP operating temperature reaching 80 $^{\circ}C$ (see

Table 2.1 in [63]). Fortunately, the stereo pipeline can be reduced to only run one detection model and still recover 3D points, by using object priors. As soon as the detected marker is available in one camera view, object priors (e.g. the length of the marker object) can be used to calculate the disparity in a stereo setup to shift the bounding box such that it fits around the same marker seen from the second camera view (see the red-dot area in Figure 7.12). We refer to the theory on stereo 3D reconstruction in Chapter 4, to see how disparity can be obtained from object depth given a calibrated and rectified stereo setup. This design clearly removes some heavy processing in the pipeline and still ensures that 3D reconstruction is performed with stereo vision.

One drawback of using one CNN instead of two is however that we loose redundancy. In case one of the CNNs produce false positives, we cannot rely on predictions from the other CNN anymore. Hence, there is clearly a tradeoff between minimizing computational power and increasing redundancy.

Chapter 8

Experiments

This chapter contains details about how the experiments are prepared. It also describes specific details about each experiment. Hopefully, this gives the reader a better overview of the objective of each experiment.

8.1 Preparation of the Experiments

Preparations were needed to conduct experiments with meaningful results. This to compare how well the camera system performs compared to a ground truth lidar sensor. The section also covers some of the necessary assumptions. To give the reader a better overview, a list of the necessary tasks to consider, in chronological order, are summarized below.

- Locate the center of the lidar and the camera, to specify a transformation between the lidar and camera coordinate systems.
- Based on the obtained sensor centers, define a transformation which relates the ground truth lidar sensor and the cameras.
- Decide which measurements to consider.
- Some details about how ground truth measurements can be recovered in the point cloud map.
- Some recording details during the experiments.
- Some trial and error before the complete vision-based positioning system was ready for the final docking experiments.

Where are the sensor measurements located?

Prior to comparison of the measurements between the camera system and the lidar, there is a need for transformation between the coordinate systems. The exact location for sensor

measurements is needed. In this context, some assumptions are needed. Since the lidar points of ground surface are circles with the lidar sensor as the center, it is reasonable to assume that the lidar produces measurements located at its center. Further, we are interested in the camera coordinate system when discussing where the camera measurements are located. From the calibration process, it should be feasible to measure quite accurate where the camera coordinate system is located in the camera. For instance, from Figure 7.9, we can observe where all the checkerboard patterns are relative to the cameras. However, it would be fairly challenging to locate this center accurately when the camera is only $29 \times 29 \times 30$ mm. For simplicity, we assumed the origin of the camera center to be located in the center of the camera. In a worst-case scenario, a ± 1.5 cm error is added to the measurements, which we consider as not too significant.

Defining the static transformation

With these assumptions, we are now able to measure how the cameras are located relative to the lidar. The ground truth measurements can be found in the point cloud map centered around the lidar. Fortunately, the cameras and the lidar are assumed to be fixed with respect to the same bracket. Furthermore, the cameras and the lidar are pointing in the same direction meaning that we only need to add an offset along the x, y, and z-axis to transform from the lidar coordinate system into the camera coordinate system. In the computer vision community, it is common to perform 3D-reconstruction with respect to the left camera when using a stereo setup. For this reason, we chose the left camera as our camera reference system. Furthermore, ROS provides a tool (which we applied) for publishing a static coordinate transform using a x/y/z offset in meters and yaw/pitch/roll in radians. The complete ROS command is

```
rosrun tf static_transform_publisher x y z yaw pitch roll frame_id
child_frame_id period_in_ms
```

where `frame_id` and `child_frame_id` represent the new and the old coordinate system, respectively. We can also specify how often to do a transform. The final ROS command for our purpose became

```
rosrun tf static_transform_publisher 0.015 0.308 0.0 0.0 0.0 0.0
left_cam os1_lidar 100
```

With this command, we publish the left camera coordinate system (`left_cam`), which is only a static offset relative to the original lidar coordinate system (`os1_lidar`). In Rviz (a visualization tool in ROS), we enable axes and a grid around the new coordinate system to visualize it in the lidar map. In Figure 8.3, we can see the left coordinate system centered around the left camera, while the right coordinate system is centered around the lidar sensor. Note that the x and y-axis in the lidar map (see Figure 8.3) corresponds to z and x-axis of the camera coordinate system (see Figure 8.1), respectively. This is because the 3D reconstruction algorithms follow the reference system as defined in the OpenCV documentation for 3D reconstruction [45]. From now on, when we refer to some translation or rotation of the camera, it will be with respect to the coordinate system as shown in Figure 8.1.

Furthermore, we allow ourselves to ignore position measurements along the y-axis. Seen from a practical perspective, the USV is floating on relatively flat water at the inside

of the harbor. This means the USV will maintain more or less the same height during a docking operation. In other words, it is reduced to a 2D-space seen from a top-down view (i.e. like in Figure 8.2). From a navigation perspective, the USV path seen from a top-down view may be sufficient enough for most cases.

Merging measurements into Euclidean distances for comparison

The next issue to address is related to which coordinate system the camera measurements is expressed in. Unfortunately, the stereo algorithm (`triangulatePoints()`) and the monocular pose algorithm (`estimatePoseSingleMarkers()`) do not express the 3D coordinates with respect to the same coordinate system. `triangulatePoints()` express the 3D points of the marker with respect to the left camera, while `estimatePoseSingle-Markers()` express the 3D points of the left camera with respect to the marker. Therefore, we decided to focus on the relative distance between the marker and the camera rather than the x and z measurement separately. The reason for this choice is simply that the relative distance between the marker and the left camera is not affected by which coordinate system it is expressed in as long as they are related to each other. We may assume that there is a strong correlation between the distance measurement and position measurement. That is, if the distance measurement is accurate, it is likely that the position measurements along the x and z-axis are accurate as well.

Seen from a top-down view, we define the relative distance between the marker and the left camera as the Euclidean distance between the middle point of the marker and the center of the left camera. That is, a straight-line distance corresponding to a radius r_c of a circle spanned out by its x_c and z_c component such that

$$\mathrm{r}_c = \sqrt{x_c^2 + z_c^2}\ . \tag{8.1}$$

If we refer to Figure 8.2, the red lines clearly visualize the Euclidean distance between the left camera and the markers which we want to measure. If we were to compare each measurement individually, an alternative would be to apply Rodrigues Formula [64] to transform one of the coordinate systems into the other for a direct comparison between each component of a 3D point measurement.

How the ground truth measurements are obtained

The reader may wonder how it is possible to measure the distance (along x and z-axis) between the marker and the left camera accurately in a point cloud map. Figure 8.3 gives us a starting point for how this can be done. It visualizes a grid centered around the left camera coordinate system where we can determine the physical length of each grid cell. We chose 2 cm length for each grid cell to achieve a decent level of accuracy. We also extended the grid so it intersects with the center of the marker. This can for instance be seen from a top-down view in the point cloud map. When a grid of correct size is chosen, we can zoom in where the marker is located. Here we find the end of the grid and can simply click on the end of the grid corresponding to the middle point between the marker along the x-axis. This can be done quite precisely as we have prior knowledge about the width (along x-axis) of the marker in the real world. In Rviz, when the desired marker

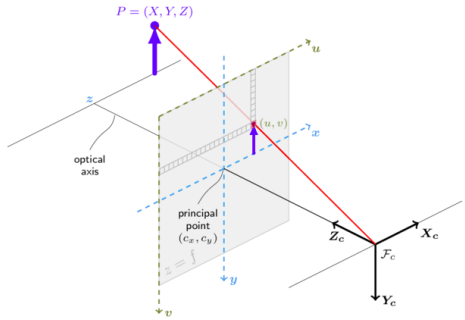


Figure 8.1: OpenCV reference system for the camera. Image courtesy by [45].

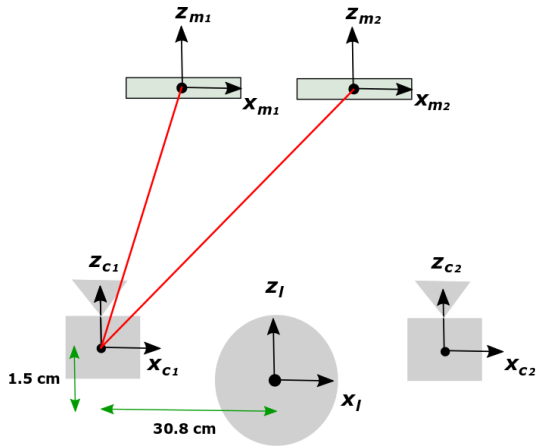


Figure 8.2: From a top-down view, the two cameras are shown with the lidar in between. The left camera is used as our reference system. The static offset to transform between the lidar frame and the left camera frame is shown with green arrows. The red lines show the Euclidean distance between the two markers and the left camera. Two markers were included to achieve over-representation and enable the possibility for more robust measurements.

point is found, this point can be published as a topic via ROS. To get the x and z ground truth measurements, we open a new terminal and type

```
rostopic echo /clicked_point
```

However, it is important to mention that this is not an optimal way to reconstruct ground truth points. It relies solely on how a human interprets the point cloud and it will therefore always be some error between the actual ground truth measurements and the ones interpreted by humans. However, we assume that the points above are limited by an error of ± 5 cm.

