NTNU
Norwegian University of
Science and Technology

Specialization project

Vision-Based Navigation for Ship Docking

Øystein Volden

Submission date: June 18, 2019
Supervisor: Thor I.Fossen, ITK
Co-supervisor: Marco Leonardi, ITK
Øivind Kåre Kjerstad, Kongsberg Maritime

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Project Description

Introduction

This is a specialization project done in collaboration with Kongsberg Maritime (KM). The work will be continued in the master thesis.

Main objective

The main objective of the overall project is to explore how stereo vision can used as a supplementary position system in autodocking operations using computer vision and deep learning techniques. The camera system should deliver accurate localization estimates nearby the dockside, run real-time and be robust with respect to outdoor conditions. For this project, the scope is limited to succeed part of the working system. This includes to successfully train and test a deep learning based object detection model and evaluate how well it detects and classifies different markers (used as reference objects) in various docking operations. Hopefully, the project will converge towards a supplementary navigation 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 deep learning techniques for computer vision applications.
- Prepare a laptop for deep learning based computing (includes parallel processing software such as CUDA).
- Record a dataset with markers in autodocking scenarios using a small, flexible USV.
- Prepare recorded dataset for deep learning tasks (includes calibration of a fisheye camera).
- Train and validate a deep learning based object detection model. Test the detector and evaluate its accuracy.
- Test and compare different marker configurations.
- Present some suggestion 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 potential to reduce costs and improve safety dramatically. However, there are several challenges to face before fully autonomous ships may enter the commercial market. One of them is autonomous docking.

The overall project is a study of how a low-cost stereo vision system can be used as a redundant position system in docking operations. That is, when GPS is not available or redundant position estimates is desirable, autonomous vehicles can obtain navigation information with cameras mounted on the vehicle. A vision-based navigation system consisting of an object detector followed by classical stereo vision techniques are proposed. For this report, the object detection model (e.g. the backbone of the perception pipeline) has been given extra attention. 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. To face the challenge of robust detections with respect to 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 is widely represented in the dataset. The key idea is to utilize data-driven methods in outdoor environments where classical computer vision techniques may fall short. To simplify the task of locating the camera relative to its surroundings, easily identifiable markers with assumed known global coordinates is used to obtain reference points.

Preface

This specialization project is carried out in the Department of Engineering Cybernetics, at NTNU in Trondheim, the spring of 2019. It is submitted as a requirement for the specialization project TTK4550.

I want to thank my supervisor, Professor Thor I.Fossen, for valuable guidance throughout every stage of this thesis. In the same regard, this work would not be possible without the help from my co-supervisor Marco Leonardi which have given great support and supervising, both on theoretical and practical aspects related to my thesis.
I would also like to thank Dr. Øivind Kåre Kjerstad for contributing with helpful information and guidance during the project. I am also very grateful for Maritime Robotics who letting me use one of their USVs for data collection related to the work.

At last, I would like to thank my family for their great support.

Contents

List of Tables

List of Figures

Abbreviations

AI	=	Artificial Intelligence
AP	=	Average Precision
AUC	=	Area Under Curve
COCO	=	Common Objects in Contexts
CNN	=	Convolutional Neural Network
CUDA	=	Compute Unified Device Architecture
DP	=	Dynamical Positioning
ECEF	=	Earth-Centered Earth-Fixed
EO	=	Electro Optical
FC	=	Fully Connected
FPN	=	Feature Pyramid Network
FOV	=	Field Of View
GD	=	Gradient Descent
GNSS	=	Global Navigation Satellite Systems
GPS	=	Global Positioning System
GPU	=	Graphical Processing Unit
IOU	=	Intersection Over Union
IR	=	Infrared
LIDAR	=	Light Detection and Ranging
POE	=	Power Over Ethernet
PR	=	Precision Recall
RADAR	=	RAdio Detection And Ranging
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
USV	=	Unmanned Surface Vehicle
YOLO	=	You Only Look Once

Chapter 1

Introduction

This is a specialization project done in collaboration with Kongsberg Maritime (earlier Rolls-Royce Marine). It is a case study on how stereo cameras can be used as a redundant position system for auto-docking operations with computer vision and deep learning methods. More specific, 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 exists on most ships today and deliver position measurements. In that sense, the camera system produce redundant measurements. However, the need for redundancy (i.e. duplication of critical measurements) increases as an autonomous ship need to be extremely reliable and perform well at all time. This is crucial in order to succeed the business of autonomous ships and maintain trust among investors, the crew, passengers and so on.

GNSS is used as the main position system on board most ships today. The technology is well-established and GNSS on board ships often holds a high standard. However, in docking operations without a human operator presented (i.e. auto-docking), 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 dramatically: 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 visual-based navigation system in the last meters of the docking and optimize the position of the ship based on both systems. The key idea is to have a low-cost backup position system specialized for auto-docking environments that can prevent the ship from unfortunate manoeuvres based on inaccurate or lack of GNSS measurements.

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 position system. The idea is to place

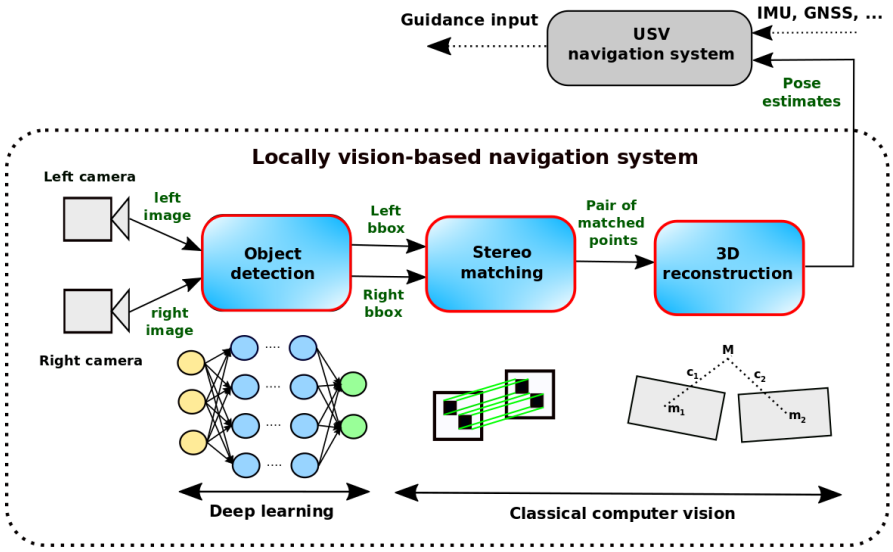


Figure 1.1: Proposed vision-based navigation system for the overall project.

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 pose estimates from the camera system. Further, it is proposed to place the cameras in a pair to achieve stereo vision view between the front of the USV and the quay. Figure 1.1 illustrates (on high level) how the vision-based navigation system is intended to work and how the pose estimates may be used by the USV together with other navigation measurements. For now, the intention is that the reader get some context of the "end-product". Further details around this system will be reviewed in Chapter 6. In this report however, the camera system will be limited to a single camera and aim to develop, test and verify that some modules of the whole camera system works as intended. Specifically, the object detection module in Figure 1.1 will be given extra attention.

On the hardware side, the focus will be on cameras and lenses suited for auto-docking operations. For instance, wide angle fisheye lenses is used to cover as much as possible 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). Anyway, there are reasons to believe that many of the ideas and principles from the proposed vision-based navigation system can be transfered 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 [17]. 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* takes 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 need 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 [2]. The demonstration was done on the finish 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. However, the fact is that although it is demonstrated in public, some challenges remains before a robust autonomous docking system can 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 set the ship in a dangerous position. Other limitations of the GPS includes multipath effects, interference, jamming and occasional high noise content [18]. For instance, the multipath phenomen is a well-known challenge in the Norwegian fjords where satellite signals tend to reflect via the mountain sides and confuse the GNSS receiver as it receives multiple signals from the same source (i.e. the satellite) [19]. Autonomous cars also suffers from related problems: Self-driving cars in the city can loose GNSS signals due to satellite signal blockage from the buildings. Obviously, indoor applications suffers 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 extensively research on how sensors like camera and lidar can benefit from their strengths, both standalone and in a sensor fusion system. The goal is to use such sensors in a local navigation system to prevent vehicles from dangerous situations where life may get lost.

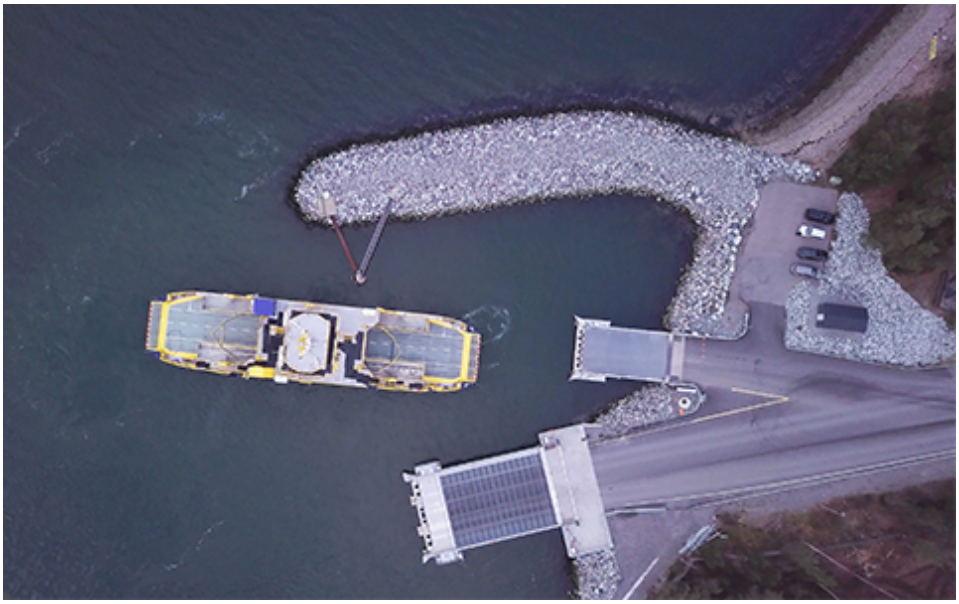


Figure 1.3: The ferry *Falco* during the world's first fully autonomous ferry transit December 2018. Image courtesy of Teknisk Ukeblad [2].

Although the auto-docking operation is not considered as a high speed application, the redundancy problem is still serious for ships in general. A system completely redundant to traditional position systems is therefore desirable and one can benefit from the related research on autonomous cars that tries to solve many the same problems. There are several relevant sensors that may be used and they are reviewed in detail in Chapter 2.

Another challenge is cyber security threats. Navigation and sensory systems are vulnerable to several cyber-physical attacks such as jamming, spoofing and bitstream manipulation. Lately, Russia is accused for jamming the GPS signals in the air-craft traffic in the north part of Norway and Finland. The signals from the satellites considered as weak and can easily be corrupted by sending signal noise from the ground and thereby jamming the GPS signals [20]. Although air-crafts have several navigation system 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 manoeuvres. This motivates the need for other systems than GNSS to address this issue. In addition, the sensor measurements should be encrypted to increase resilience of autonomous vehicles.

