ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ
«ΣΥΣΤΗΜΑΤΑ ΑΥΤΟΜΑΤΙΣΜΟΥ»

Μεταπτυχιακή Εργασία

Position Estimation of Multiple Sea Vessels Using a Stereo-Based Camera System and Neural Networks

του

Γεωργή Στέφανου

Επιβλέπων Καθηγητής: Παπαλάμπρου Γεώργιος ΣΝΜΜ

Συνεπιβλέποντες: Χ. Παπαδόπουλος, Αναπληρωτής Καθηγητής ΣΝΜΜ,

Α. Γκίνης Αναπληρωτής Καθηγητής ΣΝΜΜ

ΑΘΗΝΑ 20.....

Η σελίδα αυτή είναι σκόπιμα λευκή.

Περίληψη

Η χρήση δύο καμερών και συνεπακόλουθα στερεοσκοπικής όρασης για την μέτρηση της θέσης αντικειμένων στο χώρο μέσω εικόνας, είναι μια τερχνική, που χρησιμοποείται ιδιαίτερα συχνά για την επίλυση προβλημάτων μέτρησης τριασδιάστατης θέσης επιθυμυτών αντικειμένων, ειδικά στον τομέα των αυτόνομων ρομποτικών χειριστών για την αντίλειψη του περιβάλλοντος.

Σε ελεγχόμενα περιβάλλοντα, όπως ένα επιστημονικό εργαστήριο οι τεχνικές υπολογιστικής όρασεις για την ανίχνευση και την μέτρηση της θέσης των επιθυμητών αντικειμένων είναι αρκετά ακριβείς. Αρκετά διαδεδομένοι για αυτό το πρόβλημα αισθητήρες προσδιορισμού θέσης με χρήση υπολογιστικής όρασης όπως το Kinect ή το Intel Real Sense, ένω επιλύουν το πρόβλήμα προσδιρισμού της θέσης των αντικειμένων, εντούτοις αυτό περιορίζεται σε μικρες αποστάσεις από τον οπτικό αισθητήρα, της τάξης των μερικών μέτρων. Αρα οι αισθήρες αυτοί δεν ενδείκνυται για την μέτρηση μέγαλων αποστάσεων.

Αισθητήρες απόστασης και μέτρησης χρόνου επιστροφής εκπεμπόμενου σήματος μπορούν να χρησιμοποιηθούν για την εξαγωγή τριασδιάστατης πληροφορίας της θέσης των αντικειμένων, δημιουργώντας point cloud. Παρόλο αυτά είναι δύσκολο να ανιχνευτή το επιθυμητό αντικέιμενο από το point cloud, πόσο μάλλον όταν απαιτείται ανίχνευση πολλών αντικειμένων. Επιπλέον οι σχεδίαση αυτών των αισθητήρων είναι αρκετά περίπλοκη και κοστοβόρα με αποτέλεσμα να κοστίζουν αρκετά ακριβά σε σύγκριση με τις κάμερες (ταξεις μεγέθους διαφορά κόστους).

Για την ναυσιπλοΐα τα πιο διάσημα συστήματα παρακολούθησης και τα πιο συνηθισμένα βασίζονται σε αισθητήρες χρόνου πτήσης (time of flight) όπως το Radar. Αν και είναι αρκετά ακριβείς τις περισσότερες φορές δεν μπορούν να παρέχουν τρισδιάστατες πληροφορίες θέσης. Για το σκοπό αυτό, η παρούσα διατριβή προτείνει έναν αλγόριθμο πραγματικού χρόνου για τον εντοπισμό και την εκτίμηση της θέσης 3D πολλαπλών πλοίων χρησιμοποιώντας μόνο δύο εικόνες της ίδιας σκηνής από ένα στερεοσκοπικό σύστημα κάμερας.

Πιο συγκεκριμένα, η παρούσα διατριβή παρουσιάζει το σχεδιασμό, την ανάπτυξη και την εγαρμογή ενός στερεοσκοπικού συστήματος εντοπισμού θαλάσσιων σκαφών, με στόχο την παροχή ακριβών και ισχυρών αποτελεσμάτων θέσης που θα μπορούσαν να χρησιμοποιηθούν για την αποφυγή σύγκρουσης στον τομέα της αυτόνομης ναυτιλίας. Οι σχεδιαστικοί μας στόχοι είναι να παρέχουμε μια λύση που να είναι φθηνότερη από τις παραδοσιακές λύσεις, αλλά ταυτόχρονα να παρέχει περισσότερες πληροφορίες σχετικά με την θέση για τα αντικείμενα που εντοπίστηκαν, με ακρίβεια και ευρωστεία. Για το λόγο αυτό, μετά από εξαντλητική αναζήτηση στην τρέχουσα διαθέσιμη τεχνολογία αλγορίθμων αντίληψης, προτείναμε ένα στερεοσκοπικό σύστημα που εκμεταλλεύεται τις δυνατότητες των νευρωνικων δικτύων για την ανίχνευση θαλάσσιων σκαφών.

Επιπλέον παρουσιάζεται και αναπτύσεται ένας βελτιωμένος γρήγορος αλγόριθμος εύρεσης του ορίζοντα που χρησιμέυει στην ακύρωση λανθασμένων εκτιμήσεων από το νευρωνικό δίκτυο. Στην συνέχεια με την χρήση των οπτικών πληροφοριών από τις εικόνες του στερεοσκοπικού συστήματος και με την χρήση ενός βελτιωμένου αλγορίθμου εκτίμησης της τρισδιάστατης θέσης, διορθώνονται τυχόντα σφάλματα στην σχετική θέση της μιας κάμερας

ως προς την άλλη. Η καινοτομία του συγκεκριμένου αλγορίθμου έγκειται στο ότι δεν χρείαζεται εξωτερικά σημεία (markers) τοποθετημένους πάνω στο σκάφος σε ακριβή θέση από τις στερεοσκοπικές κάμερες. Ο πλήρης αλγόριθμος εύρεσης και εκτίμησης τρισδιάστατης θέσης θαλάσσιων σκαφών, ενσωματώνεται σε κατάλληλα ενσωματωμένο υπολογιστικό σύστημα που σχεδιάστηκε, πετυχαί οντας 5FPS συνεχών εκτιμήσεων, κάτω από διαφορετικές συνθήκες φωτισμού.