Recording the experiments

To record the measurements during the experiments, two methods were mainly used. The first stores all the camera measurements in a text file format for each timestamp. For each line, the stereo measurement is on the format

```
utc time - relative time - length_x_y_z - length_x_z - x - y - z
```

while the monocular measurement is on the format

```
utc time - relative time - length_x_y_z - length_x_z - x - y - z
- yaw - pitch - roll
```

The difference is simply that the monocular measurements provide orientation in addition to position measurements. If we refer to Figure 8.2, there can be up to six camera measurements for each timestamp. That is, two stereo measurements (one per marker), and up to four monocular measurements, as each camera produces individual measurements for each marker, in the latter case. Consequently, we made six separate log files, one for each case (depending on 3D reconstruction technique, camera and marker). This allows us to easily read and plot each type of measurement directly. As already mentioned, we are mainly interested in the Euclidean distance in the x-z plane (`length_x_z`) and its corresponding timestamp (`relative time`).

The other method we applied includes the use of ROSbags. A ROSbag is a powerful tool in ROS, which can be used to record your simulation. It allows you to record all measurements, published as topics, simultaneously. The topic of our interest is mainly the lidar point cloud. In addition, we included images of the predicted bounding boxes to be able to visualize how the CNN handled marker detection at different ranges from the dockside.

Since the Nvidia computer have quite limited storage volume (32 GB), we added a 1000 GB USB storage volume so we did not need to worry about space issues during the experiments. Remember that point cloud data from lidars can produce an enormous amount of data, even for short records. As we are supposed to compare the camera measurements stored in the text files with the lidar point cloud in the ROSbags and they are not launched at the same time, we also need to know the time offset between them. Fortunately, the global start time of a ROSbag is always included. In the same manner, we added the UTC time so that a relative time offset between the records can be obtained.

Some practical aspects

At last, we mention some practical aspects related to the experiments. In total, we did three tests before we achieved reasonable measurements. Under the first trial, we realized that the light-weight CNN tiny YOLOv3 did not perform well unless it was only a couple of meters from the dockside. Therefore, we moved over to apply the original CNN, YOLOv3, to achieve better detection quality at longer ranges as well. Under the second trial, we also realized that there was a quite basic bug in the code. To be able to recognize the corners of each marker, we needed to extend the bounding boxes slightly. A simple solution is to extend the bounding box as a function of the bounding box size. And so we did (to some extent). The width and height were correctly adjusted, but the starting point (e.g. the upper left point of the bounding box) was unfortunately set as a static offset only suitable for one fixed distance from the markers. The consequence was that the markers slowly but surely dropped out of the bounding box and consequently, no measurements were obtained when the USV was more than 4-5 meters from the docking. In our third and last trial, this issue was fixed and we were able to achieve reasonable measurements at different ranges and angles relative to the quay.

These experiences reflect how important it is to simulate and test your system under different conditions. Although you have a "working system" inside at the office, it does not mean that the same system behaves similarly outside under other conditions. For instance, the tests performed inside at the office included only one fixed distance between the markers and the cameras. In addition, an outdoor docking scenario includes a visual scene with more complex features that change more rapidly.

Now, some preparations were needed to fill the gap between experimental measurements and meaningful results. It is also important to explain why our assumptions on the way seem reasonable as they may induce some error in the ground truth measurements. With some knowledge on how the experiments are prepared, we move on to introduce what each experiment includes.

8.2 Experiment 1

The first experiment covers a full docking and undocking operation. That is, the USV starts at approximately 12 meters from the dockside and moves straight towards the dockside where the markers are located. When the USV gets close to the quay, it stops and starts to move in the opposite direction until it reaches a distance of 10 meters relative to the quay. The blue curve in Figure 8.4 represents an approximation of the path seen from a top-view. The main motivation for this experiment is to evaluate the accuracy of the position measurements for different ranges between the camera and the markers. In order to do this, we have used a lidar for ground truth verification. As it is quite time consuming to measure the fixed distance between the left camera and the markers in the point cloud accurately, it was decided to reconstruct ground truth points one time for each second (1 fps) for comparison with the position measurements produced by the camera system. For each sample, we have both monocular and stereo measurements available (at least almost). This allows us to compare the accuracy between the pose estimation tech-

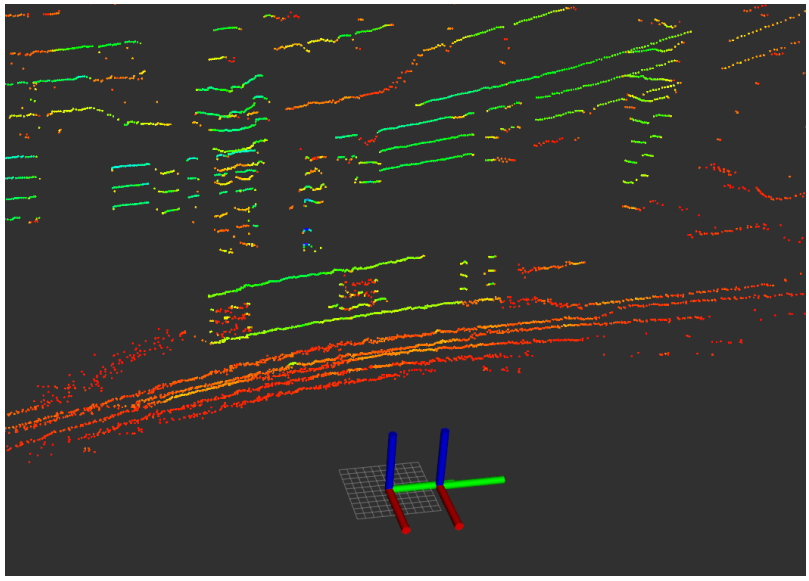


Figure 8.3: Point cloud map of a docking scenario produced by the lidar. The right coordinate system is fixed with respect to the lidar, while the left coordinate system represents the left camera coordinate system. With some prior knowledge of the scene, the markers can easily be identified in the map. Pay attention to the correspondence between in the markers in the point cloud and the markers in the right part of Figure 8.5.

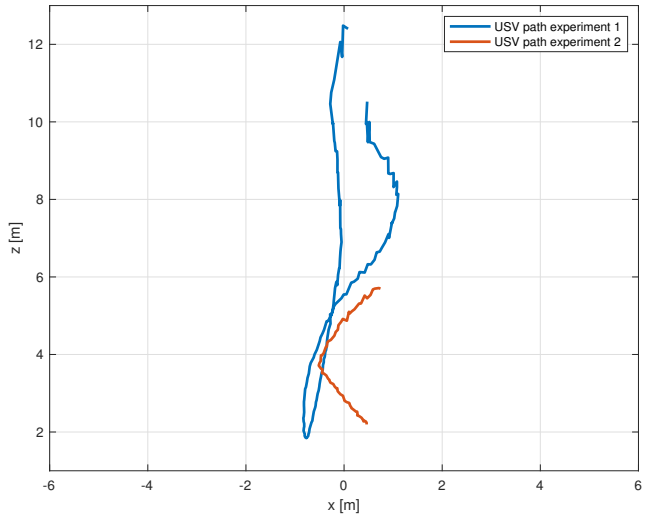


Figure 8.4: Top view of the USV paths from experiments 1 and 2. The paths consist of x-z points between the left camera and marker *m1* measured with monocular pose estimation. E.g. it does not precisely represent the ground truth paths.

niques at each timestamp. To evaluate the accuracy we simply find the difference between distance measurements provided by the lidar (e.g. ground truth) and the camera system for each timestamp. All relevant measurements for this experiment are located with respect to the left camera. This includes the ground truth lidar data as explained in the previous section.

We have chosen to focus on one marker only for this experiment. That is, the left marker with the name tag *m1*. We discovered that the CNN produced a significant amount of misclassifications when it tried to detect the right marker (e.g. *m2*). For some reason, it seems like the CNN was not able to distinguish between the markers when it tried to detect *m2*. This can indeed lead to dangerous consequences. For instance, if both CNNs do the same misclassification at once, the stereo algorithm will output 3D position measurements of the wrong marker! Unfortunately, we were not able to localize the cause of unexpected behaviour. We did not find any wrong labels in the training data. Furthermore, there is a good balance between training examples for the two classes in the custom dataset. At last, we do not consider the ArUco pattern of the right marker *m2* to be considerably harder to detect compared to *m1*. This makes the observed phenomena a bit mysterious, especially since the other marker performed well under the experiments. This problem also illustrates that the focus for this master thesis has not been to perform extensive data collection for the custom dataset. As we observed that we achieved reasonable measurements from at least one of the markers, we did not prioritize to re-train the CNN more robustly. It is a future task to collect a broader dataset to robustly detect and distinguish between the markers, even at longer distances.

We also added raw camera measurements without ground truth comparison. This helps us to visualize and test if the 3D reconstruction techniques produce any significant outliers. Recall that the ground truth measurements are only compared with camera measurements at 1 fps meaning that there may be hidden some outliers in between. In fact, one can say a lot about how trustworthy the measurements are with some prior knowledge about the docking scenario. For instance, if we know that the USV moves towards the docking with constant speed and the camera system produces a corresponding non-continuous path, we can with a high degree of confidence say that the camera system is not producing trustworthy nor stable measurements.