The economical aspect is also important, specially for older ships. To make a lot of new installations on older vessels may not be beneficial. For instance, it could be hard to configure the new sensors with older equipment and one can question if it is economical enough to make a profit of the investment. For a shipowner, it is therefore desirable to use the equipment already installed and reduce it to a software installation. New ships how-

ever should be built with sensors on an flexible infrastructure that enables and simplifies the possibility for auto-docking (if not already installed).
In addition, it is a huge 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 visual-based localization system is definitely an important part of this.

1.2 Contribution

This project explores different methods that are needed for estimating the position of an USV during docking operations. Specially, it has been paid a lot of attention to the object detection model which is considered as the backbone of the pose estimation system. A data-driven learning method is explored to achieve robust, fast and precise detections in various outdoor conditions. To achieve this, considerable amount of time was used to collect data and prepare them. This includes evaluating data relevant for the detection tasks, correct distortion from the fisheye images with a camera calibrator and label ground truth examples for the learning task.
In addition, a surprisingly complex "cocktail" of software packets are required to run deep learning models locally on laptops (with a powerful GPU) which means that several challenges may occur on the way before you can even do simple tests with the deep learning model. When the deep learning model is able to run, a more interesting part of the thesis starts. That is, to train and evaluate the deep learning model.
A lot of time has also been spent on practical work relevant for the master thesis. This includes to find suitable equipment for a camera rig which should be water-proofed for outdoor use and make sure the rig fit the test platform (e.g. the USV). It has also been paid attention to practical work related to the machine vision camera system, a pair of GiGE PoE cameras and fisheye lenses. Short summarized, this includes connecting the cameras with a laptop using a PoE switch (a combined power supply and data transmitter) and an available IP address and then record a camera stream using a SDK for FLIR blackfly cameras.
At last, the deep learning model YOLOv3 has been integrated into ROS (Robot Operating System) and this work will be continued in the master thesis. ROS is a flexible framework for writing robot software and supports many machine vision cameras (like blackfly cameras) and offers also bridging between openCV and ROS.
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 position systems in the maritime industry. This aim to reflect the motivation and theory behind the methods used in the project.

1.3 Structure

The project have now been introduced and several challenges that a fully autonomous ship must face with a special emphasize on auto-docking have been shown. Chapter 2 contains a brief introduction to autonomy, positioning principles and relevant sensors and markers for localization tasks. Chapter 3 introduces theory within the field of computer vision and deep learning related to the implementation. Chapter 4 reviews the choice of software and hardware for the project. Chapter 5 and 6 focus on the methods applied in the project and briefly introduce the whole pipeline which will be continued in the master thesis. Chapter 8 presents the results followed up by an discussion. At last Chapter 9 summarize the findings in the thesis and give some suggestions for future work.

Chapter 2

Related work

This Chapter presents literature references to related work within autonomous systems and sensors and markers for localization.

2.1 Autonomous Systems

We start this Chapter by mention a couple of autonomous systems and their history, in order to get some context within the field of autonomy. Systems that can change their behaviour in response to unanticipated events during operation are called "autonomous" [21]. The *Johns Hopkins Beast* is considered the worlds first autonomous system purely based on feedback control, a theory that was founded in the field of cybernetics. In the 1960's, 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 by its own. The robot did not use a computer and was purely based on transistors controlling analogue voltages, and it used sonar and photocell optics to navigate itself [21].
Since the 1960's, the development of autonomous systems have proceeded a long way. Today, autonomy within the car industry have received a lot of attention. Several companies like Tesla and Uber among many others are working hard on autonomous solution for the commercial market. However, the strict security requirements make it challenging to turn vision into reality and the scepticism among many people will not disappear as long as accidents related to autonomous vehicles still occurs.
The last years, autonomy has also been given more attention by the maritime industry, and companies like Kongsberg Maritime, Volvo Penta, Wärtsila, Rakuten and Maritime Robotics are all working on solutions for autonomous ships. Kongsberg Maritime is, in collaboration with DNV-GL and Yara, working on the first fully autonomous cargo ship *Yara Birkeland* as shown in Figure 2.1. It is estimated to be launched by the first quarter of 2020 and gradually move from manned operations to fully autonomous operations by 2022 [22].

Figure 2.1: Yara Birkeland is a result of a strategic collaboration between Kongsberg Maritime, DNV-GL and Yara where the goal is to make the first fully autonomous cargo ship. Image Courtesy of Teknisk Ukeblad [3].

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 2.2.

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 their 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 manoeuvres. How the docking operation is performed in detail will differ 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)

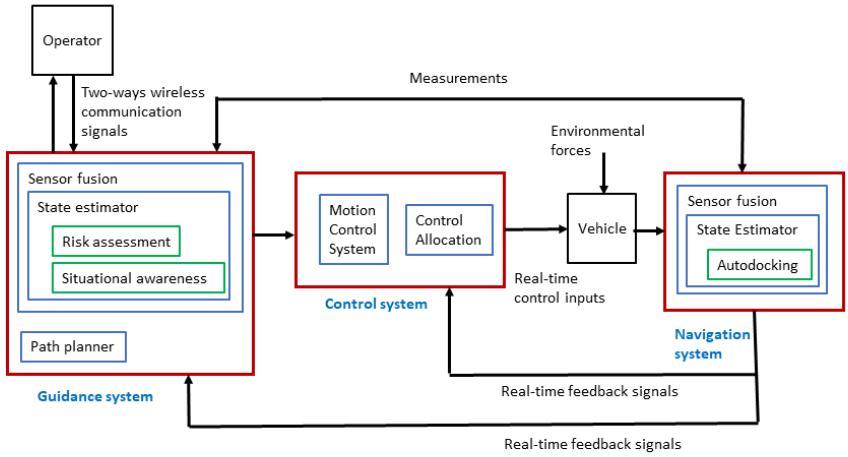


Figure 2.2: A classical guidance, navigation and control system with auto-docking proposed as a local navigation system nearby the quay. Other autonomous applications, like situational awareness and risk assessment, can be used for human-decision support as shown the guidance block. Unlike situational awareness and risk assessment, auto-docking is intended to work more independently of a human operator.

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 makes it challenging to develop. As already mentioned, a lot of companies are working on solutions for automatic docking system 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 [23]. Wärtsila develops an auto-docking system which, in contrast to Volvo Penta's system, relies heavily on the ships dynamical positioning (DP) system based on GNSS and inertial measurements units (IMU) to avoid drift in the position estimates [24]. 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 project.

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 [25]. 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 occurs. 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 [26]. To be able to perform risk analysis, one need 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 on 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 each single risk for decision-making. In 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 may 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 detects 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. 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.

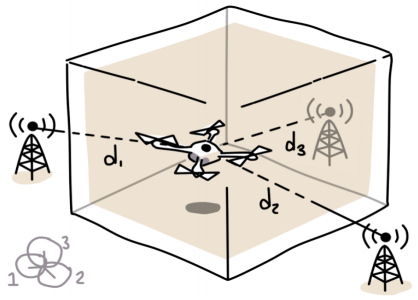


Figure 2.3: Trilateration principle in three dimensions. Image courtesy of [4]

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 position system should be able to estimate location of some object based on sensor measurements. In this context, the principles of trilateration and triangulation are very central and is 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 3 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 [4]. 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 (e.g. the tip of the boat).

2.3 Sensors

To make a position system, one need 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 intervention of a human operator. Therefore, the ultimate navigation system may use sensor fusion to benefit from strengths of each sensor and provide robustly and reliable measurements compared to what one sensor can provide alone. For this project, 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 measures 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 priori 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 extensively research on visual-based localization systems lately, specially 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 position systems are considered more robust and reliable compared to other sensor-based localization systems [18]. 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, motion blur, textures and presence of heavy rain, snow and fog [18]. 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 position 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 involves several steps like acquisition of the camera images, extraction of detected features, matching features between frames and calculate position via pixel displacement between frames.

IR Camera

While optical cameras provide images based on light reflecting off the object, infrared cameras measures heat energy emitted by the object [27]. They perceive light of wavelengths, both inside and outside the visible spectrum [4]. That means they cannot provide real-world colors the same way optical cameras does. Instead, they can be used to identify colors outside the visible spectrum. Since infrared cameras measures 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 stand out from the scene and detect it 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 an optical cameras, infrared cameras are passive as it does 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 are cooled down, it may be impossible to detect the same small object far away [27]. 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 specially popular in the development of fully autonomous cars nowadays (except Tesla and particularly Elon Musk [28]) and is widely used for object detection, obstacle avoidance and 3D mapping. It is a laser-based technology that use Time-Of-Flight (TOF) to measure 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 [4]. And they can target a wide range of materials, but also remove some of interest. For instance, one can apply filtering methods to denoise even thick snow [29]. These capabilities make the lidar very robust to different outdoor conditions.

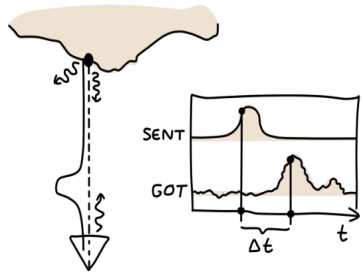


Figure 2.4: Time-of-flight principle. Image courtesy of [4].

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 is critical for the auto-docking operation.

One drawback of lidar is the price, specially for high-resolution versions. However, the increasing use of it, specially in the automotive industry, push the price down every year and it is expected to be less expensive in near future [30]. Morover, 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 have 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 too 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 is 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, leaf and birds. Furthermore, radio waves have the ability to more easily penetrate materials and this makes radar a better choice (than a lidar) in presence of fog/smoke [4].

Until now, radar sounds like the perfect sensor for outdoor localization. As with the

other sensors, it also have weaknesses. Radio waves normally reflects 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 a 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 [4].

Comparison

As seen, all sensors have benefits and drawbacks. The objective of the project is to make a simple, low-cost position system that deliver accurate estimates and is robust with respect to outdoor applications such as autodocking. With this objective in mind, electro optical cameras is a natural choice due to its price, flexibility, accuracy and amount of information stored in every image. However, optical cameras is 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 are proposed to detect such features (inside the markers) easily.

Fiducial markers

A lot of markers exists 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 has a small information payload (perhaps 12 bits), but is designed to be detected even at very low resolutions which allows 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 [31]. 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, ARToolkit,

ARTag and April Tag. Since this project focuses on outdoor localization, markers such as ArUco markers and April tags are specially 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 solve 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 operate at a close range with high localization precision.

Natural landmarks

In addition, it might be possible to use naturally-occuring 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 operation opportunities 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-occuring landmarks, and they also stores a great load of memory compared to simple fiducial markers [31]. 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 may 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 hard coded 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 needs 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 position system. Other environmental factors like fog may reduce the camera sight and make the markers useless as well.

Figure 2.5: Boston Dynamics newest 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 [5].

CNN for marker detection

Detecting and locating fiducial markers in complex backgrounds is a challenging step. A lot of research have been done in this field. Zhang et al. [32] propose a method to detect non-uniformely illuminated and perspective distorted 1D barcode based on textual and shape features, while Xu et al. [33] developed an approach for detecting blur 2D barcodes based on coded exposure algorithms. The mentioned methods show high detection rates 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 [34]. 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 differ 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 [35], 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 [35].

Chapter 3

Theory

This Chapter aims to introduce the theory related to the techniques used in the implementation in Chapter 6. This includes 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.