Τέλος η ακρίβεια και ευρωστεία του αλγορίθμου και συνολικά του ενσωματωμένου συστήματος, στην εύρεση και εκτίμηση της τρισδιάστατης θέσης των θαλάσσιων σκαφών, αξιολογείται εκτενώς μέσα από αρκετά πειράματα.

Abstract

In computer vision, triangulation via arranging two cameras in a stereo setup has become the norm in order to estimate the 3D pose of a particular object of interest and is used in most autonomous robots to help perceive the environment.

In experiments limited to laboratory environments, classical computer vision techniques such as stereo correspondence search and triangulation work decently well. Moreover, one could utilize off-the-shelf equipment such as the Kinect sensor or Intel RealSense. The drawback being that such sensors have limitations when taken outdoor and have only a limited range (up to a few meters). This makes it infeasible to use such setups for long-range pose estimation.

Range and time-of-flight sensors can be used to extract 3D information using raw data provided by such sensors from point clouds. But again, detecting particular objects in such point clouds is non-trivial. Having to do this for multiple objects of interest only compounds the task. Although, time-of-flight sensor manufacturers are trying to cut down costs and make such with competitive prices but are still a long way from manufacturing accurate sensors available at a competitive price such as cameras (which are orders of magnitude cheaper and provide most information per cent).

In maritime industry the most famous surveillance systems and the most common are based on time-of-flights sensors like Radar. Although they are quite accurate most of the times they could not provide three-dimensional positional informations. To this end this thesis propose a low-latency real-time pipeline to detect and estimate 3D position of multiple Sea-Vessels using just two images of the same scene from a stereo based camera system.

More specifically this thesis presents the design, development and implantation of a stereoscopic Sea-Vessels detection and localization system, aiming to provide accurate and robust results that could be used for avoidance collision in autonomous shipping. Our design goals are to provide a solution, which could be orders cheaper than the traditional solutions but providing more positional information for the detected objects, accurately and robustly. For this reason, after exhaustive search in current available hardware and perception algorithms technology we proposed a stereoscopic system exploiting neural networks for detecting Sea-Vessels.

An improved fast horizon line detection pipeline is also presented and implemented in order to eliminate false Sea-Vessel detections. For estimating the 3D position of those detections, our system exploits the informations provided by the stereoscopic view. Furthermore, an improved 3D estimation algorithm is proposed, using just as measurements the detected Sea-Vessels in the current frame. This eliminate the need of precisely positioning specific markers in Ship's hull and calibrate them with respect to the stereo rig. Our real time prototype system is capable of achieving 5FPS of continues detecting and pose estimating of Sea-Vessels in different Sea environments

Finally the performance of the system is evaluated by conducting several tests in different lighting conditions, after the testing and approval of each sub-module of the system

Ευχαριστίες

Η ολοκλήρωση της συγκεκριμένης μεταπτυχιακής διατριβής δεν θα ήταν δυνατή εάν δεν υπήρχαν συγκεκριμένα άτομα που μου παρείχαν την απαραίτητη γνώση και στήριξη όπου και όποτε χρειάστηκε.

Αρχικά τον καθηγητή κ. Γεώργιο Παπαλάμπρου, που ήταν ο εμπνευστής της συγκεκριμένης εργασίας και με βοήθησε σημαντικά καθ΄όλη τη διάρκεια με τις γνώσεις του, καθώς και στην ποιοτικότερη ανάλυση της διαδικασίας και των παραγόμενων αποτελεσμάτων, αλλά και της φιλικής σχέσης και υποστήριξης που μου παρείχε.

Τέλος, ένα πολύ μεγάλο ευχαριστώ στην οικογένεια μου και στους φίλους μου, που με βοήθησαν, ο καθένας με τον δικό του μοναδικό τρόπο να φέρω εις πέρας αυτό το κοπιαστικό έργο.

Σε όλους αυτούς, λοιπόν, τους ανθρώπους που πέρασαν μέρος του χρόνου τους για να με ενθαρρύνουν και να με βοηθήσουν με κάθε τρόπο για να πραγματοποιήσω όχι μία ακόμα εργασία, αλλά ένα προσωπικό όνειρο, ένα μεγάλο ευχαριστώ.

Contents

List of Figures

Chapter 1

1 Introduction

1.1 State-of-the-Art

This Chapter describes briefly why Vessel Detection and 3D Pose estimation is important for autonomous shipping. It then describes why existing traditional surveillance systems lack to fulfill the requirements of the autonomous shipping. A novel stereo based camera system with the use of deep neural networks, as well as computer vision techniques is introduced, which eases some of the problems of traditional navigational devices equipped on Sea-Vessels.

The interest in remote-controlled and autonomous ships has been increasing in the fields of marine industry and information technology[11][1][40][16][31]. A key technology in autonomous ships is the situational awareness equipment required to safely operate and navigate.

Automatic surveillance of coastal areas is gaining importance due to the increasing global ship traffic: Tankers, container ships, and bulk carriers are the most important means of transportation of our time [39]. Moreover, the presence of environment protection issues and new dangerous threats coming from the sea, including illegal smuggling and fishing, immigration, oil spills and piracy, encourage the development of intelligent monitoring systems.

Ocean-going vessels are equipped with various electronic devices for navigation, such as automatic radar plotting aid (ARPA) and automatic identification system (AIS). However, existing devices are not perfect, and the navigational abilities of ships are restricted by these devices. In addition, the AIS is not mandatory for ships under 300 GT, and small vessels cannot be detected by the AIS, which increases the collision risk. The occurrences of accidents at sea prove that existing navigational devices are inadequate. Studies show that up to 75 to 96% of maritime accidents and casualties are due to some form of human error [29][15].

More specifically, an important cause of collision is an improper look-out being maintained by navigation officers, which accounts for 86% of collisions [34]. This makes the authorities and researchers pay more attention to increase the navigational safety[31]. It is desirable to have many choices for safety navigation which support the applicability of The International Regulations for Preventing Collisions at Sea (COLREGs) (IMO, 1972) in which the target object location and course are essential information for obstacle avoidance. Figure 1.1 illustrates a crossing situation according to COLREGs rules. In the figure, V0 and V1 are speeds of own vessel and target vessel, respectively and X-Z is the coordinate system for own vessel. As it can be seen from the figure, the action for the collision avoidance depends on the location, course and the speed of the target vessel. Recently some studies are carried out for

the application of COLREGs in autonomous surface vehicles. These studies assume that the obstacles are already detected and their algorithm makes localization and mapping in accordance with COLREGs rules ([29], [5],[36]). However Sea-Vessel detection and 3D Position Estimation is a very complex step for collision avoidance, which is the topic of this thesis.