8.3 Experiment 2

Experiment 2 shares some commonalities with experiment 1, but have other areas of focus. It covers a pure undocking operation where the USV starts at approximately 2.5 meters and reverse the USV slowly away from the docking. The red curve in Figure 8.4 represents an approximation of the path seen from a top-view. In contrast to experiment 1, it focuses on a comparison between the distance measurements of the two markers. To be able to do that, we have limited the range between the USV and the dockside (approximately 6 meters) to achieve reasonable measurements from both *m1* and *m2* at the same time. E.g. we are not supposed to test the detection limit in this experiment.

Although the USV still points towards the markers, the heading angle (between the markers and the cameras) is not the that narrow compared to experiment 1 meaning that the operation can be seen as a sideway undocking operation to some extent. This lets us test whether the algorithms are able to handle different camera angles or not. Since the markers are placed along the dockside with a position offset mainly along the x-axis relative to the cameras, we can also compare directly if some of the markers produce more accurate measurements than the other. It also allows us to discuss if it would be reasonable to fuse the measurements from the two markers to achieve more robust measurements during the docking operation.

Furthermore, the USV moves more slowly, approximately at half speed compared to the first operation. By this, we can compare whether distance measurements are sensitive for fast operation or not. E.g. will we achieve more accurate measurements with a slow operation? This may be especially interesting to test for the stereo algorithms as this algorithm solely depends on how well the stereo pair is synchronized during dynamic operations. Of natural reasons, a static scene will not reflect how well the stereo pair is synchronized.

As mentioned, experiments 1 and 2 have several things in common. For instance, both experiments obtain ground truth values at 1 fps. And we still use the Euclidean distance between the left camera and the markers as our measurement objective. Further, we do still compare both stereo and monocular 3D reconstruction techniques for each timestamp. At last, as with experiment 1, we added raw camera measurements to observe if the methods produced any significant outliers.

8.4 Experiment 3

An optimal navigation system should not be limited to produce position measurements only. It should produce orientation measurements as well. In this context, we are especially interested in heading measurements since this gives us valuable information about which direction the vehicle is heading. This experiment is, in fact, an extension of experiment 2. It is recorded at the same time as experiment 2 but look into other measurements. That is, we want to test if the monocular 3D reconstruction algorithm produces trustworthy and stable heading measurements as well.

Unfortunately, the point cloud provided by the lidar is not able to provide orientation measurements (e.g. yaw, pitch and roll) by itself. One could however reconstruct orientation measurements from cartesian triangles (measured in the point cloud) and apply simple geometry formulas. This would however be time-consuming and one may question how accurate these reconstructed measurements actually are compared to the ground truth. Therefore, we did not embrace ourselves with this task. Although we do not have a ground truth for comparison, the measurements can still give us valuable insight. Especially since we have orientation measurements of two markers available at the same time. It all boils down to compare the relative offset between the heading measurements of the two markers. Intuitively, we expect the offset between the heading measurements to be more or less

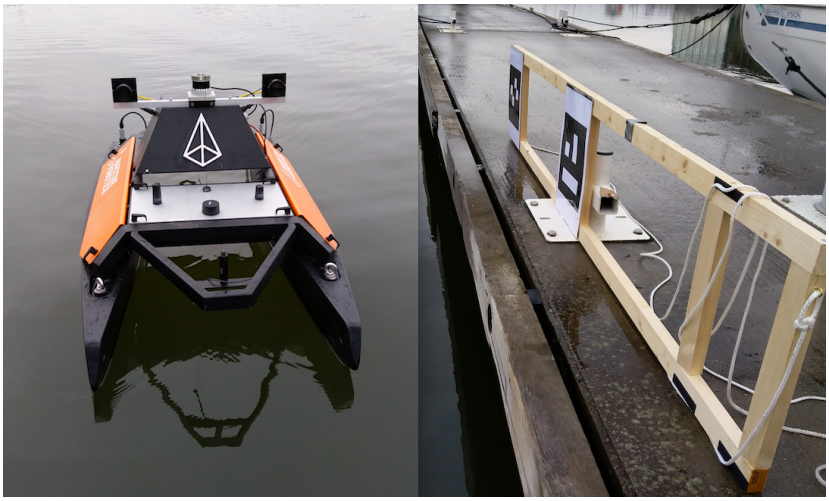


Figure 8.5: Visual shots of the two main components used during the experiments. To the left, the vision system armed on an Otter USV. To the right, the marker configuration used as the reference system to obtain position measurements.

static as the markers are fixed to each other (and the quay) during the whole experiment. If they do not, it should give us an indication of untrustworthy measurements.

We end this section with a few remarks about the third experiment. As mentioned in the previous section, the monocular measurements are located with respect to the marker frame. E.g. the heading measurements are located with respect to the marker. As the stereo algorithm does not produce orientation measurements, it is not done a comparison across the two different 3D reconstruction techniques for this experiment.

Chapter 9

Results and Discussion

The previous chapter explained the objective for each experiment. This chapter introduces the results followed up by a discussion for each experiment. Other remarks will also be included before we sum up the chapter.

9.1 Experiment 1

Results

Figure 9.1 shows the Euclidean distance measurements of both stereo and monocular camera measurements in addition to ground truth values obtained by the lidar sensor as a function of time. Only marker *m1* is used to obtain Euclidean distance measurements. The measurements is sampled at 1 fps. If we refer to the ground truth trajectory, the USV starts at 12 meters range from the docking and moves with a speed of approximately 0.7 m/s. Beyond 12 meters, camera measurements (especially for stereo vision) tend to occur more sporadic. To achieve a reasonable number of subsequent measurements, we decided to only include data up until 12 meters. Observe that some camera measurements are missing. At the third sample (after 3 seconds), the USV moves in the absence of both stereo and monocular measurements available. Also when the USV gets too close to the markers, it is not able to find the stereo correspondences from both camera views anymore and consequently, we lack stereo measurements during some samples between 15 and 20 seconds. Figure 9.2 complements the camera measurements from Figure 9.1. Thus, all camera measurements (sampled at 5 fps) are plotted to get a better picture of how stable the camera measurements are. As the reader may observe, no ground truth lidar data is provided in this plot.

To get more insight into how accurate the camera system actually is, we attached a Figure 9.3 that shows the error between the ground truth lidar data and the camera measurements as a function of the range (between the marker and the left camera). We used the ground truth range as reference. We divided it into two plots, where the first part shows the

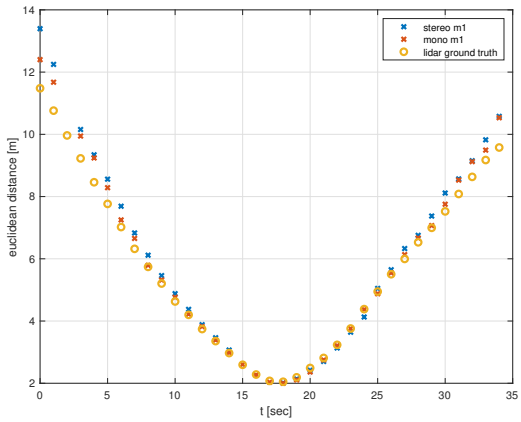


Figure 9.1: The Euclidean distance measurements provided by the stereo technique (blue cross), the monocular technique (red cross) and the ground truth lidar (yellow circle) as a function of time. The USV starts at approximately 12 meters and docks until it reaches 2 meters from the markers. Then it starts the undocking operation by reversing the path until it reaches approximately 10 meters.

error during the docking operation until the USV stops near the dockside and the second part where the USV starts to reverse until it reaches approximately 10 meters. Another option would be to merge all measurements into one plot. However, we thought it may be confusing as we could potentially have several error values in the same range.

We summarize the results from experiment 1 with some key numbers from Figure 9.3:

- At 11.55 meters range, the stereo measurement reached the highest error, **1.84 m**. At 2.02 meters range, the stereo measurement reached the lowest error, **2.58 cm**.
- At 9.6 meters range, the monocular measurement reached the highest error, **0.92 m**. At 4.35 meters range, the monocular measurement reached the lowest error, **0.52 cm**.

Discussion

The main motivation with the first experiment was to investigate how accurate localization estimates the camera system can deliver across different ranges from the docking. That is, given the trained CNN and the chosen camera configuration for the experiments, how accurate measurements can we achieve at 2 meters? 5 meters? Or 10 meters? If we study the plots in Figure 9.3, we can see that both stereo and monocular measurements produce quite accurate measurements during the first couple of meters (i.e. 2-4 meters). Here we have less than 10 centimeter error. After 4 meters, the error tends to increase decently, especially for the stereo measurements. However, the monocular measurements tends to deliver acceptable measurements (less than 25 cm) up until the USV reaches 6-7 meters. Figure 9.6 from experiment 2 provides the same degree of accuracy for the monocular

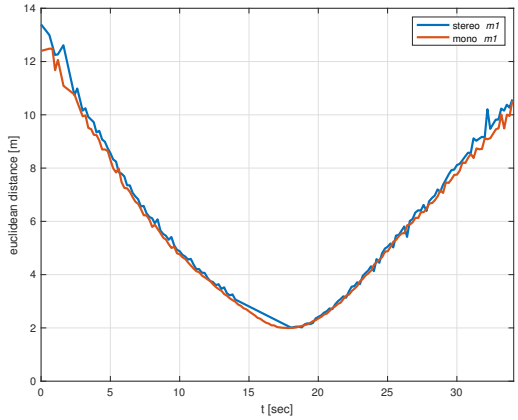


Figure 9.2: This Figure complements Figure 9.1 with camera measurements hidden between 1 fps samples. It shows up to five measurements per second for both the stereo (blue) and monocular (red) method. This time without lidar ground truth data. It reveals how stable measurements the camera system can deliver and if there are any significant outliers.