3.1 Convolutional Neural Networks

On the highest level, a 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 context of CNNs with deep layer structure. It has been proved that the interpretability of a CNN tend to decrease as the number of hidden layers increase (which again means the CNN tend 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 a some specific decision.

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

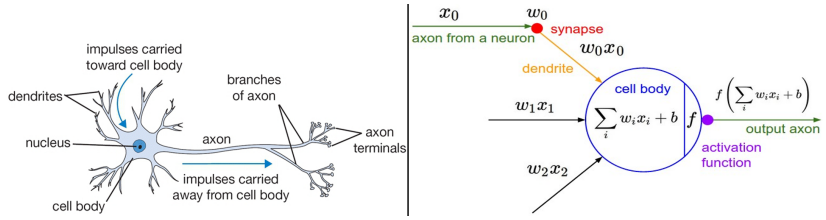


Figure 3.2: The biological neuron to the left and its mathematical model to the right. Image courtesy of Stanford University [6].

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 how *neurons* works (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 [6].
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 gives 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 sum 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 aim 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 the bias b_1^2. The single result, $\boldsymbol{z}_1^2 = w_{11}^1 x + b^1$, is then processed through an activation

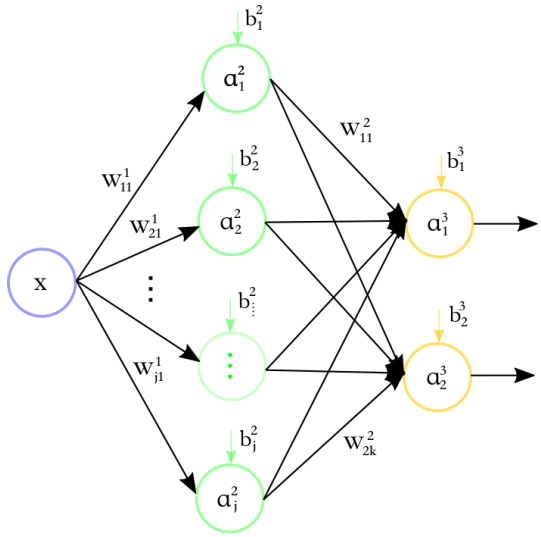


Figure 3.3: Single layer, fully connected neural network.

Table 3.1: Model parameters of a neural network.

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

$a_1^2 = \sigma(\boldsymbol{z}_1^2)$ which becomes an input signal for neurons in the next layer. Note that the next layer receives input from several neurons and hence, it need 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 defined as

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

And 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 the these single numbers (sorted into an array) represents a feature map.

The network treat 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 [8]. 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. In other words, many small filters are needed to build representations of complex classes (even for a cartoon mouse).

With some intuition on how the convolutional layer works, we can describe it more generally. What characterizes any respective convolutional layer are hyperparameters.

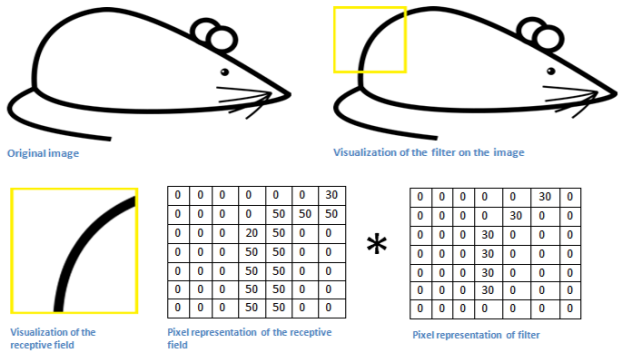


Figure 3.4: Convolutional filter visualized. A specific filter slides over the original image and activates when it recognize the low-level feature it is looking for. Image courtesy by [7].

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 [36].

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 the last years. ReLU is specially designed for scale invariance and efficient computations which makes it an suitable candidate for activations of a convolutional layers. It is an 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 struggles 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 through 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 face 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 [37].

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 exists and shortly said they determine how the output feature map (from each convolutional layer) is filtered. The two common pooling operations is average pooling and max pooling. Average pooling calculate the value for each patch of the feature map, while max pooling calculate the maximum value for each patch of the feature map (see Figure 3.5). The pooling layer require two hyperparameters: Stride **S** and Spatial dimension **F**. Note that max pooling is mostly used, and specially with 2×2 pixel size. A bigger pixel size will tend to loose a lot of information.

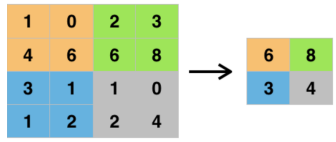


Figure 3.5: Max pooling operation.

Fully connected layer

In a Fully Connected (FC) layer, each neuron has full connectivity to all activations in its pervious 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 defined 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 is 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-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).

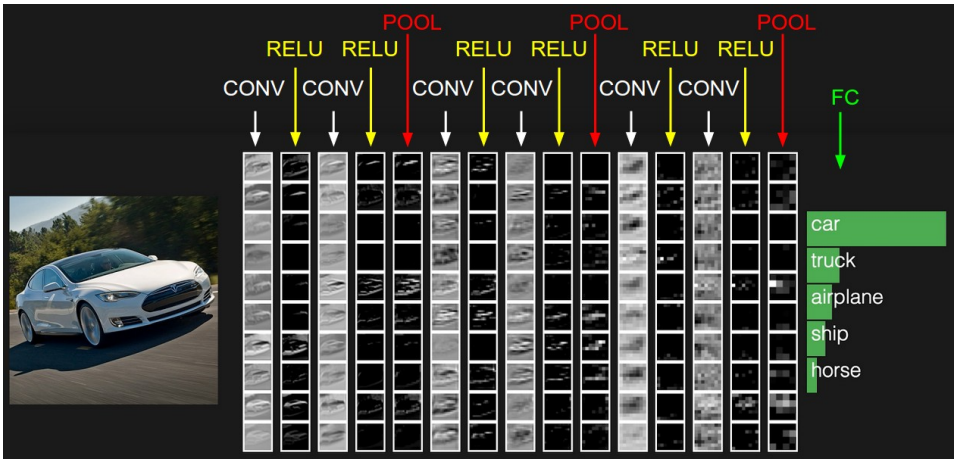


Figure 3.6: Activations of an example CNN architecture for car recognition. Image courtesy of Stanford university [8].

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 is adjusted to minimize this objective function. That 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 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 defined 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 changes 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:

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

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 $\frac{\partial C}{\partial \boldsymbol{w^l}}$ and $\frac{\partial C}{\partial \boldsymbol{b^l}}$. 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 backpropagates 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 reach 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 [38].

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 computations. 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 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

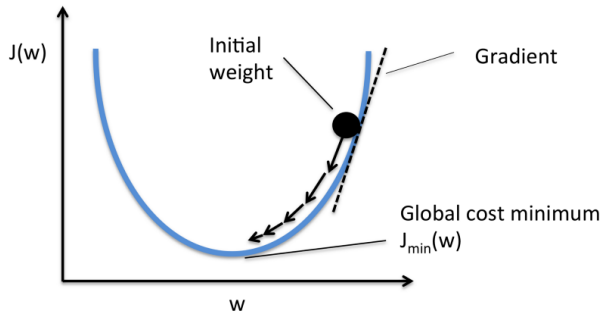


Figure 3.7: Gradient Descent procedure illustrated in 2 dimensions. For simplicity, the cost function, here denoted as $J(\boldsymbol{w})$, is only optimized with respect to the weights $\boldsymbol{w}$. Image courtesy of hackernoon [9].

function C given some initial starting point. Also notice how the steps tend to decrease as the weights gets closer to the optimal minimum. Further, the weights and biases are only updated after each batch of training images are evaluated. The batch size determines the size of the subset of training images used for one iteration and also, how many images is 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 are 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 [39]. 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 do 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 [40]. 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.

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

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.

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 [41], 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 5. 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.

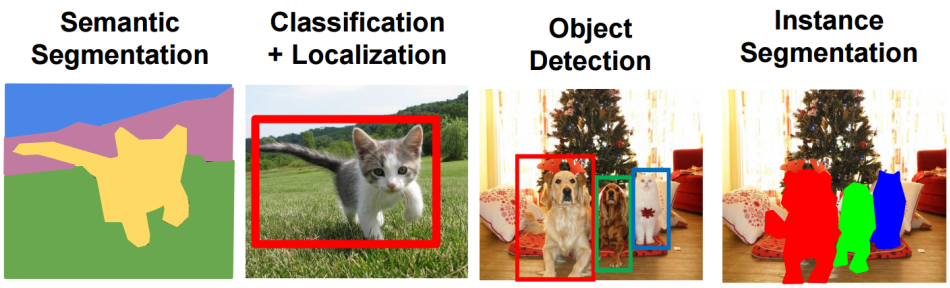


Figure 3.8: Object detection compared to closely related concepts. Image courtesy of [10].

3.2 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.

Object detection involves detecting and classifying multiple objects and where they are in the image marked by rectangle shaped bounding boxes. In this context, there does also consist closely related concepts like classification and segmentation, but how do they differ? First of all, classification simply focus on whether a single object exist in an image or not, but its pixel coordinates (e.g. localization) is 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 aim to not only distinguish between classes, but also distinguish between objects at pixel-level. In fact, instance segementation only differ slightly from object detection. That is, object detection use bounding boxes to assign classes, while instance segmentation assign each class at pixel-level (which is more computationally expensive). Mask R-CNN is a known model for instance segmentation.

3.2.1 State-of-the-art models

Here, several state-of-the-art models is 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 CNN for object detection progress.

Faster R-CNN

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

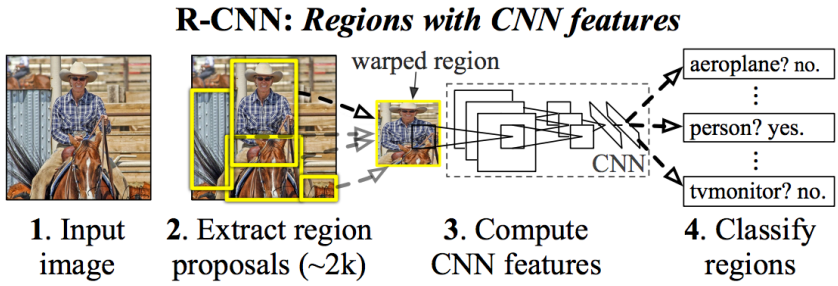


Figure 3.9: R-CNN framework. R-CNN takes input image, creates a set of regions, computes features for each proposals using a deep CNN and classifies each region using class-specific linear SVMs. Image courtesy of [11].

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 is 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 size (also called sliding window) and for each tries to group together pixels in same region by texture, color or intensity to identify objects. Next, the region proposals is transformed to standard square size and pass it through the CNN as shown in the slide. Overall, the convNet works as an feature extractor and classifier, e.g. it transform 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 [11]. 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 [42]. 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 [42]. 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. The first step, selective search was applied to generate a set of region proposals which is fairly slow. A team of researchers at Microsoft realised 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 [42].

Mask R-CNN

Briefly said, Mask R-CNN extend faster R-CNN for pixel level segmentation. The basic idea of Mask R-CNN is to go one step further and locate exact pixels of each object instead of just bounding boxes. This is known as instance segmentation. It adding 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 lead to some problems. 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 is avoided by using bilinear interpolation [42]. With this improvement, Mask R-CNN can generate more precise segmentations.