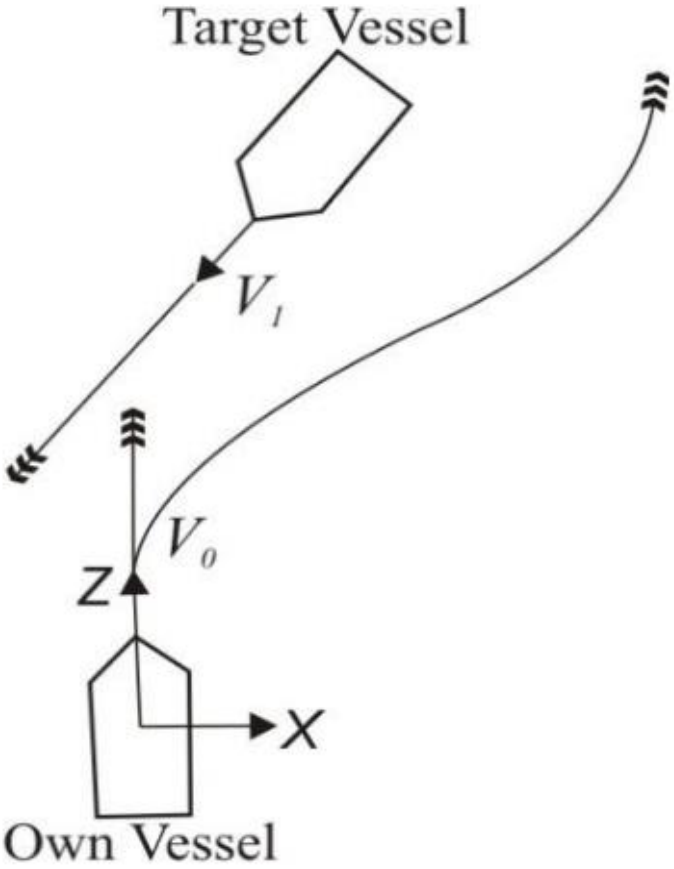

Figure 1.1 Crossing (Own vessel gives way) [16].

Especially, automatic Sea-Vessel detection becomes more important for the safety navigation of ships. Automatic ship detection is reported in some papers ([33],[26]), which utilized digital images. However, these studies use a single camera. In this case, it is very difficult to obtain the location of a target. The [15][16] have proposed a new approach for the detection and localization of other ships by means of a stereo vision system. Although they are using stereo vision system they need a huge amount of computing power and the resulting detection is not real time.

1.2 Engineering Challenges

Two are the main engineering challenges hinder the development of a Sea-Vessel detection system capable of estimating the 3D position of each Vessel using conventional sensor technology.

- System Design

  With the current micro-processor technology, it is challenging to build a 3D position Vessel detection system that is capable of flawless detection and localization of Sea-Vessels in real time, using conventional sensors like cameras and small micro-processors, but still matches the size, cost, performance and delay preferences for use in maritime industry, from various types of Sea-Vessels. The problem arise because of the use of conventional cameras as the only sensor of the system, which they generate a huge amount of raw data ready to be processed in every frame.

- Efficient Detection Algorithm

  A surveillance system capable of detecting Vessels and estimate the 3D position of each, must have an efficient detection algorithm, which is fast and low computing power consuming. Traditional choices for vision systems are conventional computer vision techniques, which are although consuming in computing power [15],[16].

1.3 Scope of Investigation

A stereo camera based system is presented with the use of neural networks and computer vision techniques in order to precisely detect and estimate the 3D pose of several vessels in each camera frame. Especially a stand alone micro-computing system with two cameras and an external TPU(Tensor Processing Unit) was designed and implemented, capable of capturing frames, recognize and detect Vessels in every frame and then estimate their 3D pose relative to the ship in real time. Novel position estimation algorithm was implemented, with rotational position error of stereo cameras elimination. Test were carried out to determine how the actual embedded system performed in practice.

1.4 Thesis Outline

- Chapter 2 Hardware Design Concept – Technology
- Chapter 3 Sea-Vessels Detection Algorithm
- Chapter 4 Fast Horizon Line Detection Algorithm
- Chapter 5 Stereo Fusion & 3D Position Estimation
- Chapter 6 Experiments
- Chapter 7 Conclusion

Chapter 2

2 Hardware Design Concept-Technology

Setting up a perception pipeline on a real-time system involves a coherent and clever interplay between software and hardware. Although, undermined more often than not, choosing appropriate hardware can greatly benefit the whole system. With proper choices it can improve raw sensor dataflow management, sub-system performance and eventually the overall throughput of the system. To ensure an efficient and effective perception system we build a customized camera hardware setup with choices based on goals and requirements defined at the onset. We postpone the discussion of the novel "Sea-Vessel detection" scheme and consequently position estimates of multiple vessels at every image frame by exploiting stereo vision configuration until the next chapter. The main focus of this chapter is to discuss the development of the hardware setup and raw data acquisition which is as crucial, if not more, as the software and processing of raw data.

2.1 Design Choices

As mentioned earlier, choosing the most relevant sensor based on the application can already have a monumental impact on the pipeline. Keeping multiple constraints such as limited on-board computation and sharing with other modules, sensor envelope and the package goals, a computational cost effective detection pipeline was implemented. The time delay from the raw image information to be taken till the end of the detection and estimation pipeline played a central role to the overall design hardware architecture.

2.1.1 Choosing the right Sensor

The perception pipeline has a two camera setup. Two cameras, on the extremities, act as a stereo pair to help triangulate detected Sea-Vessels and compute the position in space regarding to the coordinate system attached to left camera. Parameters that are thoroughly considered:

- Noise and corruption free transmission: The speed and quality transmission of raw image data from the cameras to the micro-processing board plays a key role to the overall throughput of the system. One more key factor is the need of taking frames simultaneously from both cameras. These requirements and with the simplicity that USB communication is provide, led as to adopt it as the main communication protocol of each module in our system.