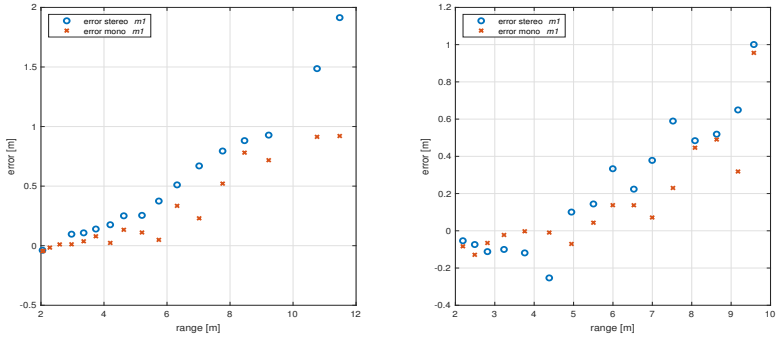


Figure 9.3: The error between the ground truth lidar data and the camera measurements as a function of the Euclidean distance (between the left camera and marker *m1*). Both camera techniques are compared to ground truth. The left plot shows the error until the USV docks along the quay and the right plot shows the error from the USV starts the reverse operation until it reaches approximately 10 meters.

measurements. As mentioned in the result part, camera measurements beyond 12 meters tend to occur more sporadic. That is, camera measurements became less and less available, between 12 and 20 meters. We also observed that the monocular measurements tend to last a bit longer in terms of the range from the quay. This is not very surprising as it may be somehow harder to detect markers from two camera views at once (e.g. stereo), in contrast to only one (e.g. monocular), if the USV is far from the docking. Across any range, we found the monocular method to beat the stereo method in terms of minimizing error.

The reader may wonder why stereo vision performed fairly poor compared to monocular vision. There might be a complex composition of reasons behind, but we will still try to highlight some relevant candidates. Primarily, there are more critical components involved in a stereo setup and therefore, more components may fail. The first relevant candidate is low-latency synchronization between the camera measurements. We have already implemented hardware triggering to ensure that the left and right camera image are captured (approximately) at the same time. However, if we refer to Figure 9.4, the hardware triggering only ensures that the camera streams input for each object detection model is synchronized. We will try to figure out what may affect the synchronization before the images are merged into a stereo pair and triangulation is performed. Firstly, observe that the object detection models run in parallel and do not communicate with each other. As there will be slightly varying processing time for the CNN (e.g. inference time) to detect, we cannot guarantee that a bounding box predicted from the left camera view is produced at the same time as the one from the right camera view. This is however not a critical problem as long as the CNN produces estimates with high frame rate.

Unfortunately but also not very surprisingly, the computing unit (Nvidia Jetson Xavier) was not able to run two YOLOv3 instances in real-time (e.g. 20-30 fps). The other obvious problem was that the camera driver was only able to deliver an acquisition frame rate around 8-9 fps for two camera streams with 1280×1024 resolution. In other words, a CNN with higher fps may produce the same position measurement consecutively (since it has the same image available). Also note that simultaneously streaming of lidar data may affect as well. Interestingly, we experienced heat issues when launching two YOLOv3 instances at once. The computer tends to shut off, most likely triggered by the TTP operating temperature reaching 80 $^{\circ}C$ (see Table 2.1 in [63]). The solution was to remove some heavy processing that was not utilized anyway (due to fairly low acquisition frame rate). Thus, we forced the CNN to produce measurements at 5 fps by using a sleep function in ROS. This also lets us log measurements with regularly intervals. However, this choice may affect synchronization slightly. The reason is that the camera driver delivers nearly two images at the time the CNN have produced a bounding box and is ready to capture a new image. As we have not implemented some code to handle whether the CNNs running in parallel are processing the same image or not, we cannot be sure that they actually produce a suitable bounding box pair for the stereo algorithm or not. And consequently, if the scene is changing rapidly (as is the case in a docking scenario), the stereo vision algorithm will most likely induce some error.

To make it easier to compare the stereo and monocular vision techniques, the methods

share many of the same calibration parameters. They share the same intrinsic parameters. Stereo vision does however contain a transformation matrix, which relates the left camera to the right camera. If the calibration is done properly, the camera should be able to deliver 3D point estimates with high precision. The assumption is that the cameras maintain their fixed position and orientation relative to each other. We observed under the tests that this assumption may not be totally correct. For instance, if we moved slightly on the ethernet cables connected to the cameras and we experienced that the cameras were slightly moved out of its fixed position relative to the camera house (e.g. camera protection). It is conceivable that the cameras may not have maintained their relative offset in position and orientation between the calibration and the experiments. It is hard to quantify how significant this aspect may affect the final measurements, but we believe it will only impact the accuracy of the measurements slightly. This is because we experienced the cameras to only shift its position with a fine-grained offset. One can imagine that if the USV hits the quay, the cameras may change their relative pose slightly. The solution is obviously to build a more robust camera rig that handles even more harsh environments. This also includes compensating for natural frequency, i.e. the frequency at which the USV tends to oscillate in the absence of any driving force. As the monocular vision technique is not dependent on how the cameras are related to each other, they are in fact less sensitive for a physical perturbation. This aspect reflects, together with the strict synchronization requirements, that the stereo vision system includes more aspects that can affect the position measurements.

If we were to improve the accuracy of the system somehow, it is a good choice to increase the resolution. A 1280×1024 image does indeed provide sufficient accuracy for localization tasks near the docking (e.g up to 6-7 meters), but it would be interesting to see if the camera system would provide even better accuracy at this range and also reduce the gap for distances above 6-7 meters. The drawback is of course higher bandwidth to handle which may affect how frequently we can push out camera measurements.

In addition, we are also limited to how well the object detection model recognizes the markers. Fortunately, the detection model seems to recognize the markers well up until 15 meters, at least under optimal scenarios. The problem is however that the CNN struggled to distinguish between the markers properly, especially the second marker *m2* was somehow hard to recognize at times. Based on the results from the project thesis, we thought the CNN was able to perform with considerable fewer misclassifications than our results. This shows that operational data tend to make the detection task harder, compared to what the test data from the custom data set can offer. Specially since we consider the custom data set as quite homogeneous. Recall that the detection model is constrained by the training data. That is, if the variation and complexity of the operational environment are not entirely reflected in the training data, the model will struggle to detect and distinguish between the markers in operational use. This reflects how important it is to not underestimate the training scheme of a CNN.

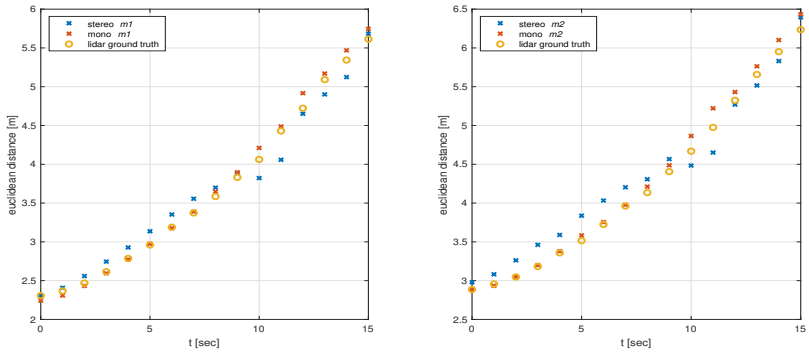


Figure 9.4: The Euclidean distance measurements provided by the stereo technique (blue cross) monocular (red cross) and the ground truth lidar (yellow circle) as a function of time. The left plot shows Euclidean distance measurements relative to marker *m1*, while the right plot shows the Euclidean distance measurements with respect to marker *m2*. It shows a pure undocking operation of an USV until it reaches approximately 6 meters from the markers.

9.2 Experiment 2

Results

In the same manner as experiment 1, we have chosen the same setup of figures to visualize the results from experiment 2. That is, Figure 9.4 shows Euclidean distance measurements provided by the left camera versus ground truth lidar data as a function of time, now with two plots. The left plot shows the camera measurements relative to marker *m1*, while the right plot shows the measurements relative to marker *m2*. Also observe that this undocking operation is significantly slower than the one in experiment 1. Here, the USV drives at approximately 0.2 m/s speed. As with Figure 9.1 and 9.2, Figure 9.5 complements Figure 9.4. Interestingly, we can easily observe an outlier in the right plot of Figure 9.5. Also here, no ground truth lidar is included in the plot.

Figure 9.6 shows the error between the ground truth lidar data and the camera measurements as a function of range, separated into two plots depending on which marker is used. In contrast to Figure 9.3, Figure 9.6 shows all the measurements during the time line each plot.

We summarize the results from experiment 2 with some key numbers from Figure 9.6:

- Within a range of 4 meters, monocular measurements relative to both marker m1 and *m2* are limited by an error of **6.5 cm**.

- Within a range of 4 meters, stereo measurements relative to marker *m1* is limited by an error of **18.5 cm** and stereo measurements relative to marker *m2* is limited by an error of **31.8 cm**.