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 from Figure 3.10, YOLO divide 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

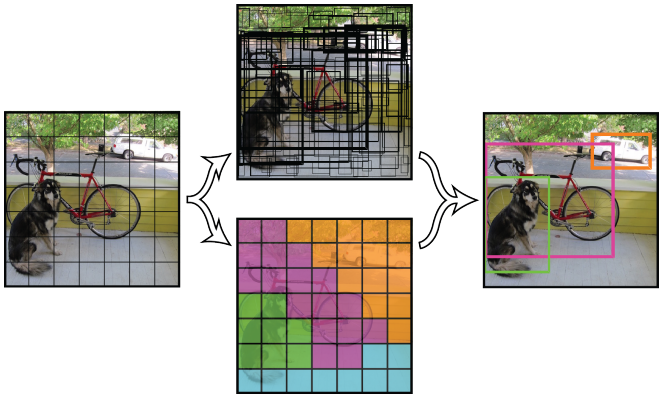
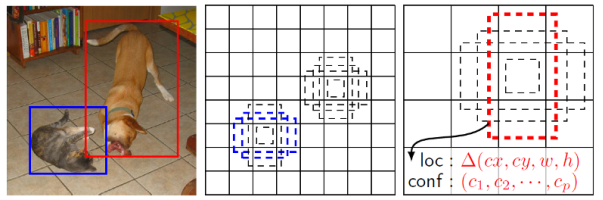


Figure 3.10: YOLO framework. Given an input divided into $S \times S$ grids, YOLO produce bounding box predictions (upper image) and a class probability map (lower image). These predictions are merged to final detections as seen in the right image. Image courtesy of [12].

a grid cell, that grid is responsible for detecting that object [12]. Each bounding box contain a confidence scores on how certain it is that an object exist or not within the bounding box. Notice that many of the bounding boxes (in the upper image in Figure 3.10) will have very 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 about how 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 [13]. 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 which allow predictions of detections at multiple scales. Another feature is default boxes and aspects 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 [13].



(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 [13].

Comparison

It is somehow hard to have a fair comparison among the different object detectors. A bunch of different papers use 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 is 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 use 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.

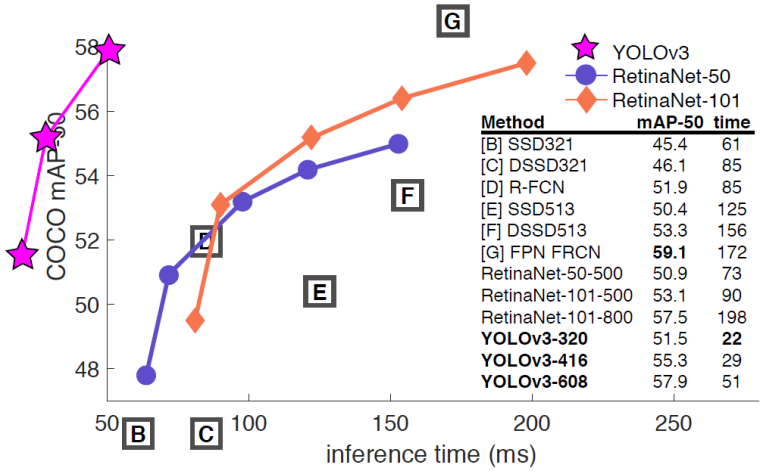


Figure 3.12: Inference time and corresponding mAP for different networks. The networks is tested at COCO dataset (80 classes) with 0.5 IOU threshold. Image courtesy of [14].

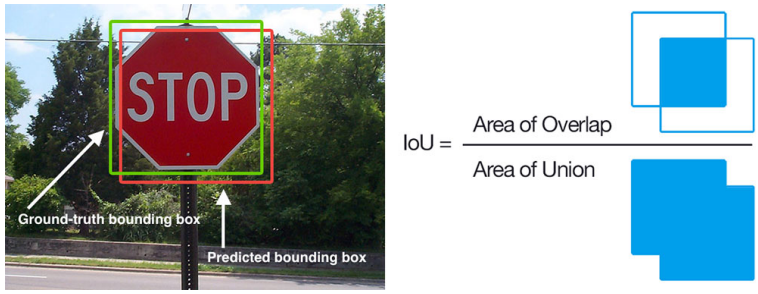


Figure 3.13: Intersection over Union (IoU) metric visualized.

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 specially characterized by its low inference time enabling for detections in real-time. This section aim 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 ratios. So instead of directly predicting a bounding box, YOLOv3 predict offsets from a pre-defined set of boxes with a particular height-width ratios. These predefined boxes are referred to as anchor boxes. In [43], a k-mean clustering algorithm is runned to obtain a number of anchors

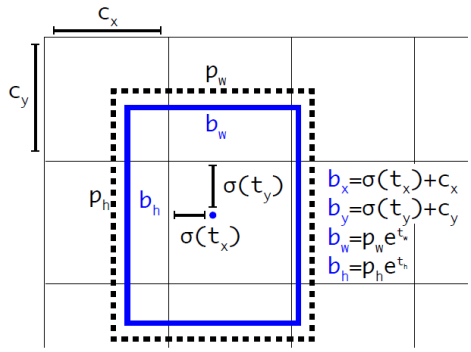


Figure 3.14: Bounding box prediction. The width and height of the blue box is predicted using offsets from the prior known anchor boxes. In the figure, only the closest anchor box with width p_w and height p_h is shown. The other anchor boxes is illustrated in Figure 3.15. The center coordinates of the box is predicted using a sigmoid function. Image courtesy of [14].

boxes which gives a good tradeoff for recall vs. complexity of the model. In YOLOv3 9 anchor boxes is 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 [14].

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 [14].

Class predictions

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

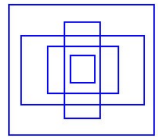


Figure 3.15: The 5 different anchor boxes used in YOLOv2 providing different height-width ratios. They are applied at different scales. In comparison, YOLOv3 provides 9 anchor boxes, 3 for each scale.

So how do YOLO figure out if a bounding box contains a specific type of object? Recall that YOLO divide 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 exist, 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 [12].

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 address this problem by using a logistic classifier instead which is able to perform multilabel classification for detected objects [15].

Predictions across scale

In order to detect small objects in the image well, YOLOv3 predicts boxes at 3 different scales. YOLOv3 have 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 is extracted from those scales.

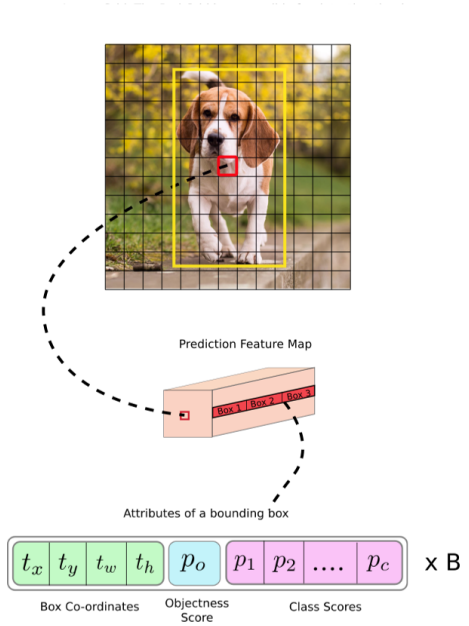


Figure 3.16: The content of a feature map for a single grid cell. B refers to the number of bounding boxes a grid cell can predict. Image courtesy of [15].

Features from those scales is extracted similar to a feature pyramid network (FPN). From our base feature extractor, we add several convolutional layers where the last represents a 3-d tensor with bounding box prediction, objectness and class predictions (see Figure 3.16). Next we take the feature map from 2 layers previous 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 provide 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 final scale [14].

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 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 stacked onto it, giving

	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 [14].

us a 106 layer architecture for YOLOv3. As there exists 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

Hardware and Software Choices

This Chapter represents the software and hardware components chosen for the project.

4.1 Hardware

This section introduces the hardware used in the project. This includes a camera, a computer, markers and a small USV.

Camera

The camera used in this project is GoPro Hero Session 5. It is small and flexible, and can be remoted wirelessly from a smartphone. Some of the motivation for using a goPro camera in this project was that it can easily be mounted anywhere on a small USV and controlled wirelessly from the quay. This makes it easy to record docking operations of the USV. In addition, it provides a fisheye lens which is useful for the application since a wide field of view allows the camera to see the markers as often as possible. The specific camera settings used for the project is reviewed in Chapter 5.

Computer

The main computer used for this project was a 17.4 inch Alienware laptop. It has a intel core i7 (7th gen) CPU, 8 GB RAM and a Nvidia GTX 1060 GPU. It is runned with Ubuntu 16.04 LTS. For future work, a more compact PC (i.e. Jetson Xavier or similar) may be used for computations on-board a small USV.

Marker configuration

After reviewing different types of markers in Chapter 2, the final choice of marker for testing in this thesis was ArUco markers. These fiducial markers are easy to recognize and

contain a small information payload which allows for fast, accurate and robust detections.

The main configuration is three markers fixed to a piece of wood with 75 cm between each one as shown in Figure 5.2. For comparison, different colors have been tested to see if some colors was easier to detect rather than others. Also, two different sizes (A3 paper size and A4 paper size) of the markers was made to check how much the size of the marker affect the accuracy of the detections from different ranges and views. These marker configurations will be reviewed in more detail in Chapter 5.

Otter USV

The Otter USV made by Maritime Robotics (see Figure 1.2) is the physical platform for this project. It is small, flexible and easy to navigate around the quay. The camera was mounted on the front of the USV.

4.2 Software

This section provides a wide range of software environments used in the project. Some of them are related to deep learning frameworks while other are used for more basic computer vision tasks like image acquisition and processing and camera calibration.

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.

Matlab

Matlab is one of the most used mathematical softwares among engineers around the world and provides toolboxes for a wide range of applications. One relevant toolbox used for this thesis is the camera calibrator.

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) have 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.

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 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.

FlyCapture

Flycapture is a Software Development Kit (SDK) for FLIR blackfly cameras. For this project, it was used to control the camera (i.e. record images) from a laptop. This SDK will most likely be investigated more in the master thesis.

ROS

The 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 the communication and data transfer across multiple systems. Different submodules communicates via messages, they receive data and output processed information. ROS also offers bridging between openCV and ROS. Further, blackfly cameras is widely supported by ROS. For this project, YOLOv3 have been integrated into the ROS environment to ensure that bounding box information can be used by other modules in future work. However, this work is not reflected in the results in this thesis, but may play an important role for future work in the master thesis.

Chapter 5

Data Acquisition

In order to explain the methods explored in the project, it is natural to start with the datasets as these lays the basis for data-driven learning methods in the computer vision field. This Chapter focus on the various data sets as well as the construction of a dataset designed for auto-docking operations.

5.1 ImageNet

As it is very time consuming to train a CNN from scratch, it is common 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 let 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 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.

5.2 Collecting Data

On April 5th, 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 harbour and slowly approaching the dockside from different angles. A flexible

Table 5.1: Camera specifications for the experiment.

Camera settings			
Camera	resolution	camera mode	fps
GoPro Hero session 5	3840 x 2160	Wide FOV	30

Table 5.2: Split between training, validation and test images.

Training, validation and test data			
Dataset	Training	Validation	Test
Custom	770	96	96

GoPro camera was mounted at the front of the USV and records where taken with settings as shown in Table 5.1 using a GoPro remote app.