- Neutralizing sudden changes in lighting: To ensure that glares and sudden lighting changes do not render the cameras useless, CMOS (Complementary Metal-Oxide Semiconductor) sensors are chosen over CCD (Charged Coupled Device). CCD sensors are susceptible to washed out image pixels due to glare while CMOS based sensors are not.

2.1.2 Additional Hardware Parameters

After the selection of the sensors, it is time for the computational cost of the Sea-Vessel detection pipeline to be considered and the way that should be treated the raw data, that coming from the camera. The following parameters led as to the selection of the embedded computation system.

- Computing Power: As mentioned previously cameras generating a huge amount of raw image data which should be processed by a powerful computation system in order to achieve satisfactory results in respect to the time of vessel identification. Further more, the computation system should provide an acceptable amount of external-peripheral modules connectivities and should be capable of monitoring them flowless. These parameters in addition with the need of a compact, low power consuming and powerful computational board led as to the adoption of a Raspberry Pi 4

- Simplicity to configuration: Another key parameter was that we needed a system that is simple to configure to our needs and yet easy to program it. The supported Debian OS which is based on Linux and the vast community of Raspberry Pi played a crucial role to the selection of the board

- Another reason led as in this decision was that, we wanted a system which could be a stand-alone computing system, carrying external monitors and all the sensors and modules and be commercially available and low cost.

- Detection-Pipeline: The traditional computational-heavy computer vision techniques and the progress of technology in the field of creating specialized hardware for tensors processing, led as to switch our design and implementation of the detection pipeline to the deep neural networks. That gave the possibility to build a system which is scalable and facilitate the computational load of the main processing unit of the system. For this reason the Intel Neural Compute Stick 2 (NCS2) [18] was chosen as the main processing unit of the detection pipeline over other modules. This peripheral is fully functional and configurable with the Raspberry Pi [17] and also provides, at a low-cost, a powerful Vision Processing Unit (VPU). The VPU takes advantage of the computation power of 16 tensor processing units with a shared high speed memory. The NCS2 is connected to the main system (Pi 4) via a USB3 port of the main board.

- Open Source Software: was an essential parameter of the system implementation. Since we wanted a hardware that is supported by open source

and publicly available software we ended up with the above hardware selection[17][27][37].

In the following picture the Raspberry Pi 4 with its main modules is depicted. Following a table with the most essential characteristics of the Raspberry Pi is prevented.

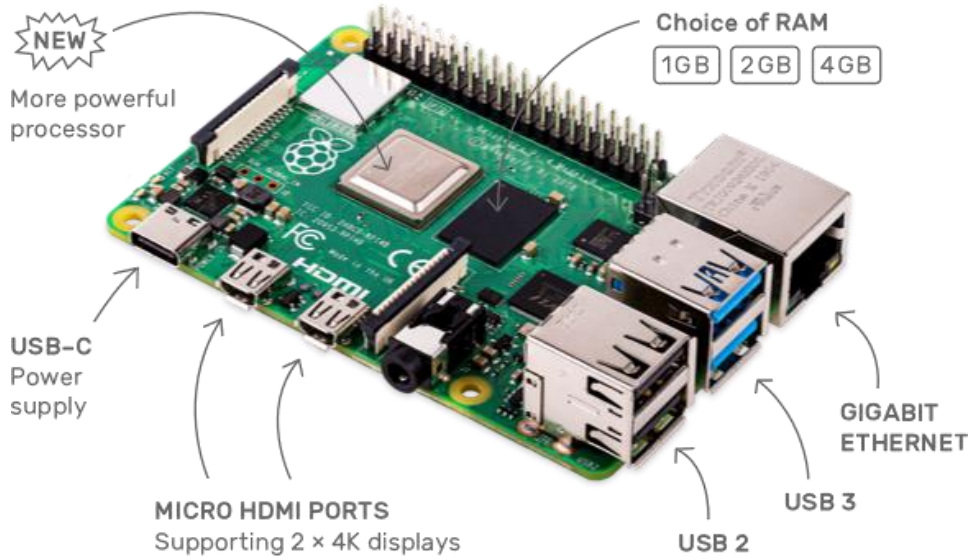

Figure 2.2 A view of the most essential raspberry pi4 characteristics

Table 2.1 Most Important Raspberry Pi4 Model B Specifications

CPU	Quad core Cortex-A72 (ARM v8) 64-bit
CPU Clock	1.5GHz
RAM Memory	4GB LPDDR4-@3200MHz SDRAM
Serial Communication	2 USB 3.0 ports; 2 USB 2.0 ports, I2C, SPI
Display Support	2-lane MIPI DSI display port, 2×micro-HDMI ports
Embedded Graphics	500 MHz VideoCore VI, OpenGL ES 3.0 graphics
Wireless Communication	802.11ac (2.4 / 5 GHz), Bluetooth 5.0
Power Consuption	3.4 watts
Power Driving	3A, 5V
Size	88 x 58 x 19.5mm

2.2 System integration

The Raspberry Pi4 (Figure 2.3) which is the main computing board of the system was connecting with the cameras and the Neural Compute Stick (NCS2) via USB (Figure 2.3). Furthermore a monitor was connected to one of the HDMI ports. It was found that the temperature of the Raspberry Pi board was increasing a lot during inference and for this reason an active cooling system was attached, as shown bellow (Figure 2.3).

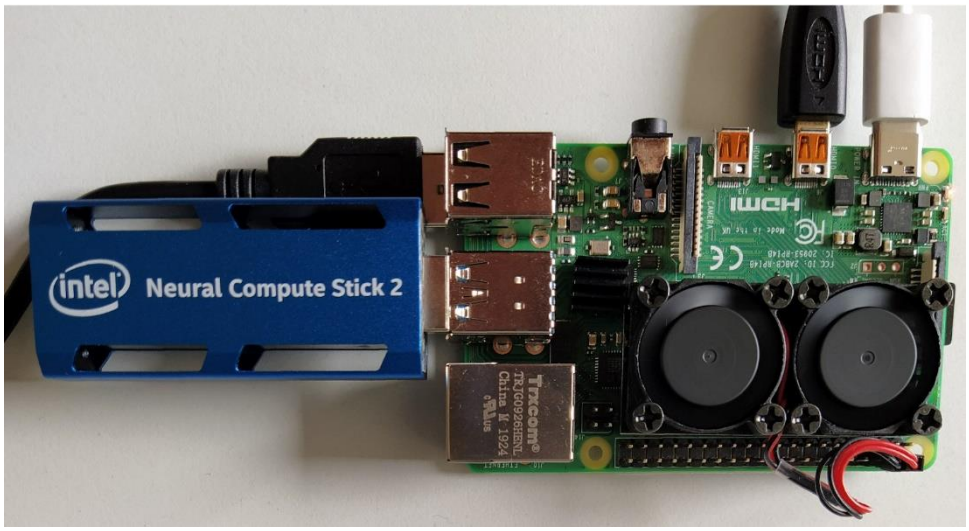

Figure 2.3 Presents our hardware computational selection, comprises of Raspberry Pi4 as the main system control board and the Intel Neural Compute Stick2 which executes our deep neural network