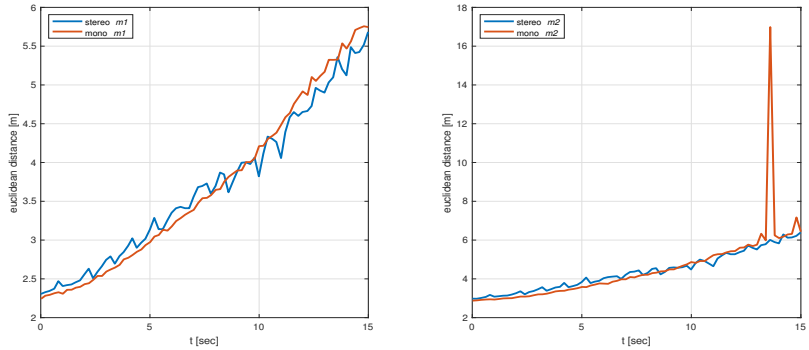


Figure 9.5: This Figure complements Figure 9.4 with camera measurements hidden in between 1 fps samples. It shows up to five measurements per second for both camera measurements. As with experiment 1, lidar ground truth data is not included. The left plot shows the Euclidean distance measurements relative to marker *m1*, while the right plot shows the Euclidean distance measurements with respect to marker *m2*. The Figure reveals how stable measurements the camera system can produce. Pay attention to an outlier produced by the monocular method in the right plot.

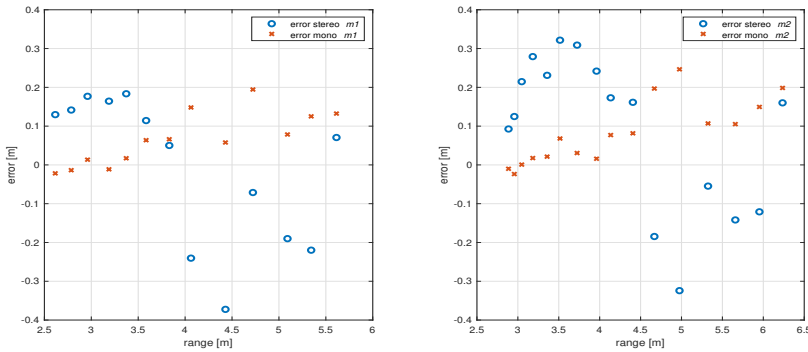


Figure 9.6: The error between the ground truth lidar data and the camera measurements as a function of the Euclidean distance. Both camera techniques are compared to ground truth. The left plot shows the error with respect to marker *m1*, while the right plot shows the error with respect to marker *m2*. In contrast to Figure 9.3, both plots in this Figure represents the same timeline.

Discussion

This experiment investigates how the camera system handle another type of camera view when the USV docks. That is, the heading angle between the markers and the cameras are not that narrow compared to in experiment 1. If we compare both plots in Figure 9.3 (up until 6 meters) and the left plot in Figure 9.6, it is hard to see that the accuracy of the measurement is affected by the camera view significantly. This includes both camera techniques. It is however not very surprising as the heading angle relative to marker *m1* is no more than 6-7 ° the first couple of meters before it increases up until 15 ° during the operation (see Figure 9.7). Most likely, we will need more extreme angles (e.g. higher heading angle) to get more insight into how the camera system handles a sideway docking operation where the USV is supposed to dock along the dockside.

In addition, this experiment includes two markers used as reference points at once. Since the markers are placed along the dockside mainly along the x-axis relative to the cameras, we can also compare directly whether one of the markers produces more precise measurements than the other. If we study both plots in Figure 9.6, we can observe that the monocular vision technique produces measurements with more or less the same degree of accuracy for both markers. The stereo vision technique does however produce slightly less accurate distance measurements for marker *m2*, especially during the first couple of meters (e.g. up to 4 meters). If we look at Figure 9.5, it is easy to observe that the stereo vision method produces significantly less stable measurements. With the support of ground truth lidar data from Figure 9.4, we know that the USV moves at nearly constant speed and consequently, we expect the camera system to produce measurements to follow this straight line. In contrast, the stereo vision method produces a zigzag path/pattern (see Figure 9.5) which is less trustworthy. The zig-zag pattern can however be stabilized/improved with a filtering technique, such as the extended Kalman filter (EKF). For direct comparison with the first experiment, we can study the last part of Figure 9.2 which produces fairly less noisy measurements during the undocking.

From the left plot in Figure 9.5, one can also observe that the stereo measurements tend to repeatedly alternate between fairly accurate measurements and less accurate measurements (almost like a sawtooth wave). We might suspect this behavior to be related to strict synchronization requirements as already discussed. Recall that the CNNs wait for a 200 ms interval before they grab a new image (if available), independent of the inference time. However, since the CNNs do not communicate with each other, it cannot be guaranteed that the grabbed image pair is taken at the same time. An unfortunate consequence might be that the stereo algorithm receives a pair of images, which are not well synchronized periodically. As discussed in experiment 1, there might be other reasons why the stereo vision method performs worse such as maintenance of extrinsic parameters. To not repeat ourselves, we will not discuss these aspects further.

Compared to experiment 1, where the USV moves at approximately 0.7 m/s speed, the USV in this experiment undocks slightly more gently with approximately 0.2 m/s speed. We want to investigate whether the camera system is sensitive to fast operations or not. Although the USV is not moving particularly fast during the first operation (0.7 m/s), it may represent a typical speed of a docking operation. In contrast, a docking with 0.2 m/s

may be quite slow if the process starts 10-15 meters from the docking station. As already observed, we could not find any notable change in the accuracy of the camera measurements between experiments 1 and 2. Therefore, we may conclude that the change of speed did not affect the measurements significantly. As with the camera view, we may need to increase the speed significantly to observe some notable differences.

We allow ourselves to discuss how the measurements obtained from marker *m1* and *m2* can be fused. Recall that multiple markers in the scene can be used to produce redundancy (e.g. duplication of critical measurements). Since we assume the fixed position offset between the markers to be known, we can simply measure the relative position between the left camera and the two markers and then, transform one of the measurements into the other. From Figure 2.5, one can observe that the camera system developed by Boston Dynamics uses two ArUco markers in a pair for close-range operations where the robot needs to know its relative pose with high precision in order to operate successfully. This reflects that the idea is commonly used by autonomous robots for self-navigation in GPS-denied environments.

At last, we can easily observe an outlier in the right plot of Figure 9.5. That is, an outlier produced by the monocular vision technique with respect to marker *m2*. We suspect that the outlier is caused by a misclassification from the CNN. As outlined in the previous chapter, the CNN produced a significant amount of misclassifications of marker *m2*, especially during experiment 1 where the USV was tested at a longer range.

Interestingly, we only experienced an outlier produced by the monocular technique. At that timestamp, we would rather trust the stereo measurement. In this context, a question that may arise is, will we achieve overall better accuracy if the stereo and monocular measurements were fused? Based on the results, it would in most cases not be very likely. Briefly said, in most cases, a fusion at each timestamp will perform better than the stereo measurement. At the same time, it will also perform worse than the monocular measurement. We will therefore not recommend this procedure, based on the experimental data available. Then, it would rather be reasonable to invest more time in improvements of the camera techniques, followed up by new tests and more experimental data. This will provide us with more insight into which camera method to trust in.

9.3 Experiment 3

Results

Figure 9.7 shows the heading measurements produced by the monocular pose estimation technique as a function of time. The blue curve in Figure 9.7 represents the heading measurements relative to marker *m1*, while the red curve shows the heading measurements relative to marker *m2*. As mentioned in the previous chapter, no ground truth data is provided for this experiment. In fact, the experimental data from this experiment is recorded at the same time as experiment 2. Except for some outliers, Figure 9.7 shows a slowly varying offset between the heading measurements for each timestamp, especially during

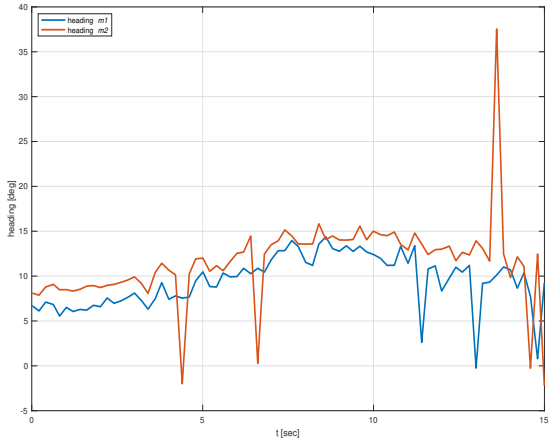


Figure 9.7: This Figure shows the heading measurements produced by the monocular vision technique as a function of time. The blue graph represents the heading measurements relative to marker *m1*, while the red graph represents the heading measurements relative to marker *m2*. The measurements are recorded at the same time as the measurements from experiment 2.

the first 10 seconds. If we refer to Figure 9.4, this corresponds to an undocking until the USV reach nearly 5 meters.

Discussion

This experiment aims to investigate whether the monocular heading measurements produce a static offset between the markers or not. We do not emphasize the accuracy of the heading measurements in itself as we have no ground truth orientation measurements available. Except for some outliers, we can observe that the measurements tend to follow each other with a fixed offset, especially during the first half of the undocking operation (see Figure 9.7). This coincides with our intuition, as the markers are fixed to the same piece of wood on-board a very slow-varying dockside. With some prior knowledge of the operation and the obtained offset between the markers, we may assume the heading measurements to be fairly trustworthy. On the other side, both measurements (relative to *m1* and *m2*) produce some outliers which is not desirable. We already know that the USV does not undock with very quick maneuvers (0.20 m/s constant speed), so it is unlikely that the ground truth heading measurements produce a corresponding zigzag path (see Figure 9.7 for reference). For this reason, we are cautious when discussing how trustworthy these measurements actually are.