Obviously, only relevant examples (i.e. images showing at least one marker) from the records was included in the custom dataset. Sampling of the relevant videos resulted in a custom dataset consisting of 962 images. Sampling methods effectively removes redundancy in the dataset as two frames in a row (i.e. a video sequence) shares almost the same patterns. This allows for effectively covering more variations in the dataset without using weeks for labeling. In addition, YOLOv3 performs on-line data augmentation which improves existing training data. Distortion from the wide FOV GoPro camera was also removed during the process using a camera calibrator. This is reviewed in more detail in Chapter 6.

5.2.1 Train, validation and test data

The custom dataset was randomly shuffled and then, 80 % of it was assigned to training, and 10 % was assigned to validation and 10 % was assigned to test data. Figure 5.3 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 sufficiently amount of data is trained on while some is left for validating the training and pick the most optimal weights for testing. At last, some test data verifies how accurate the model performs on unseen data.

The dataset in considered quite small, which sets a limit on how general the model can be. However, one may not need a very big custom dataset for detecting simple ArUco markers. Recall that YOLOv3 performs data augmentation online to produce synthetic data from the custom dataset and thereby improve training. The most important improvement may therefore be to increase the variations of how the markers looks like from different views and different environmental conditions.

5.2.2 Classes and marker configurations

Classes

The custom dataset contains in total four classes, where three of them is different ArUco markers and one is a natural occurring landmark. Figure 5.1 shows all the different markers for each class. As seen, the ArUco markers of a class shares many patterns, but vary in size and color. Variation within each class (except class 3) is included to see if some

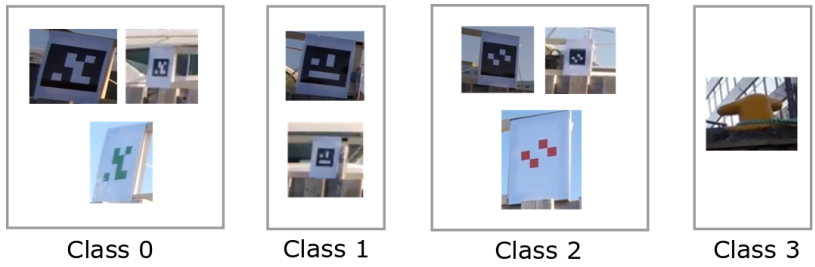


Figure 5.1: All the different markers within each class in the custom dataset. The ArUco markers (class 0-2) within the same class shares many patterns, but vary in size and color. Class 3 contains only one single natural occuring landmark, a yellow bollard.

Table 5.3: Number of ground truth labels for each class in the part of the custom dataset used for training/validation. Each class name is simply chosen as numbers from 0 to 3 corresponding to its class number.

Classes in custom dataset		
Class name	Marker type	ground truth labels
0	ArUco marker	945
1	ArUco marker	948
2	ArUco marker	955
3	Natural landmark	329

specific color or physical size of the ArUco markers is easier to detect compared to others. Class 3 contains only one single natural occurring landmark and is included to compare against ArUco markers in different contexts. Further, Table 5.3 shows that there is a balance between the first three classes (i.e. how often they occur in the dataset), while the last class, in comparison, have less ground truth labels in the dataset. This may not be a big problem as the last class only shows up as one type of object (as seen in Figure 5.2).

The reader may wonder why not ArUco markers with different patterns stays within the same class (as they all could have been defined as a class "markers"). For this project, the motivation is to verify if the detection model is able to distinguish between the markers. And for future work, it will hopefully simplify the pipeline of the whole working system. Normally, a real-time pose estimation system with ArUco markers need some sort of tracking capabilities 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 one marker configuration. By this assumption, one can simplify the tracking problem and assume that the class number corresponds to the object id. One can also apply simple filtering methods on top in case the machine learning algorithms makes wrong predictions. This will be explored further in future work.

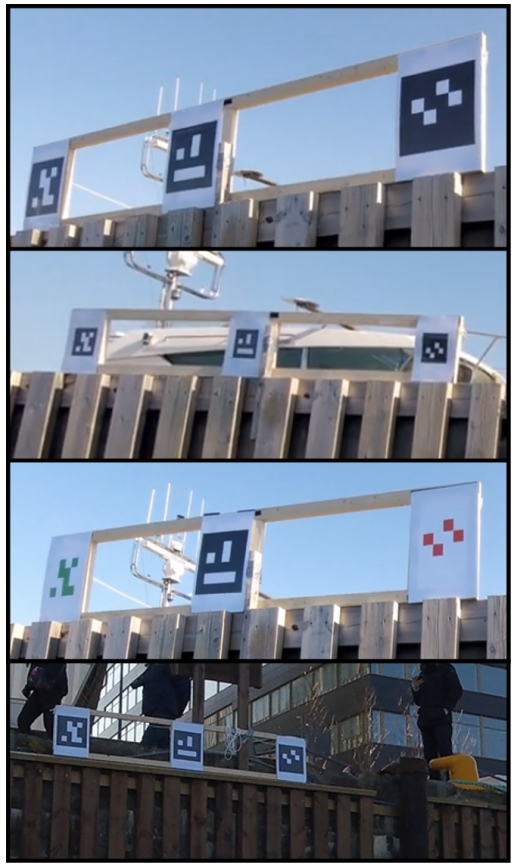


Figure 5.2: The four different marker configurations in the custom dataset. Note that lower subimage is an extension of the marker configuration in the upper image. That is, a yellow bollard (natural landmark) "extends" the row of markers making a total of four markers.

Marker configurations

While the classes tries to sort different markers by their patterns, the marker configurations shows which combination of markers (three or four markers in a row) that are used for each time. That is, only one marker configuration is used simultaneously when the USV approach the dockside. Figure 5.2 and Table 5.4 summarizes how the different marker configurations looks like and how often they occur in the custom dataset. One may observe that there are very few examples of A4 markers. Unfortunately, this is due to limited recorded data relevant for the thesis. However, the patterns learned from the other marker configurations (in the same class) may compensate for the low presence of A4 size paper configuration. The goal is to make one detector (instead of several more fine-tuned) that is able to detect and classify all the markers across the different marker configurations.

Table 5.4: The different marker configurations in the custom dataset provided with their physical size and number of occurrences (e.g. images). Note that the physical size of the natural landmark is left unknown.

Marker configuration	images	physical size (cm $\times$ cm)
A3 black/white ArUco marker	279	28×28
A4 black/white ArUco marker	13	14×14
A3 colored ArUco marker	314	28×28
Natural landmark + A3 black/white ArUco marker	353	unknown

5.2.3 Additional test videos

In addition to the custom dataset, it was decided to manually pick out some test videos showing different parts of the docking operation with different marker configurations. The main motivation is to see how the trained object detector performs in real-time in certain situations rather than on randomly picked test images. These scenarios will be reviewed in Chapter 7. To ensure that these test videos is independent of the custom dataset, they were manually handpicked before the custom dataset was made. In other words, the rest of the relevant material is leaved to the custom dataset.

Chapter 6

Implementation

This Chapter focus on the backbone of the pipeline, namely the object detection model. How the datasets are prepared and how the model can be trained, validated and tested within the framework will be reviewed. In addition, an overview of the whole perception pipeline is included in order to get some context of how the project work contributes towards a robust, real-time pose estimation system.

6.1 Pipeline Overview

This section aim to briefly introduce how the whole perception pipeline works. The final working system employ 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 classical computer vision techniques when the scope and complexity of the task is reduced. These design choices where chosen to make the pipeline more intuitive and easier to debug compared to an end-end-end deep learning pipeline.

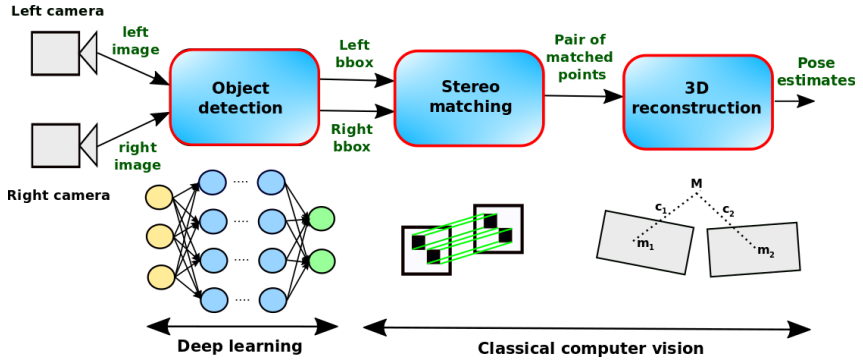


Figure 6.1: The locally visual-based navigation system revisited from Chapter 1.

It consists of three main modules as shown in Figure 6.1, where the object detection is deep learning based and the stereo matching/3D reconstruction modules relies within 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.

This is how it works briefly said: The object detection model receives a sequence of frames from the left and the right camera. 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 is feeded into a stereo matcher which finds corresponding points in the two images. Note that the search space is reduced dramatically which may favor classical computer vision methods. Now, these points may be corners given by the pattern of each fiducial marker. Also note that if the cameras are aligned to be coplanar, image rectification is not needed and the search of corresponding points is reduced to one dimension. Once the pair of points is matched, 3D reconstruction (for each pair of matched points) can be obtained directly using stereo triangulation to calculate a disparity (e.g. depth) map. The down-right corner of Figure 6.1 shows how a pair of corresponding points together with the position of each camera is used to construct and intersect lines to obtain 3-D coordinates of the point. From the disparity map, the relative 3D pose between one of the cameras and points in the markers can then be obtained from each frame in the sequence.

To speed up the pipeline, it is proposed to sample every second frame from each camera. That means, for each frame, switch between images from the left and the right camera and feed the (resulting) consecutive sequence of frames into the object detection model. As the docking operation in general is a slow-varying process, one can assume that the left and right image (e.g. two consecutive frames) forms a image pair that approximately represents the same time instance. Hopefully, this assumption will not affect the overall 3D point estimation significantly. Another, more process heavy approach would be to run two instances of YOLOv3 simultaniously (e.g. in parallel). Most likely will both approaches be tried in future work.