2.3 Image Acquisition

The cameras that were used are shown in Figure …. They acquire raw image frames with a rate of maximum 30 frames per second at a resolution of 1280x720 pixels . Another key feature of this camera is that it has a fixed focal length, which ensures that there is no need to compute it in every frame compared with automatic focus cameras. As it is already known the cameras are connected to the main board via USB.

Table 2.2 Camera Sensor - Logitech C270 Specifications

Resolution	720p/30fps
Focus	Fixed
Field of View	60 degrees

We also designed and builded in a 3D printer bases for our cameras in order to form a stereoscopic image acquisition system, as it is shown in figure 2.5.

Figure 2.4 An image of the camera sensor that were used

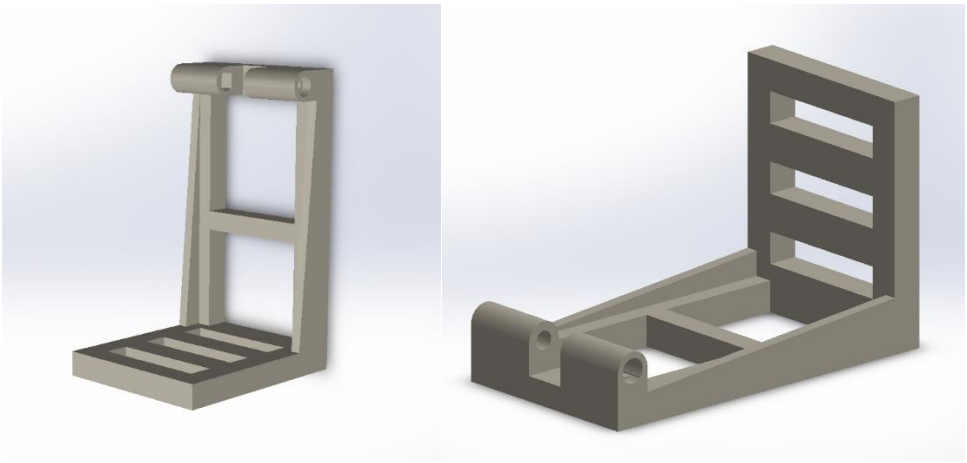

Figure 2.5 Two different views of our designed camera bases in solidworks.

2.4 Neural Network Processing Unit

As described already special study was carried out in the time execution of the detection pipeline. This led us to bypass the limited onboard computational power concerning to tensors multiplications of the CPU and the embedded GPU of the Raspberry Pi4, which have not sufficient tensor processing throughput and to substitute with the Intel Neural Compute Stick2 (Fig. 2.5).

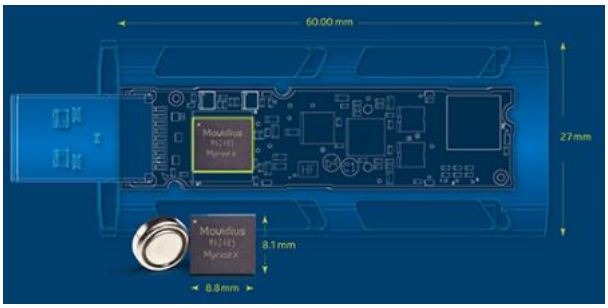

Figure 2.5 A view from the neural compute stick. One can gain a basic intuition of the size of the stick and the arrangement of the processing unit.[18]

This device is based in a tensor processing unit, which exists in each of the 16 Cores (Shaves) and leverages the convolutional operations that is executed in each of the millions neurons of the neural network, that we used. Apart from that there is also some hardware implemented computer vision operations, which means that in a single clock the computation is done. In the following figure 2.6 one can see the semantic architecture of the TPU (Tensor Processing Unit)

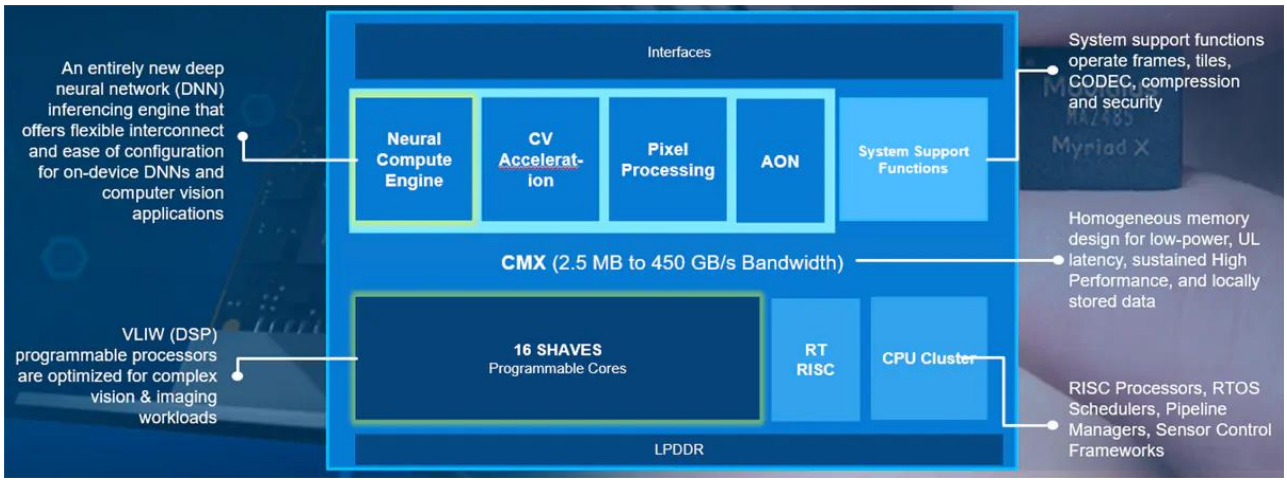

Figure 2.6 Schematic architecture of the Tensor Prosecing Unit of the Neural Compute Stick [18]

Chapter 3

3 Sea-Vessels Detection Algorithm

This chapter shifts the focus on how to detect multiple Sea-Vessels from a single image exploiting deep neural networks. Although, it is an ill-posed problem but with a priori information and a new dataset regarding to the desired new object detection task, a well trained neural network can be exploited and reconfigured for our new task. At the beginning of this chapter the basic theory of deep neural networks is presented, focused on the problem of object detection. Then the benefits of knowledge transfer in neural networks is thoroughly discussed. Finally, the methods and the implementation of our deep neural network based on YOLO architecture is deeply discussed and some visual results are presented.