The measurements from Figure 9.7 shows that the heading measurements relative to marker *m1* does not produce any significant outliers until 11 seconds have passed. At this time, two outliers from *m2* are already obtained. We have already discovered that the CNN performs a lot of misclassifications for *m2*, which propagates through the vision pipeline

and most likely produces outliers. If we were to trust some of the markers with the trained CNN, we would most likely choose marker *m1*. Discussing this in detail is however out of scope as we have no ground truth for benchmarking.

Also note that the monocular technique produces an outlier in the right plot of Figure 9.4 at the same time instance (after 13-14 seconds). E.g. the red graph corresponding to *m2*. In other words, we may assume a strong correlation across the measurements for each technique. For instance, it is unlikely that the monocular technique produces an accurate position measurement and at the same time an inaccurate orientation measurement (i.e. heading) or the opposite.

9.4 Other Remarks

In this section, some observations not directly connected to a specific experiment but still of importance, will be introduced.

First of all, it has been proven that CNNs are vulnerable to adversarial attacks. In general, adversarial attacks are inputs to machine learning models that an attacker intentionally designed to cause the model to make wrong predictions [65]. In the image classification domain, adversarial attacks refer to subtly modifying an original image in such a way that it looks almost the same for a human eye. Since deep CNNs are used in safety-critical tasks, such as autonomous vehicles, to recognize potential threats or localize itself, the consequences may be catastrophic if CNNs are fooled by adversarial attacks. For example, an autonomous car may use deep learning techniques to recognize road signs. This can lead to dangerous outcomes if the CNN is fooled by an adversarial attack, and thereby not able to detect the "STOP" sign along the roadside such that the car is still going.

In terms of a docking operation, an unmanned ship may put itself in a dangerous situation if the ship solely trusts on deep CNNs to detect reference markers during the last phase of the docking. Not only adversarial attacks due to disruptive pixel perturbation may be feasible from an attacker's perspective; physical disruptions of the markers are also possible, which may lead to misclassifications or no classifications at all. At last, nature itself may produce "illusions" of the markers. As a concrete example, reflection from the water in a docking operation may introduce double pairs of markers, one at the dockside and a corresponding one in the water. Figure 9.8, taken from the training data, illustrates this clearly. Fortunately, the CNN was able to correctly ignore these candidates, as far as we know. However, if such false positives were produced, they can be filtered by a geometric evaluation. As we have prior knowledge of the relative location of the markers, one may ignore false positives in the water based on their pixel position. Thus, if two markers of the same class are predicted by the CNN at once, choose the upper one in the image.

In a sensor fusion scheme, it is desirable to know how much you can trust each sensor individually, before any filtering and/or fusion is performed. For this reason, this thesis focuses solely on how well different vision-based 3D reconstruction pipelines performs, in terms of a positioning error, without filters included. We do however allow ourselves to discuss briefly the next step, which should include some filtering. Clearly, the camera system produces outliers, as shown in Figures 9.4 and 9.7. Thus, unfiltered positioning

Figure 9.8: Reflection from water may confuse the detector. Specially the left marker can be challenging as it is nearly symmetric.

measurements from the cameras should not be fed into a feedback control system, as it may lead do unfortunate ship maneuvers. A good first step is the extended Kalman filter [66], frequently used in navigation systems, to remove noise and detect outliers in raw sensor measurements represented as time series. If we refer to Figure 2.1, our implementation can be seen as the green block "Autodocking" inside the navigation system block. Then, a filter (i.e. the extended Kalman filter) is used in the outer block "State Estimator" to produce more robust position measurements, which again can be used in a sensor fusion scheme.

Chapter 10

Conclusion

This Chapter concludes the thesis. That is, summarizing the most important findings and presenting some future challenges. But first, an overview of the core of the thesis is given.

10.1 Overview

This thesis has investigated how a data-driven, locally vision-based positioning system can be used in docking operations. When GPS signals are not available or redundant position estimates are desirable, an autonomous vehicle can obtain navigation information with the proposed positioning system mounted on-board the vehicle, if visual markers (used as reference points) are placed on the dockside.

The final vision-based pipeline employs a complementary part-based approach that uses a combination of data-driven deep learning methods, and at the same time using traditional computer vision techniques when the scope and complexity of the task are reduced. Figures 7.2 and 7.1 summarize the high-level pipelines used in this thesis, which represent methods to perform 3D reconstruction of the scene from single-view and stereo-view, respectively. The first part of the pipeline investigates how a flexible stereo camera system is integrated into the perception pipeline. Then, the different steps needed to achieve fine-tuned object recognition for a specific application, such as marker detection, are reviewed. Thus, a custom dataset is collected and prepared for supervised methods within the field of deep learning. Further, a large general dataset (ImageNet) was applied to learn general patterns and to reduce overfitting issues. With the prepared custom dataset and the pre-trained model parameters as a basis, the deep CNN was trained, validated and tested. As the reader may recognize, there is more to object recognition than neural networks. Intuition about the quality, amount, relevance and variety of the data used for the learning task may be just as important. Since humans, in general, have no idea on the specific decision process throughout the layers in deep CNNs (e.g. black box), intuition about input data and corresponding output classifications is crucial.

Given a fine-tuned object recognition model, producing accurate bounding boxes around the markers, the complexity of the 3D reconstruction task is reduced dramatically. Instead of performing 3D reconstruction directly on the whole image, the task is now reduced to perform 3D reconstruction on a small sub-part of the whole image, which also reduces the chances of producing outliers. Figures 7.11 and 7.10 illustrate the core of the algorithms used in this thesis, to obtain 3D information from single-view and stereo-view, respectively. The stereo reconstruction algorithm receives a pair of bounding boxes around the same marker and detects the marker corners in each marker. Since the corners are produced in the same order, the matching strategy is simplified significantly. It allows us to create stereo pairs directly by its index and pass them into a triangulation algorithm along with projection matrices to recover 3D position of the marker with respect to the camera system. The monocular vision design shares many commonalities but uses object priors in the last step of the pipeline to obtain the 3D pose of the marker using a PnP solver.

10.2 Findings

Throughout section 9.1, 9.2 and 9.3, different aspects of a vision-based positioning system have been investigated, with a special focus on realistic docking-operations from outdoor experiments. We found the camera measurements from experiment 1 to be quite promising, in terms of accuracy compared to ground truth lidar measurements, especially with monocular camera design. More specific, the monocular camera system produced acceptable euclidean distance measurements up until 6-7 meters (less than 20 cm error within this range). A camera with an even higher resolution than 1.3 megapixels may be desirable for accurate positioning measurements above this range. This will however require more bandwidth. The gap in accuracy between the stereo vision design and the monocular vision design may be caused by several factors. For instance, the stereo setup is not entirely synchronized throughout the pipeline, and as a consequence, it tends to alternate between fairly accurate and less accurate measurements. Hence, a suitable synchronization algorithm should be implemented in the future, to control that the calculated stereo pairs are synchronized properly before the final triangulation algorithm is performed to recover 3D information.

We also noticed that the CNN struggled to classify the second marker *m2* properly, especially at long distances. This may be caused by a homogeneous custom data set, which the CNN is trained on. We believe that the variation and complexity of the operational environment are not entirely reflected in the training data, and as a consequence, the operational data tends to make the detection task harder. This reflects how important it is to not underestimate the training scheme of a CNN. Therefore, a more comprehensive dataset for more robust detections should be collected, so the CNN is able to handle challenging weather conditions as well.

While experiment 1 focuses on one single marker to investigate the detection limit as a function of the range, experiment 2 includes two markers to focus on direct comparison between the markers for short-range position measurements. The monocular design produced more or less the same degree of accuracy for both markers, while the stereo vision technique produced slightly less accurate distance measurements for marker *m2*, espe-

cially during the first couple of meters. Furthermore, some changes in speed and heading angle, compared to experiment 1, did not affect the measurements noteworthy. We may need more extreme angles or change of speed to observe any notable difference.

At last, we experienced that the monocular pose estimation scheme produced fairly trustworthy heading measurements in experiment 3. Except for some outliers, the heading measurements (of the left camera relative to marker *m1* and *m2*) tend to follow each other with a fixed offset. This coincides with our intuition, as the markers are placed at the dock-side with a fixed offset. Due to the obtained outliers, unfiltered localization measurements should not be applied in a feed-back control system directly. We propose to use a suitable filter, i.e. the extended Kalman filter [66], to produce more robust estimates.

To summarize, a complete vision-based positioning system has proven to work well for outdoor docking operations as shown in the experiments. Yet, more comprehensive tests should be performed, to fix bugs and increase the robustness, before a more complete prototype converges towards commercial use. We do hope however that the practical work, as well as all the acquired knowledge throughout the thesis, can be used to supplement the traditional navigation system for safety-critical docking operations in near future.

10.3 Future Work

This thesis covers a broad spectrum of research areas as well as a lot of practical work, and it has not been possible to go into detail in all topics covered. Not surprisingly, some challenges have been discovered throughout the thesis. Hence, some suggestions for future work can be summarized by the following list.