The pipeline is partly inspired by a thesis performing 3D pose estimation with an older version of YOLO (v2) and a monocular camera [44]. Instead of using fiducial markers, natural occuring cones on the track is used as reference objects. In comparison to 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 [44] with our proposed pipeline in this section to understand why. In addition, we propose to use classical computer vision techniques for stereo matching (referred to as keypoint regressor in [44]), 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 descriptors are sufficient for stereo matching when the problem is reduced (e.g. only detect points in bounding boxes and not in the whole image). The final working system in this thesis was used in several formula student competitions in 2018 and proved

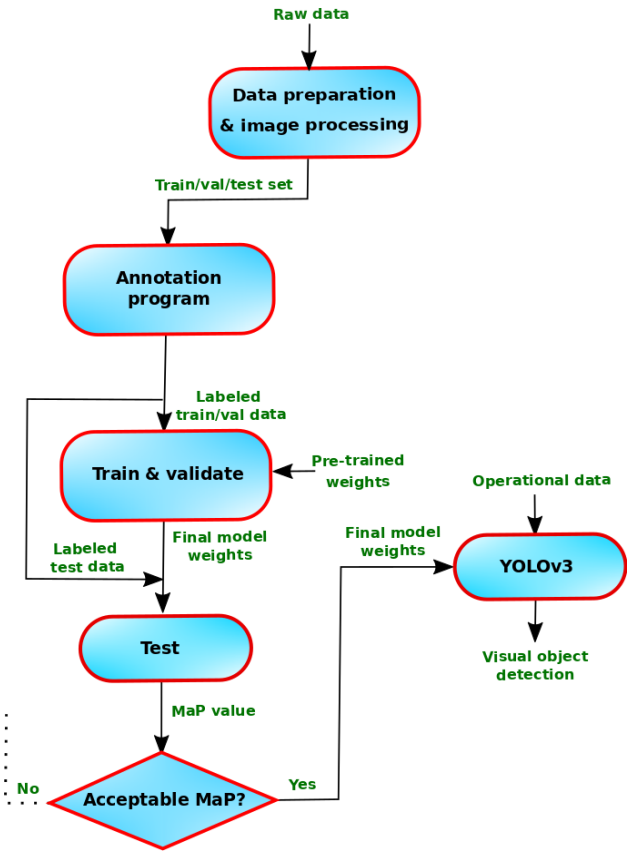


Figure 6.2: Object detection pipeline.

that the computer vision pipeline not only works in theory and for simple tests, but also on the track where accurate and robust estimates in real-time is required.

6.2 Object Detection Pipeline

Before diving into each sub-part of the process, lets briefly get an overview of the learning based pipeline. Figure 6.2 summarize 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 in docking operations. Step 2 involves transforming the prepared data to a format that the deep learning model understand. 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, stop training, validate the weights and conclude your final weights for testing. In Step 4, the fine-tuned model is tested on new unseen data (i.e. test data). Statistical metrics like mAP, IOU and Recall is 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 was not presented in the dataset the model was trained for.

6.2.1 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 is relevant for the learning task. But before any such examples can be learned by the model, the data should be prepared and transformed to 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 5 describes in detail how the custom dataset is gathered and which data that is included in it.

In addition, it is often prefered 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 is applied. Camera calibration is a widely used technique to remove distortion in the image and deserves some attention given in the next section.

Camera calibration

A consequence of using fisheye 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 the mapping between 3D coordinates and the image plane is correct.

In total, 31 fisheye images of a 8x8 checkerboard was taken from different views and orientations using a GoPro camera (same as in the custom dataset). Figure 6.3 shows one shot of the checkerboard pattern taken from a certain view. By using the single camera calibrator toolbox in Matlab [16], 31 images were added and 22 of them were approved by the calibrator app. Further, the prior length of the squares (22 mm) was given (necessary for computing any extrinsic parameters) in Matlab. A fisheye camera model was chosen in order to get correct calibration.
To evaluate the camera calibration, the reprojection error is used as a measure. The original mean reprojection error (of the 22 images) was 0.62 in pixels. To improve the calibration, outliers (images with the highest reprojection error) was removed and a re-calibration was done. In total, 4 images where removed and the 18 remaining images gave 0.53 mean reprojection error in pixels as shown in Figure 6.4. With 4K resolution images provided by the GoPro camera, this is considered as quite good results. Therefore, the model with the given reprojection error was saved and applied on the custom dataset to correct distortion. Figure 6.5 verifies a successfully calibration process.

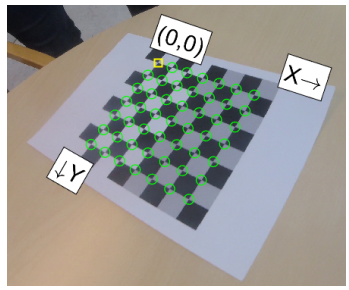


Figure 6.3: The camera calibrator app in Matlab [16] estimates camera intrinsics, extrinsics, and lens distortion parameters. Here, the app provide corner detections of the checkerboard pattern for one of the 22 approved images during the calibration process.

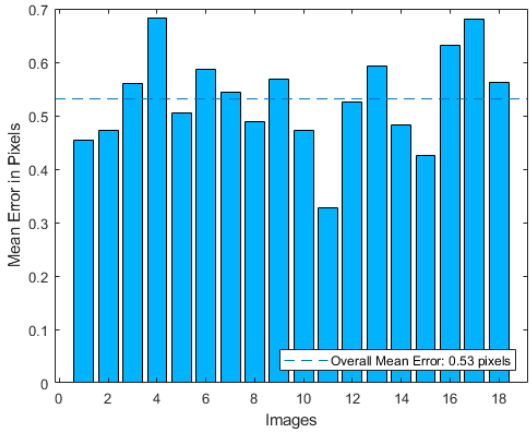


Figure 6.4: Reprojection error after re-calibration.

Figure 6.5: The upper image is distorted, while the lower image is undistorted after the calibration process. It is easy to recognize the difference since the quay represents straight lines in the real world.

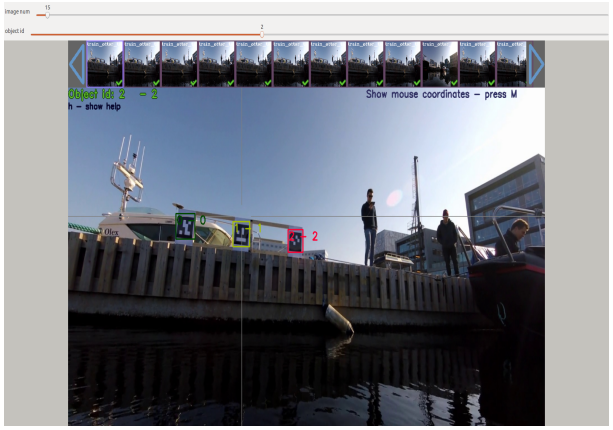


Figure 6.6: The annotation tool Yolo Mark in use during the labeling process.

6.2.2 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 is 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 6.6 shows 3 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.

6.2.3 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 aim to transfer the learning outcome of low level features that many object shares in common and also, avoid to train the model from scratch which can be very time-consuming, even for GPUs [45]. For this project, model parameters trained on a big and general dataset called ImageNet (reviewed in Chapter 5) was used such that the model already have 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 6.2, 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 refered to as the pre-trained weights. For more information about transfer learning, see here [45].

Table 6.1: The final choice of training parameters for the YOLOv3-spp architecture.

Training model and parameters	
Architecture	YOLOv3-spp
Batch size	64
Subdivision	64
Width	608
Height	608
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

Training parameters

Table 6.1 summarize the final choice of training parameters. The first horizontal row shows the network architecture, the second consider parameters related to GPU processing power, while the third focus 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 exists. 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.

- **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 tells us how many images is averaged over at the same time which helps generalizing the training more. 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 is trained for in parallel (i.e. the number of

images processed by the GPU simultaneously). With a quite high image resolution (608x608), it was necessary to divide the batch into 64 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 is resized to Width $\times$ Height. Choosing high resolution increase accuracy during training, but slower the processing time. A bit high image resolution (608x608) is chosen as it is desirable to learn detection of small objects during training. Channels is chosen to 3 as the model should process RGB images (e.g. 3 channels).

- **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 differ largely between iterations [46].
  A CNN typically have 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 reach an optimum during training. However, learning rates are all dynamic in modern gradient descent based networks. Some combination of problems and networks does train with learning rates bigger than "normal". 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 [46]. This short time is controlled by the burn in parameter. The value is set to 1000, meaning that the learning rate is lower between until iteration 1000.
  Max Batches: Max batches determines how many iterations the network is trained for and hence, the total training time. Different tests showed that 10000 iterations was enough to find an optimal model and to confirm over fitting 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 decides 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.

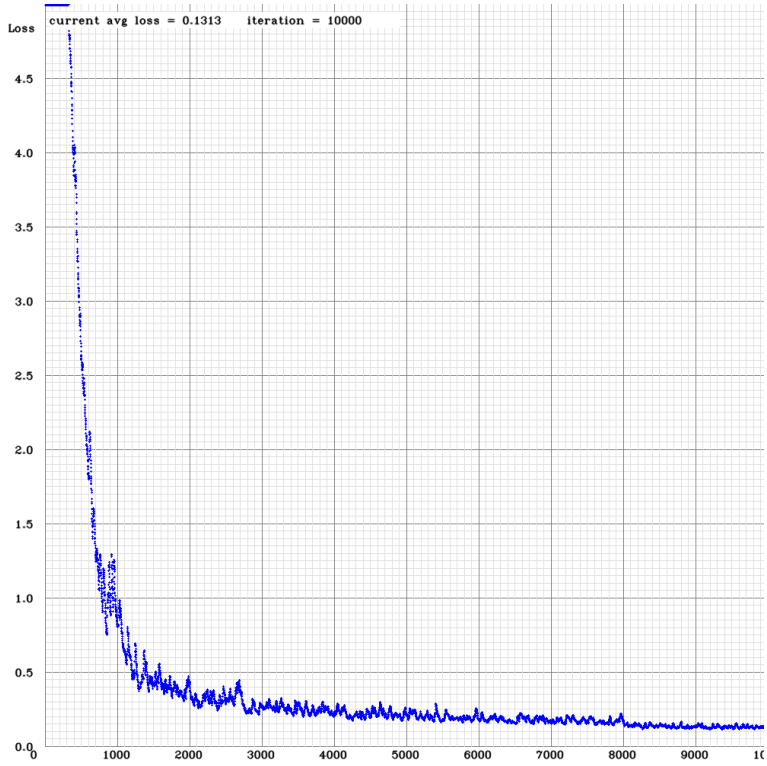


Figure 6.7: Training loss with the chosen training regime.

- **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 6.7 shows training loss during training.

Validate training

When the training period is succeeded, it is desirable to measure somehow which model parameters performs optimally across time (e.g. iterations or epochs). By definition, deep learning models usually learns the optimal model parameters towards the end as the learning is an optimization process (if training parameters is tuned correctly). But the main problem is 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 to the expectations of a human evaluator (i.e. overfitted to training data). Therefore, it is 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 exists like MaP, Recall and IOU as reviewed in Chapter 3.

Now, the model selection serves 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 6.2, 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 3.1.5.

The final model is therefore chosen as the one with the highest calculated MaP on the validation set, which is, after 6000 iterations. Notice how the MaP slightly decrease for the last computed models in Table 6.2. This may indicate overfitting. Now, in comparison with the normal validation practice described, this is not an optimal model selection. As seen from Figure 6.7, 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 work around could be to sample saved model parameters more frequently, but a local laptop could easily run out of memory as one model parameter file stores more than 250 MB. In addition, a small validation set also makes it harder to select an optimal model and one may also consider (in the future) to use other techniques such as cross-validation when working with small validation sets (or simply collect more data for validation).

Table 6.2: Comparison of MaP for different weights with training configuration.

MaP comparison on validation set	
Iterations	MaP (%)
1000	93.02
2000	98.77
3000	97.74
4000	99.73
5000	99.38
6000	99.74
7000	99.73
8000	99.71
9000	99.44
10000	99.11

6.2.4 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 validation procedure, MaP will be used as the main accuracy measure. In general, one can accept the model if the MaP value achieves a considerable high value. This value, of course, depends on how big and challenging the dataset is. For our case, the dataset is considered quite simple as the classes contains 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 99.73 % MaP among all the classes. This result will be reviewed and discussed in detail in Chapter 7.