3.1 Anatomy of neural network

As it is already known machine learning is about mapping inputs (such as images) to targets (such as the label "cat"), which is done by observing many examples of input and targets. It is also known that deep neural networks do this input-to-target mapping via a deep sequence of simple data transformations (layers) and that these data transformations are learned by exposure to examples. But how this learning happens, concretely? The specification of what a layer does to its input data is stored in the layer's weights, which in essence are a bunch of numbers. In technical terms, it could be said that the transformation implemented by a layer is parameterized by its weights (see figure 3.2). (Weights are also sometimes called the parameters of a layer.) In this context, learning means finding a set of values for the weights of all layers in a network, such that the network will correctly map example inputs to their associated targets. But here's the thing: a deep neural network can contain tens of millions of parameters. Finding the correct value for all of them may seem like a daunting task, especially given that modifying the value of one parameter will affect the behavior of all the others![13]

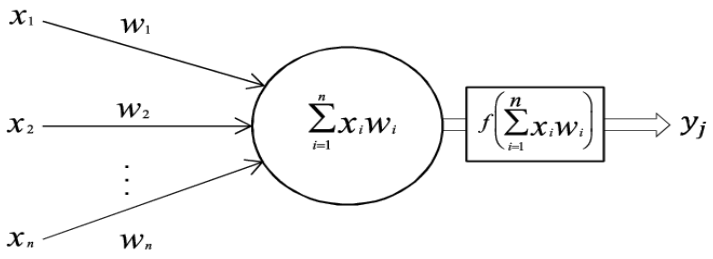

Figure 3.1 The fundamental cell of a neural network, a neuron.

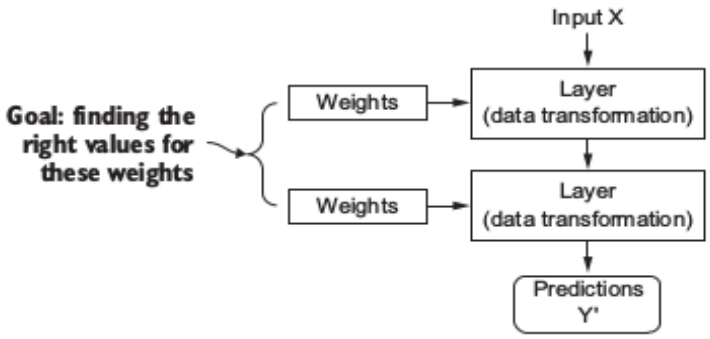

Figure 3.2 A neural network is parameterized by its weights.

To control something, first we need to be able to observe it. To control the output of a neural network, we need to be able to measure how far this output is from what it is expected. This is the job of the loss function of the network, also called the objective function. The loss function takes the predictions of the network and the true target (what we wanted the network to output) and computes a distance score, capturing how well the network has done on this specific example (see figure 3.3).

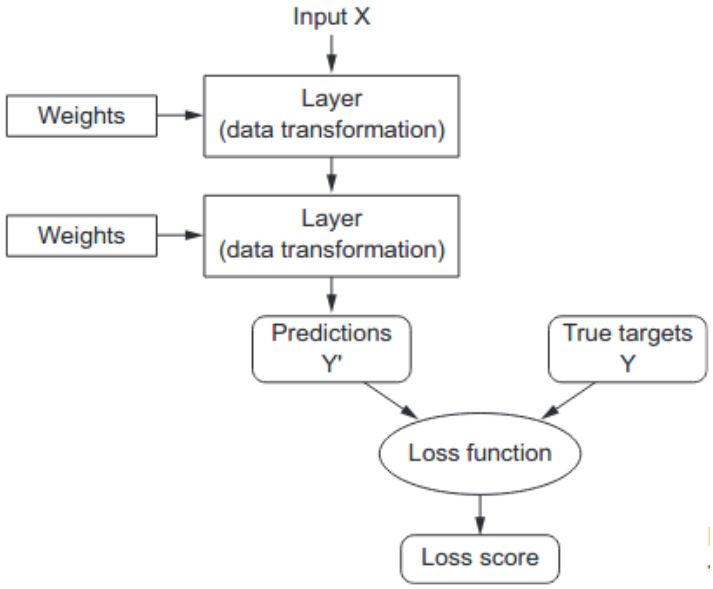

Figure 3.3 The loss score is used as a feedback signal to adjust the weights.

Initially, the weights of the network are assigned random values, so the network merely implements a series of random transformations. Naturally, its output is far from what it should ideally be, and the loss score is accordingly very high. But with every example the network processes, the weights are adjusted a little in the correct direction, and the loss score decreases. This is the training loop, which, repeated a sufficient number of times (typically tens of iterations over thousands of examples), yields weight values that minimize the loss function.

A network with a minimal loss is one for which the outputs are as close as they can be to the targets: a trained network. Once again, it's a simple mechanism that, once scaled, ends up looking like magic[13].

Training a neural network revolves around the following objects:

- Layers, which are combined into a network (or model)
- The input data and corresponding targets
- The loss function, which defines the feedback signal used for learning
- The optimizer, which determines how learning proceeds

The interaction of all them can be visualized as illustrated in figure 3.4 : the network, composed of layers that are chained together, maps the input data to predictions. The loss function then compares these predictions to the targets, producing a loss value: a measure of how well the network's predictions match what was expected. The optimizer uses this loss value to update the network's weights.