- Include a RTK GPS (instead of a lidar) in the pipeline to verify not only position estimates but also orientation estimates.
- Include fisheye cameras (instead of wide-angle camera) to capture even more of the scene. Also, add higher resolution to the camera to observe if camera measurements from middle/long range are improved.
- Implement synchronization algorithms to achieve low latency between stereo pairs throughout the whole pipeline.
- Perform post-processing techniques (e.g. filtering) to obtain more robust position estimates, without outliers.
- Include future design improvements in the pipeline as reviewed in subsection 7.4.5.
- Collect a more comprehensive dataset for more robust detections, which helps the CNN to handle more harsh and challenging weather conditions.
- Test the implemented system more comprehensive on outdoor docking operations. I.e. execute the same USV path several times (repeatability) using the same methods and conclude the outcome of the independent test results.

- Include the camera measurements in a closed-loop control system for increased redundancy and reliable control of a vehicle during the (autonomous) docking phase.
- Implement a path planning algorithm suitable for camera-based localization tasks. For instance, [67] presents a path planning algorithm for underactuated vehicles with a limited field of view, in particular for vehicles that cannot control its sway motion and can only move forward. As the test platform Otter USV is an underactuated vehicle, this paper is indeed relevant for my work. Recall that a single-view and/or stereo vision system provides only a limited field of view, even with fisheye lenses, and the working system assumes the markers are visible in the camera view.

[sorting=nyt,firstinits=true]biblatex

Bibliography

[1] "The portable usv system," 2018, accessed: 2019-04-04. [Online]. Available: https://maritimerobotics.com/mariner-usv/otter/

[2] "Rolls-royce to supply first automatic crossing system to norwegian ferry company fjord1," 2016, accessed: 2019-03-12. [Online]. Available: https://www.rolls-royce.com/media/press-releases/2016/18-10-2016-rr-to-supply-first-automatic-crossing-system-to-norwegian-ferry-company-fjord1.aspx

[3] "Verdens første helt autonome fergeseilas gjennomført - teknologien er 100 prosent klar," 2018, accessed: 2019-03-12. [Online]. Available: https://www.tu.no/artikler/verdens-forste-helt-autonome-fergeseilas-gjennomfort-teknologien-er-100-prosent-klar/452610

[4] Aqel, M. et al., "Review of visual odometry: types, approaches, challenges, and applications," 2016, accessed: 2019-03-12. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5084145/

[5] "Gnss error sources," *Novatel*, 2016, accessed: 2019-03-12. [Online]. Available: https://www.novatel.com/an-introduction-to-gnss/chapter-4-gnss-error-sources/error-sources/

[6] "Frykter at russlands gps-jamming kan føre til ulykker," 2016, accessed: 2019-03-12. [Online]. Available: https://www.nrk.no/finnmark/frykter-at-russlands-gps-jamming-kan-fore-til-ulykker-1.14292013

[7] D. Watson and D. Scheidt, "Autonomous systems," *Johns Hopkins APL Technical Digest*, 2005, accessed: 2019-03-12. [Online]. Available: https://www.jhuapl.edu/techdigest/TD/td2604/Watson.pdf

[8] "Autonomous ship project, key facts about yara birkeland," *Kongsberg Maritime*, 2017, accessed: 2019-03-12. [Online]. Available: https://www.km.kongsberg.com/ks/web/nokbg0240.nsf/AllWeb/4B8113B707A50A4FC125811D00407045?OpenDocument

[9] "Volvo penta unveils pioneering self-docking yacht technology," *Volvo Penta*, 2018, accessed: 2019-03-12. [Online]. Available: https://www.volvopenta.com/marineleisure/en-en/news/2018/jun/volvo-penta-unveils-pioneering-self-docking-yacht-technology.html

[10] "Look, ma, no hands! auto-docking ferry successfully tested in norway," 2018, accessed: 2019-04-04. [Online]. Available: https://www.wartsila.com/twentyfour7/innovation/look-ma-no-hands-auto-docking-ferry-successfully-tested-in-norway

[11] Johansen, T. et al., "Autonomous uav surveillance of a ship's path with mpc for maritime situational awareness," *IEEE*, 2017, accessed: 2019-03-12. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7991361

[12] M. Rausand, *Risk Assessment: Theory, Methods, and Applications*. John Wiley Sons, 2011, ch. 1, pp. 1–28, accessed: 2019-03-12.

[13] S. Haugo, "Sensors for localization and mapping," 2018, accessed: 2019-03-12. [Online]. Available: https://lightbits.github.io/papers/sensors.pdf

[14] B. Mesnik, "Detection, recognition, and identification – thermal vs. optical ip camera," 2016, accessed: 2019-04-04. [Online]. Available: https://kintronics.com/detection-recognition-and-identification-using-thermal-imaging-vs-optical-ip-camera/

[15] A. Liszewski, "Elon musk was right: Cheap cameras could replace lidar on self-driving cars, researchers find," 2019, accessed: 2019-05-14. [Online]. Available: https://gizmodo.com/elon-musk-was-right-cheap-cameras-could-replace-lidar-1834266742

[16] "Real-time filtering of snow from lidar point clouds," 2018, accessed: 2019-03-12. [Online]. Available: http://wavelab.uwaterloo.ca/?weblizar_portfolio=real-time-filtering-of-snow-from-lidar-point-clouds

[17] T. Lee, "Why experts believe cheaper, better lidar is right around the corner," 2018, accessed: 2019-05-14. [Online]. Available: https://arstechnica.com/cars/2018/01/driving-around-without-a-driver-lidar-technology-explained/

[18] E. Olson, "Apriltag: A robust and flexible visual fiducial system," *IEEE*, 2011, accessed: 2019-04-04. [Online]. Available: https://april.eecs.umich.edu/media/pdfs/olson2011tags.pdf

[19] W. J. H. S. Zhang, C. and M. Yi, "Automatic real-time barcode localization in complex scenes," *IEEE*, pp. pp. 497–500, 2006, accessed: 2019-04-04.

[20] W. Xu and S. McCloskey, "2d barcode localization and motion deblurring using a flutter shutter camera," *IEEE*, pp. pp. 159–165, 2011, accessed: 2019-04-04.

[21] H. C. Chou, T. and Y. Kuo, "Qr code detection using convolutional neural networks," *IEEE*, 2015, accessed: 2019-04-04. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7158354

[22] "About handle," 2019, accessed: 2019-04-04. [Online]. Available: https://www.bostondynamics.com/handle

[23] S. Tørdal and G. Hovland, "Relative vessel motion tracking using sensor fusion, aruco markers, and mru sensors," *Modeling, Identification and Control, Vol. 38, No. 2*, pp. pp. 79–93, 2017, accessed: 2019-04-04. [Online]. Available: http://www.mic-journal.no/PDF/2017/MIC-2017-2-3.pdf

[24] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018, accessed: 2019-05-16. [Online]. Available: https://arxiv.org/pdf/1804.02767.pdf

[25] A. Karpathy, "Modeling one neuron," 2019, accessed: 2019-05-06. [Online]. Available: http://cs231n.github.io/neural-networks-1/

[26] ——, "Convolutional neural networks (cnns / convnets)," 2019, accessed: 2019-05-06. [Online]. Available: http://cs231n.github.io/convolutional-networks/

[27] A. Deshpande, "Modeling one neuron," 2016, accessed: 2019-05-06. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[28] E. Tangstad, "Visual detection of maritime vessels," 2017, accessed: 2019-05-06. [Online]. Available: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2452113

[29] J. Brownlee, "A gentle introduction to pooling layers for convolutional neural networks," 2019, accessed: 2019-05-12. [Online]. Available: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/

[30] M. Nielsen, "How the backpropagation algorithm works," 2015, accessed: 2019-05-06. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap2.html

[31] S. Suryansh, "Gradient descent: All you need to know," 2019, accessed: 2019-05-06. [Online]. Available: https://hackernoon.com/gradient-descent-aynk-7cbe95a778da

[32] "Gradient descent and stochastic gradient descent from scratch," 2017, accessed: 2019-05-12. [Online]. Available: https://gluon.mxnet.io/chapter06_optimization/gd-sgd-scratch.html

[33] "Precision and recall," 2019, accessed: 2019-05-12. [Online]. Available: https://en.wikipedia.org/wiki/Precision_and_recall

[34] M. Everingham and J. Winn, "The pascal visual object classes challenge 2012 (voc2012) development kit," 2012, accessed: 2019-05-12. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/devkit_doc.pdf

[35] "Detection and segmentation," 2017, accessed: 2019-05-12. [Online]. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

[36] R. e. a. Girshick, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014, accessed: 2019-06-08. [Online]. Available: https://arxiv.org/pdf/1311.2524.pdf

[37] D. Parthasarathy, “A brief history of cnns in image segmentation: From r-cnn to mask r-cnn,” 2017, accessed: 2019-05-28. [Online]. Available: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4

[38] J. e. a. Redmon, “You only look once: Unified, real-time object detection,” 2016, accessed: 2019-05-16. [Online]. Available: https://arxiv.org/pdf/1506.02640.pdf

[39] W. e. a. Liu, “Ssd: Single shot multibox detector,” 2016, accessed: 2019-05-16. [Online]. Available: https://arxiv.org/pdf/1512.02325.pdf

[40] J. Redmon and A. Farhadi, “Yolo9000: Better, faster, stronger,” 2016, accessed: 2019-05-16. [Online]. Available: https://arxiv.org/pdf/1612.08242.pdf

[41] A. Kathuria, “What’s new in yolo v3?” 2018, accessed: 2019-05-28. [Online]. Available: https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b

[42] “digital image,” 2019, accessed: 2020-01-02. [Online]. Available: http://www.digitizationguidelines.gov/term.php?term=digitalimage

[43] “Capturing the image ccd and cmos sensors,” 2019, accessed: 2020-01-02. [Online]. Available: https://cpn.canon-europe.com/content/education/infobank/capturing_the_image/ccd_and_cmos_sensors.do

[44] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*.