6.2.5 Step 5: Operational use

If the results from the test set is 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 is constrained by the training data and in that context, data collection is essential. That is, 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 expensive will the data collection process be. In the next Chapter, additional test data will be used to evaluate how the detector performs (offline) in different scenarios.

Chapter 7

Results and Discussion

Having introduced the relevant datasets and the theory and implementation concerning YOLOv3, it is time to look into the results followed up by a discussion.

This Chapter can be divided into three parts. First, the results of the custom test-set is presented which tells us how well the detector performs on unseen data with accuracy metrics. Then, to get some context of how the detector works in real-time docking operations, additional test videos (without ground truth labels) is used to test the detector more extensively. The goal is to relate some of the observations in these test videos to the statistical metrics from the custom test-set. A video analysis containing a wide range of docking scenarios is presented and with this as a basis, several remarks are done. Some of the motivation behind is to test how well a deep CNN can help cameras, which are highly sensitive to environmental changes. E.g. how cameras can overcome its main weakness by using data-driven methods. The final goal is to test and observe the robustness of the learning based detector for outdoor docking operations using a suitable marker configuration. The different docking scenarios in clip 1-6 in 7.2 will be used to investigate this. At last, the results in 7.1 and 7.2 is summarized.

7.1 Custom Test-Set

This section presents results related to custom test-set with a special focus on accuracy metrics.

From Table 7.1, Average Precision (AP) for each class on custom test-set is presented along with True Positives (TP) and False Negatives (FP) for IOU threshold 0.5 (same threshold as the model is trained for). Recall that the data from the custom test-set represents 10% of the (randomly shuffled) custom data set. This ensures that a varied set of images is tested. Also recall how Precision and Average Precision (AP) is defined from 3.1.5. These metrics summarize the detection quality and is used to evaluate the detector.

Table 7.1: AP for different classes on custom test set.

custom test set			
Class	AP (%)	TP	FP
0	99.99	95	1
1	98.94	94	0
2	100	95	0
3	100	31	0

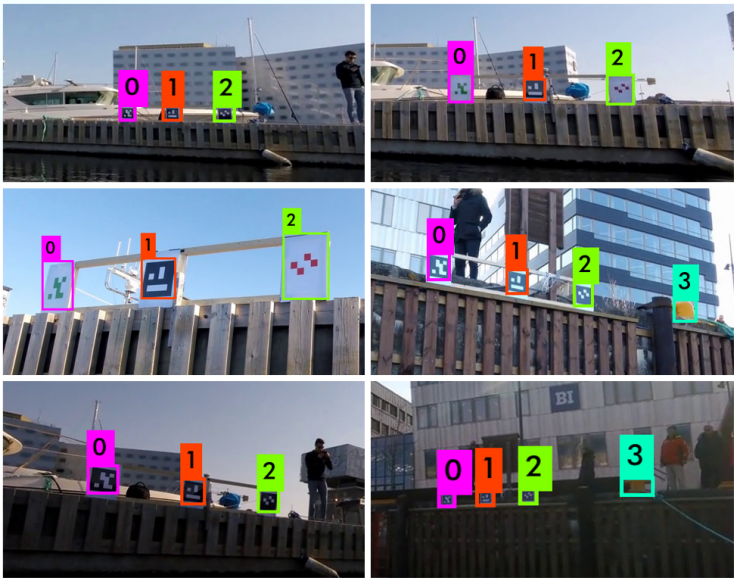


Figure 7.1: Various images from the custom test-set provided with detections, where natural landmarks occurs in two of them.

Figure 7.2: A false positive from the custom test-set.

Results

The overall mAP achieved across the classes is 99.73%, that is, the mean of the APs for class 0-3 (see Table 7.1). Figure 7.1 shows six different images from the custom test-set successfully detected and classified by the detector. These images also includes natural occurring landmarks and shows, together with Table 7.1, that there is no difference in detection quality between the ArUco markers and the natural landmark. It reflects some of the performance of the detector.

From Table 7.1, one can also observe that the model predicts a false positive (FP) in the custom test-set. Figure 7.2 visualize this, e.g. the ArUco marker placed in the middle is wrongly classified as class 0. This was however the only FP presented in the custom test-set.

Discussion

Table 7.1 shows that all individual APs achieves a very high value and in that sense, the model is able to detect and classify all the markers very accurately. As obtained, the model is able to detect the natural landmark with the same (extremely) high accuracy as the ArUco markers, although this landmark is more rich in sense of its features.

Another aspect is the weather during data collection. As it was a sunny day during the records, this may simplify the challenge of detecting markers. Other more challenging weather conditions can potentially introduce new challenges, i.e. how to robustly detect objects in presence of fog, heavy snow or rain or completely other light conditions.
Hopefully, a more comprehensive and challenging data set will be collected in future work in order to deal with these challenges.

The false positive presented in the custom data set is not a big surprise. The markers presented in Figure 7.1 are so small that it is hard to distinguish between the markers, even for humans. What is more remarkable is the fact that the CNN is able to detect and classify all the other markers correctly from long ranges (e.g. very small objects). With this observations, several questions arise. How small objects can a CNN detect and classify without producing false positives? And how small can objects be labeled in the annotation program (e.g. assigned ground truth to be learned)? At some point, the markers becomes so small that they may not be distinct from other features in the image. For instance, to label only some black and white pixels as an ArUco marker (i.e. a 3x3 pixel image) can be very dangerous. The consequence could be that the detector starts to detect a bunch of objects in the image, because it finds small patterns that are more or less equal to those black and white pixels assigned as the ArUco marker. This consideration is important when ground truth labels are made for the detector.

At last, the custom data set is considered quite homogeneous. This makes the learning task easier as the test data tends to be quite similar to the training data. The drawback is, of course, that the model loose some of its generality and may not be able to detect the

markers in other contexts. This may specially be a challenge for the natural landmark as it contains more features and may look quite different from different angles. In comparison, the ArUco markers are quite distinct and simple and the patterns looks quite similar from different angles.

7.2 Video Analysis

Now, as the custom test-set is relatively small and thus gives limited insight, it was decided to add more test videos to observe how the model performs on more unseen data. Hopefully, more extensively testing will find weaknesses and challenges that didn't showed up in the custom test-set. This may add value and more insights for future work. Table 7.2 shows the different test videos. Among the aspects that will be investigated in this section is comparisons of different marker configurations, detection quality on long and short ranges and related challenges to these aspects. Note that the same fine-tuned detector is used for all the videos (clip 1-6). The 6 different clips is concatenated into one single video. A link to the video can be found at:

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

Discussion

In this part, each clip is discussed in detail and several remarks is done for each clips. In addition, some general remarks not specified to a single clip is also provided.

Clip 1

The clip shows that the model is able to detect and classify the markers with high confidence score, both for long and short ranges. Not surprisingly, the confidence score oscillates a bit more at long ranges since the objects are very small and therefore, the model produce more "unsure" predictions. Figure 7.3 shows the confidence score of the markers for four different ranges during the docking operation. Observe that the left marker in the upper image has has a confidence score of 94% at one time instant, while the three other images (closer to the dockside) provides a higher confidence score, approximately 99-100%.

Clip 2 and 3

Clip 2 and 3 shows, not surprisingly, that the small ArUco markers (20 cm $\times$ 20 cm) cause more oscillating confidence scores during the long range detections and also produce several false positives. Comparison of detection quality with clip 1 verifies more or less that bigger markers should be chosen to simplify the detection task, specially when the USV is far from the dockside. When comparing with clip 1, it is clear that the A3 size markers (or maybe even bigger markers) should be chosen to ensure accurate and robust detections during the whole docking operation.

Table 7.2: The additional test videos tries to cover a wide range of scenarios.

Additional test data			
Clip	Video length	Marker configuration	Description
1	0.00-1.04	A3 black/white	The USV is slowly approaching the dockside from the side and un-dock when it is close to the markers. It explores both long and short range detections during the docking operation.
2	1.04-1.34	A4 black/white	The USV is standing more or less still and detect small ArUco markers from a far distance.
3	1.34-1.46	A4 black/white	The USV un-dock with the same small ArUco markers as in clip 2.
4	1.46-2.13	A3 colored	The USV is approaching the same dockside from another angle than the first three video clips. It provides middle and short range detections of colored markers.
5	2.13-2.48	Natural landmark	This clip contains a natural occuring landmark in addition to ArUco markers. All markers are located at another dockside. The USV is slowly approaching the dockside from a far distance.
6	2.48-3.04	Natural landmark	The USV is approaching the same dockside as in clip 5, but from a shorter range.

As earlier mentioned, it is very remarkable that the model is able to distinguish between the small objects, and specially the A4 size markers in clip 2 ans 3. It shows how well YOLOv3 robustly can detect objects at different scales. There is no way a human evaluator can do the same. Only prior knowledge about the scene, like the relative position between the markers can help us to distinguish between them. In fact, as long as all the markers can be detected (but not classified) and the order is fixed, knowledge about the relative order between the markers can be used to distinguish between them.

Clip 4

From clip 4, one can observe that there are no significant differences in detection quality between "standard" black/white ArUco markers and colored ArUco markers. This can be seen directly by comparing the confidence score of the marker in the middle with the left and the right marker. As all the markers, independent of its color, results in almost perfectly detections, they can all be used in a final marker configuration for the specific dockside. However, the color of the marker should in general be chosen to stand out from the scene, e.g. the choice of color depends on how the dockside as well as other backgrounds looks like. In other words, there is no guarantees for that the markers (used in this project) will perform the same way for other docksides.

One may also observe that although the USV is approaching the dockside from another angle, the light conditions are almost the same. And consequently, the detector does not seem to be affected by slightly different light conditions. Due to a short record period (approximately 2 hours in the morning), it was not possible to provide a dataset with more varying light conditions. It is therefore interesting, for future work, to record a data set with various light conditions during a day, in addition to various weather conditions from day to day. With such a dataset, one can experiment with the robustness by testing the detector under (very) different environmental conditions.

Clip 5 and 6

In clip 5 and 6, one can observe a natural occuring landmark (the yellow bollard), in addition to the three ArUco markers. Other natural landmarks was also considered, including the red ladder at the dockside. While it is easy to recognize the red ladder in clip 6, it becomes considerably harder to recognize the same landmark in clip 5 where the USV approach the dockside from a longer distance. Specially note how the red ladder struggling to stand out from the background scene (e.g. the brown/dark docking) during the whole docking operation. As seen, the scene can be completely changed due to different light conditions dependent of the angle and the range from the docking and therefore, natural landmarks must be carefully chosen to address this challenge. For the same reason, the yellow bollard was chosen as the natural landmark as it was quite easy to recognize it during the whole docking operation (e.g. in clip 5 and 6).

Another interesting aspect is a comparison of detection quality between fiducial markers and natural landmarks. As can be observed during clip 5 and 6, the ArUco markers achieves a higher overall confidence score compared to the natural landmark, although the difference is not considerably high. There may be several reasons why. First of all, the ArUco markers provides a very simple pattern to recognize and in that sense an easy detection task for a trained CNN. The natural landmark have, in comparison, more features and may vary more in different contexts. Secondly, the natural landmark only have 329 ground truth labels, while the ArUco markers have 945-955 labels (depending on the marker). Therefore, providing more ground truth labels of the landmark in the custom data set may reduce the small gap in confidence scores. However, the mentioned observations favorizes ArUco markers marginally. In addition, it is easier to detect keypoints from the corners of a an ArUco code compared to a more complex natural landmark. These keypoints will be used for stereo matching and 3D reconstruction later which makes ArUco markers a safe choice.