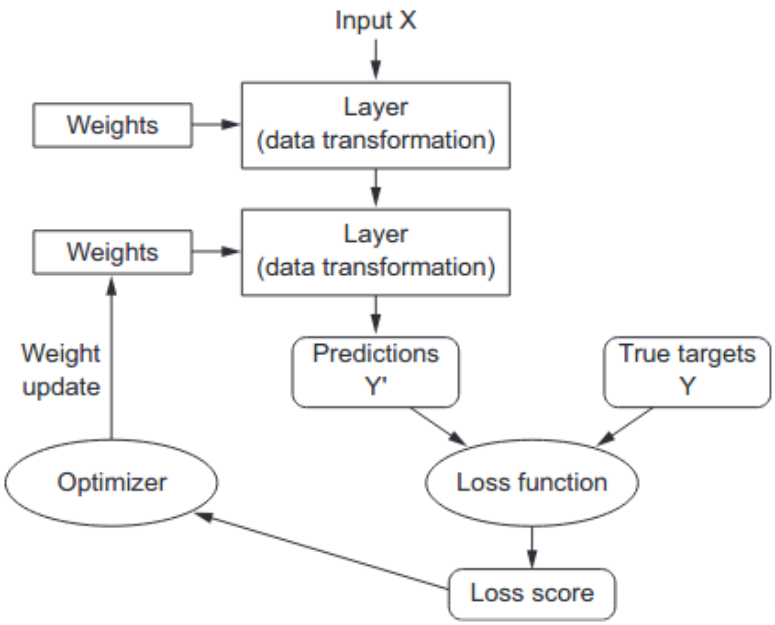

Figure 3.4 A generic diagram of how neural-networks trained and how they constructed. The loss score is used as a feedback to adjust the weights.

The fundamental data structure in neural networks is the layer. A layer is a data-processing module that takes as input one or more tensors and that outputs one or more tensors. Some layers are stateless, but more frequently layers have a state: the layer's weights, one or several tensors learned, which together contain the network's knowledge.

One can think of layers as the LEGO bricks of deep learning.

3.1.1 Neural Models: networks of layers

A deep-learning model is a directed, acyclic graph of layers. The most common instance is a linear stack of layers, mapping a single input to a single output. The most common network topologies, which although are not covering all the cases, are:

- Two-branch networks
- Multihead networks
- Inception blocks

The topology of a network defines a hypothesis space. A meaningful definition of machine learning can be as "searching for useful representations of some input data, within a predefined space of possibilities, using guidance from a feedback signal." By choosing a network topology, the space of possibilities (hypothesis space) is constrained to a specific series of tensor operations, mapping input data to output data. What will then be searched for is a good set of values for the weight tensors involved in these tensor operations.

Picking the right network architecture is more an art than a science; and although there are some best practices and principles one can rely on, only practice can help in becoming a proper neural-network architect. The next few chapters will concentrate in explicit principles for building neural networks and developing intuition as to what works or doesn't work for specific problems[13].

3.1.2 Loss functions and optimizers: keys to configuring the learning process

Once the network architecture is defined, there is still need two more things to be defined[13]:

- Loss function (objective function)—The quantity that will be minimized during training. It represents a measure of success for the task at hand.
- Optimizer—Determines how the network will be updated based on the loss function. It implements the weight parameters update strategic at the end of each training iteration, could be for example a specific variant of stochastic gradient descent (SGD).

A neural network that has multiple outputs may have multiple loss functions (one per output). But the gradient-descent process must be based on a single scalar loss value; so, for multi-loss networks, all losses are combined (via averaging) into a single scalar quantity.

Choosing the right objective function for the right problem is extremely important: the network will take any shortcut it can, to minimize the loss; so if the objective doesn't fully correlate with success for the task at hand, the network will end up doing things you may not have wanted. Imagine a stupid, omnipotent AI trained via SGD (Stochastic Gradient Descent) [13], with this poorly chosen objective function: "maximizing the average well-being of all humans alive." To make its job easier, this AI might choose to kill all humans except a few and focus on the well-being of the remaining ones—because average well-being isn't affected by how many humans are left. That might not be what we intended[13]! Finally, it could be argued that all neural networks that are build will be just as ruthless in lowering their loss function, so one should choose the objective wisely, or will have to face unintended side effects.

Fortunately, when it comes to common problems such as classification, regression, and sequence prediction, there are simple guidelines that can be followed, to choose the correct loss. For instance, binary cross-entropy could be use for a two-class classification problem, categorical cross-entropy for a many-class classification problem, meansquared error for a regression problem, connectionist temporal classification (CTC) for a sequence-learning problem, and so on. Only when working on truly new research problems there is need to be developed new-customized objective functions.

3.2 Fundamentals of Machine Learning

Having a deep intuition of how neural-networks are constructed and the fundamental elements that they consist of, it is time to structure into branches the scientific field of machine learning. Although it is a vast field with a complex subfield taxonomy. Machine-learning algorithms generally fall into four broad categories, described in the following sections[13].

3.2.1 Supervised learning

This is by far the most common case. It consists of learning to map input data to known targets (also called annotations), given a set of examples (often annotated by humans). Generally, almost all applications of deep learning that are in the spotlight these days belong in this category, such as optical character recognition, speech recognition, image classification, and language translation. Although supervised learning mostly consists of classification and regression, there are more exotic variants as well, including the following[13] (with examples):

- Sequence generation—Given a picture, predict a caption describing it. Sequence generation can sometimes be reformulated as a series of classification problems (such as repeatedly predicting a word or token in a sequence).
- Syntax tree prediction—Given a sentence, predict its decomposition into a syntax tree.
- Object detection—Given a picture, draw a bounding box around certain objects inside the picture. This can also be expressed as a classification problem (given many candidate bounding boxes, classify the contents of each one) or as a joint classification and regression problem, where the bounding-box coordinates are predicted via vector regression.
- Image segmentation—Given a picture, draw a pixel-level mask on a specific object.

3.2.2 Unsupervised learning

This branch of machine learning consists of finding interesting transformations of the input data without the help of any targets, for the purposes of data visualization, data compression, or data denoising, or to better understand the correlations present in the data at hand. Unsupervised learning is the bread and butter of data analytics, and it’s often a necessary step in better understanding a dataset before attempting to solve a supervised-learning problem. Dimensionality reduction and clustering are well-known categories of unsupervised learning[13].