[45] “Camera calibration and 3d reconstruction,” 2019, accessed: 2019-12-14. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findcirclesgrid

[46] “What is camera calibration?” 2019, accessed: 2020-01-03. [Online]. Available: https://se.mathworks.com/help/vision/ug/camera-calibration.html

[47] J. Heikkila and O. Silven, “A four-step camera calibration procedure with implicit image correction,” *in Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. pp. 1106–1112, 1997, accessed: 2020-01-03. [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/devkit_doc.pdf

[48] P. e. a. Sturm, “Camera models and fundamental concepts used in geometric computer vision,” 2011, accessed: 2020-01-04. [Online]. Available: https://hal.inria.fr/file/index/docid/590269/filename/sturm-ftcgv-2011.pdf

[49] R. Holt and A. Netravali, “Camera calibration problem: Some new results,” *CVGIP: Image Underst. 54.3*, p. pp. 368–383, 1991, accessed: 2020-01-04. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/104996609190037P?via%3Dihub

[50] G. Schweighofer and A. Pinz., “Robust pose estimation from a planar target.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2006.

[51] “Bundle adjustment,” 2019, accessed: 2020-01-11. [Online]. Available: https://en.wikipedia.org/wiki/Bundle_adjustment

[52] F. Li and J. Niebles, “Stereo vision,” 2016, accessed: 2020-01-06. [Online]. Available: http://vision.stanford.edu/teaching/cs131_fall1617/lectures/lecture9_10_stereo_cs131_2016.pdf

[53] D. Betsy, “What is focal length and which lens should i use?” accessed: 2019-12-08. [Online]. Available: https://www.colesclassroom.com/focal-length-basics-every-photographer/

[54] “Configuring synchronized capture with multiple cameras,” 2017, accessed: 2019-12-08. [Online]. Available: https://www.flir.com/support-center/iis/machine-vision/application-note/configuring-synchronized-capture-with-multiple-cameras

[55] “Lidar-os-1-datasheet,” 2018, accessed: 2019-12-08. [Online]. Available: https://www.positics.fr/wp-content/uploads/2019/02/LIDAR-OS-1-Datasheet.pdf

[56] A. Dhall, “Real-time 3d pose estimation with a monocular camera using deep learning and object priors,” 2018, accessed: 2019-05-12. [Online]. Available: https://arxiv.org/pdf/1809.10548.pdf?fbclid=IwAR0gEosgUNUZKcRn-57IcyXdvND_xKZ01BYivf8eblkcNqrxlu87tzEBaiU

[57] “Spinnaker sdk camera driver,” 2019, accessed: 2019-12-29. [Online]. Available: https://github.com/neufieldrobotics/spinnaker_sdk_camera_driver

[58] A. Karpathy, “Transfer learning,” 2019, accessed: 2019-05-06. [Online]. Available: http://cs231n.github.io/transfer-learning/

[59] S. Nayak, “Training yolov3 : Deep learning based custom object detector,” 2019, accessed: 2019-06-17. [Online]. Available: https://www.learnopencv.com/training-YOLOv3-deep-learning-based-custom-object-detector/

[60] “Stereo camera calibrator app,” 2019, accessed: 2019-12-22. [Online]. Available: https://se.mathworks.com/help/vision/ug/stereo-camera-calibrator-app.html

[61] “Aruco marker detection,” 2019, accessed: 2019-12-25. [Online]. Available: https://docs.opencv.org/master/d9/d6a/group__aruco.html#ga84dd2e88f3e8c3255eb78e0f79571bd1

[62] “solvepnp,” 2019, accessed: 2020-01-08. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html?highlight=findcirclesgrid#bool%20solvePnP(InputArray%20objectPoints,%20InputArray%20imagePoints,%20InputArray%20cameraMatrix,%20InputArray%20distCoeffs,%20OutputArray%20rvec,%20OutputArray%20tvec,%20bool%20useExtrinsicGuess,%20int%20flags)

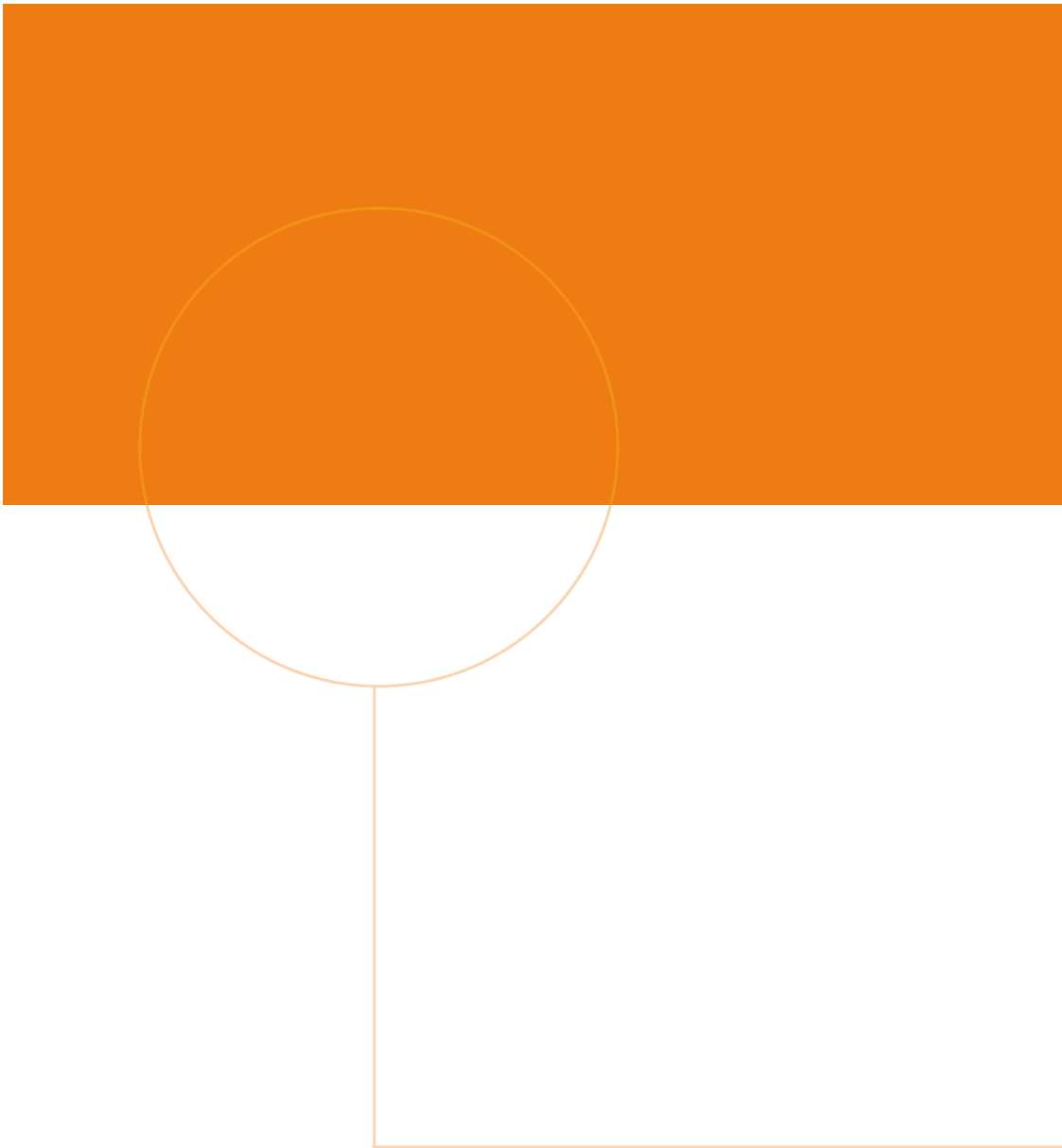
[63] “Jetson agx xavier thermal design guide,” 2018, accessed: 2020-01-11. [Online]. Available: https://static5.arrow.com/pdfs/2018/12/12/12/22/1/565659/nvda_/manual/jetson_agx_xavier_thermal_design_guide_v1.0.pdf

[64] “Rodrigues,” 2019, accessed: 2019-12-14. [Online]. Available: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#void%20Rodrigues(InputArray%20src,%20OutputArray%20dst,%20OutputArray%20jacobian)

[65] H. e. a. Xu, “Adversarial attacks and defenses in images, graphs and text: A review,” 2019.

[66] G. Welch and G. Bishop, “An introduction to the kalman filter.”

[67] A. e. a. Sans-Muntadas, “Path planning and guidance for underactuated vehicles with limited field-of-view,” 2019, accessed: 2019-06-08. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0029801818310333?via%3Dihub



NTNU

Norwegian University of
Science and Technology