Other remarks

As seen in Figure 7.4, outdoor light also introduce other (maybe unexpected) challenges. The light reflecting from the water confuses the detector which produce a false positive. This may specially be a challenge when the water is flat (e.g. no waves) such that less distorted candidates can be produced in the water near the dockside. However, this observation was done during an early stage in the training phase (e.g. not reached the early stopping point). When training for more epochs, the model was able to reject such candidates. In addition, prior knowledge on relative positions between the markers can be used to reject false positives. For instance, since the detector produce two candidates of class 2 in Figure 7.4, one can reject the candidate in the water since the object of class 2 always occur to the right of the other markers (of class 0 and 1). Most likely, it will have a lower confidence score as well and one can pick the one with the highest confidence score. Simple filtering methods like this can be used to make the detection system more robust, but it is of course desirable that the CNN is able to make decisions robustly on its own.

Another aspect is that 3D reconstruction with stereo vision techniques will most likely not be performed at long ranges due to a relatively small baseline between the cameras, approximately 0.5 m. This is because stereo vision converges to monocular vision when the depth (between the cameras and markers) is significantly bigger than the baseline. The baseline is indeed constrained by the size of the USV. Therefore, the most critical is to have a detection system that works reliably on middle and short ranges from the dockside. However, monocular vision techniques together with prior knowledge of the geometry the markers can be used to produce 3D pose estimates as well. One approach is to develop a two-phase system where stereo vision is used for the last couple of meters, and monocular vision is used before the USV reach this distance. This way, one can extend the overall perception range. Where the phase-shift is executed precisely, e.g. where stereo vision does not provide accurate 3D pose estimates anymore, will be figured out in future work.

Lastly, from clip 1-6, several false positives can also be observed. This confirms, not

surprisingly, that there is a little gap in accuracy between the custom test-set (providing 96 test images) and the other test videos. On the other hand, the model performs almost perfectly on clip 1-6 and many of the false positives can in fact be rejected if the confidence threshold is increased (it is set to 0.25). On the other hand, a too high confidence threshold will tend to reject true positives. A suitable value in between should be chosen to deal with this tradeoff.

7.3 Summary

Throughout section 7.1 and 7.2, different aspects of the object detection model has been investigated, with a special focus on suitable marker configuration for simple and robust recognition during a docking operation. The results shows that there is a small gap in accuracy between the custom test-set and the additional videos. However, the detection quality is still considered very satisfying as seen from clip 1-6. Despite a well performing object detector is obtained, it is always relative to how challenging the data set is. Therefore, to increase insight and to make sure the detector works under more extreme conditions, it is proposed to collect a more comprehensive and challenging dataset for future work.

Furthermore, it is clear that fiducial markers like ArUco markers is suitable for outdoor marker recognition. The results showed that YOLOv3 was able to detect and classify the ArUco markers very well (as expected). The results also showed that all the fiducial markers performed well independent of its color, but not surprisingly decreased in detection quality when the smallest marker configuration (e.g. A4 size paper) were chosen. This was particularly visible at long ranges from the quay. The general approach is indeed to use an appropriate size of the markers depending on the detection range for the application. Natural landmark did also showed promising results. However, as a safe starting point, fiducial markers will most likely be used for future development.

To summarize, the results obtained in section 7.1 and 7.2 confirms that the fine-tuned object detection model (aka the backbone of the perception system) works well for outdoor marker recognition under docking operations. This lays a solid basis for future work and allows us to progress further and complete the pose estimation pipeline.

Figure 7.3: Comparison of detections for different ranges under a docking operation.

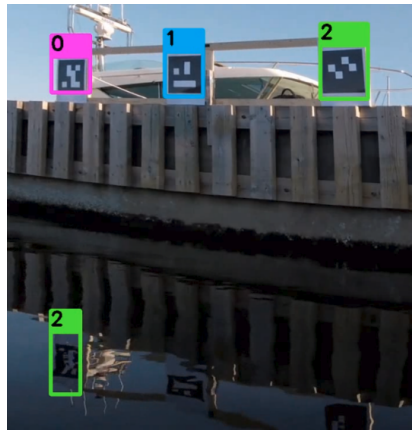


Figure 7.4: Light reflecting from the water confuse the detector during an early training stage.

Chapter 8

Conclusion

This Chapter concludes the thesis. That is, summarizing the most important findings and presenting some future challenges.

8.1 Overview

This thesis has investigated how a deep learning based detector is able to detect markers during various docking operations. The motivation was to provide robust detections with an elecro optical (EO) camera. The detector use the state-of-the-art object detection model YOLOv3 as a starting point.

Figure 6.2 summarizes the different steps needed to achieve fine-tuned object recognition for a specific application (such as marker detection). That is, how 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 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, a CNN is just a piece of the learning-based pipeline (although CNN is the core). In other words, there are more to object recognition than neural networks. Intuition about the quality, amount, relevance and variance 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.

8.2 Findings

8.2.1 Markers

We start by give some important remarks about ArUco markers and natural landmarks.

In addition to the great detection quality obtained with ArUco markers, we find the natural landmarks to perform well. One clear benefit of using natural occuring landmarks is that we use what is already in the scene. In [44], natural occuring cones is used instead of fiducial markers as reference objects. Indeed, placing fiducial markers around a track for racing cars is impractical. In comparison, a docking operation is more constrained and the environment around the dockside is assumed to be well known in prior. For this reason, a few fiducial markers placed around the quay may not be impractical. This could be static features easily recognisable at the dockside. Several companies use fiducial markers for visual-based navigation these days. This includes the robot *Handle* from Boston Dynamics which use several pairs of ArUco markers for close range operations (see Figure 2.5).

Further, we find the fiducial markers to be even more robust with respect to environmental changes. For instance, as outlined in 7.2, several natural landmarks looks completely differently depending on how the scene is affected by environmental changes (i.e. different light conditions). The red ladder in clip 5 and 6 is one such example.
In addition, the results shows that the model produce slightly lower (and more oscillating) confidence score for the natural landmark. Still, not many false positives are produced although the natural landmark is considered more challenging to detect due to its rich amount of features. This proves some of the great potential a data-driven detection model like YOLOv3 have.

8.2.2 The detection model

Next, we presents some important findings from the detection model.

Due to the promising accuracy metrics achieved from the custom test-set in 7.1 (99.73% mAP), we find the model to perform very satisfying. An important point is that the same model parameters is used for all the docking scenarios. This gives a somehow more general model compared to several fine-tuned models for each marker configuration. In an early development stage of the training process, this approach was also applied which gave, not surprisingly, even better results due to more homogenous datasets. On the other hand, these models did in fact produce significantly more false positives for docking operations looking slightly different from the training data.

Furthermore, YOLOv3 achieves surprisingly high detection quality for small objects. This may be due to recently improvements in YOLOv3. YOLOv2 often struggled with small objects due to loss of fine-grained features. As discussed in 3.3, YOLOv3 approach this problem by improving the feature maps. It predicts bounding boxes at 3 different scales, where 3 different anchor boxes is used for each scale which allows for more fine-

grained feature information.

At last, YOLOv3 is characterized by real-time performance. That is, the inference time is so low that humans is not able to observe a time delay while the input image is processed throughout the network. Clip 1-6 in 7.2 illustrates this clearly. As the reader may have observed, the focus on real-time performance have not been given a lot of attention in the thesis. The reason is simply that a navigation system not necessarily require position estimates in (hard) real-time. However, the tradeoff between speed and accuracy deserves some attention. As seen from the final training configuration in Chapter 6, the width and height of the input images is resized to 608×608 (which was also used at test time). This resolution achieved a good tradeoff between speed and accuracy. However, a lower resolution like 416×416 did indeed reduce the inference time making the detector even faster, but at the same time slightly less accurate. Particularly small objects in the image suffered from the reduced resolution (as it affected the detection quality negatively).

8.3 Future work

Deep CNNs for object detection is a field of heavy research and the methods are improved every year. By the time the thesis was written, YOLOv3 is considered a state-of-the-art recognition model suitable for many computer vision applications. One of these is auto-docking.

Further, as the proposed working system is developed (see Figure 1.1), one can benefit from research in the same field. In [47], a path planning algorithm for underactuated vehicles with limited field of view is proposed, 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. A stereo vision system provides only a limited field of view, even with fisheye lenses. Therefore, to control the vehicle such that markers are included in the camera view is particularly interesting (since the working system only works when the markers are visible for the cameras).

In Chapter 1, the overall idea for the project was introduced for the reader where the deep learning based detector was a minor part of the proposed working system. Figure 1.1 gives an overview of the different modules proposed to compute 3D pose estimates with stereo vision cameras. As obtained from the results in Chapter 7, the deep learning based detector is able to produce satisfying marker detections and classifications (for one camera feed).
For the master thesis, the plan is to continue the work and extend the pipeline. That is, to develop software to perform stereo matching of predicted bounding boxes and 3D reconstruction of matched points (see Figure 1.1). Software frameworks such as ROS will be investigated and ROS may be a good candidate for integration of the modules into the final working system. And obviously, calibration of the stereo camera system is needed to relate the left camera to the right and recover depth. The proposed extensions represents classical techniques within the field of computer vision.

At last, YOLOv3 have shown great detection capabilities for the recorded custom dataset (described in chapter 5), but it is still left unknown how the model handle a more challenging dataset. Therefore, it is proposed to collect a more comprehensive dataset for more robust detections (i.e. harsh and challenging weather conditions).

The plans are preliminary and can be adjusted during the work. Now, some of the mentioned suggestions for future work can summarized by the following list:

- Prepare a more compact PC for deep learning computations on-board a small USV.
- Mount the cameras on a camera rig and perform stereo calibration.
- Extract a pair of bounding boxes from left and right camera, and perform 2D-2D feature matching and triangulation for 3D pose estimation.
- Merge sub-modules into a complete 3D pose estimation system in Robotic Operating System (ROS).
- Synchronize and compare camera pose estimates with ground truth RTK GPS measurements.
- Record a broader data set covering several weather and light conditions, with a special focus on USVs in docking operations.

[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] "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

[3] T. Stenvold, "Verdens første autonome skip i drift skal erstatte 40.000 vogntogturer i året," 2017, accessed: 2019-06-17. [Online]. Available: https://www.tu.no/artikler/verdens-forste-autonome-skip-i-drift-skal-erstatte-40-000-vogntogturer-i-aret/382717

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

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

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

[7] 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/

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

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

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

[11] 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

[12] 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

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

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

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

[16] "Single camera calibrator app," 2019, accessed: 2019-06-17. [Online]. Available: https://se.mathworks.com/help/vision/ug/single-camera-calibrator-app.html

[17] "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

[18] 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/

[19] "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/

[20] "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

[21] 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

[22] "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

[23] "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

[24] "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

[25] 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

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

[27] 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/

[28] 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

[29] "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

[30] 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/

[31] 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

[32] 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.

[33] 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.

[34] 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

[35] 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

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

[37] 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/

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

[39] “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

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

[41] 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

[42] 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

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

[44] 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

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

[46] 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/

[47] 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