3.2.3 Self-supervised learning

This is a specific instance of supervised learning, but it's different enough that it deserves its own category. Self-supervised learning is supervised learning without human-annotated labels—you can think of it as supervised learning without any humans in the loop. There are still labels involved (because the learning has to be supervised by something), but they're generated from the input data, typically using a heuristic algorithm.

For instance, autoencoders are a well-known instance of self-supervised learning, where the generated targets are the input, unmodified. In the same way, trying to predict the next frame in a video, given past frames, or the next word in a text, given previous words, are instances of self-supervised learning (temporally supervised learning, in this case: supervision comes from future input data). Note that the distinction between supervised, self-supervised, and unsupervised learning can be blurry sometimes—these categories are more of a continuum without solid borders. Self-supervised learning can be reinterpreted as either supervised or unsupervised learning, depending on whether you pay attention to the learning mechanism or to the context of its application[13].

3.2.4 Reinforcement learning

Long overlooked, this branch of machine learning recently started to get a lot of attention after Google DeepMind successfully applied it to learning to play Atari games (and, later, learning to play Go at the highest level). In reinforcement learning, an agent receives information about its environment and learns to choose actions that will maximize some reward. For instance, a neural network that "looks" at a video-game screen and outputs game actions in order to maximize its score can be trained via reinforcement learning.

Currently, reinforcement learning is mostly a research area and hasn't yet had significant practical successes beyond games. In time, however, it is expected reinforcement learning to take over an increasingly large range of real-world applications: self-driving cars, robotics, resource management, education, and so on. It's an idea whose time has come, or will come soon[13].

3.3 Deep learning for Computer Vision

This section introduces convolutional neural networks, also known as convnets, a type of deep-learning model almost universally used in computer vision applications[13].

3.3.1 The convolution operation

The fundamental difference between a densely connected layer and a convolution layer is this: Dense layers learn global patterns in their input feature space, whereas convolution layers learn local patterns (see figure 3.5): in the case of images, patterns found in small 2D windows of the inputs[13].

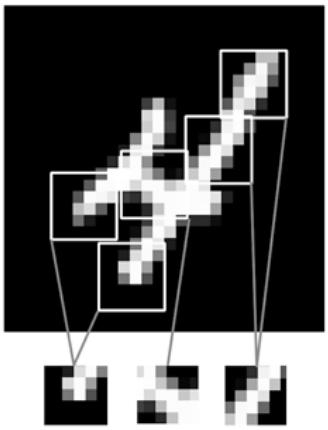

Figure 3.5 Images can be broken into local patterns such as edges, textures, and so on.

This key characteristic gives convnets two interesting properties:

- The patterns they learn are translation invariant. After learning a certain pattern in the lower-right corner of a picture, a convnet can recognize it anywhere: for example, in the upper-left corner. A densely connected network would have to learn the pattern a new if it appeared at a new location. This makes convnets data efficient when processing images (because the visual world is fundamentally translation invariant): they need fewer training samples to learn representations that have generalization power.
- They can learn spatial hierarchies of patterns (see figure 3.6). A first convolution layer will learn small local patterns such as edges, a second convolution layer will learn larger patterns made of the features of the first layers, and so on. This allows convnets to efficiently learn increasingly complex and abstract visual concepts (because the visual world is fundamentally spatially hierarchical).

Convolutions operate over 3D tensors, called feature maps, with two spatial axes (height and width) as well as a depth axis (also called the channels axis). For an RGB image, the dimension of the depth axis is 3, because the image has three color channels: red, green, and blue. For a black-and-white picture, the depth is 1 (levels of gray). The convolution operation extracts patches from its input feature map and applies the same transformation to all of these patches, producing an output feature map. This output feature map is still a 3D tensor: it has a width and a height. Its depth can be arbitrary, because the output depth is a parameter of the layer, and the different

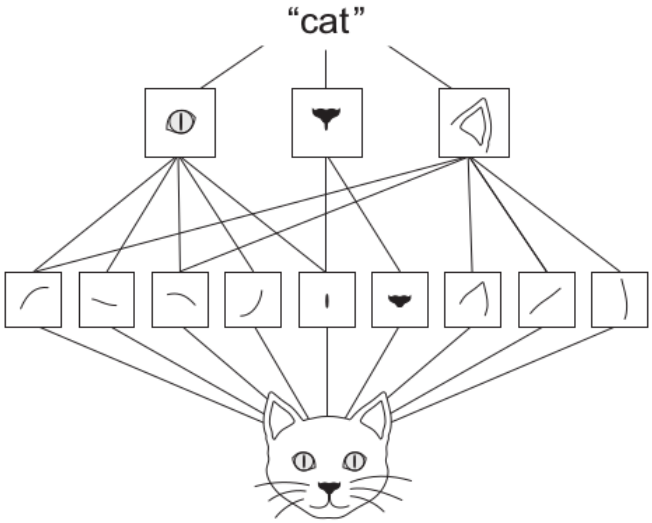

Figure 3.6 The visual world forms a spatial hierarchy of visual modules: hyperlocal edges combine into local objects such as eyes or ears, which combine into high-level concepts such as “cat.”

channels in that depth axis no longer stand for specific colors as in RGB input; rather, they stand for filters. Filters encode specific aspects of the input data: at a high level, a single filter could encode the concept “presence of a face in the input,” for instance [13].

Convolutions are defined by two key parameters:

- Size of the patches extracted from the inputs—These are typically 3 × 3 or 5 × 5 or 7 × 7
- Depth of the output feature map—The number of filters computed by the convolution. Common depth values are power of 2, for example 32, 64, 128, 256 and so on.

A convolution works by sliding these windows of size 3 × 3 or 5 × 5 over the 3D input feature map, stopping at every possible location, and extracting the 3D patch of surrounding features (shape (windowheight, windowwidth, inputdepth). Each such 3D patch is then transformed (via a tensor product with the same learned weight matrix, called the convolution kernel) into a 1D vector of shape (outputdepth,). All of these vectors are then spatially reassembled into a 3D output map of shape (height, width, outputdepth). Every spatial location in the output feature map corresponds to the same location in the input feature map (for example, the lower-right corner of the output contains information about the lower-right corner of the input)[13]. For instance, with 3 × 3 windows, the vector output[i, j, :] comes from the 3D patch input[i-1:i+1, j-1:j+1, :] . The full process is detailed in figure 3.7.

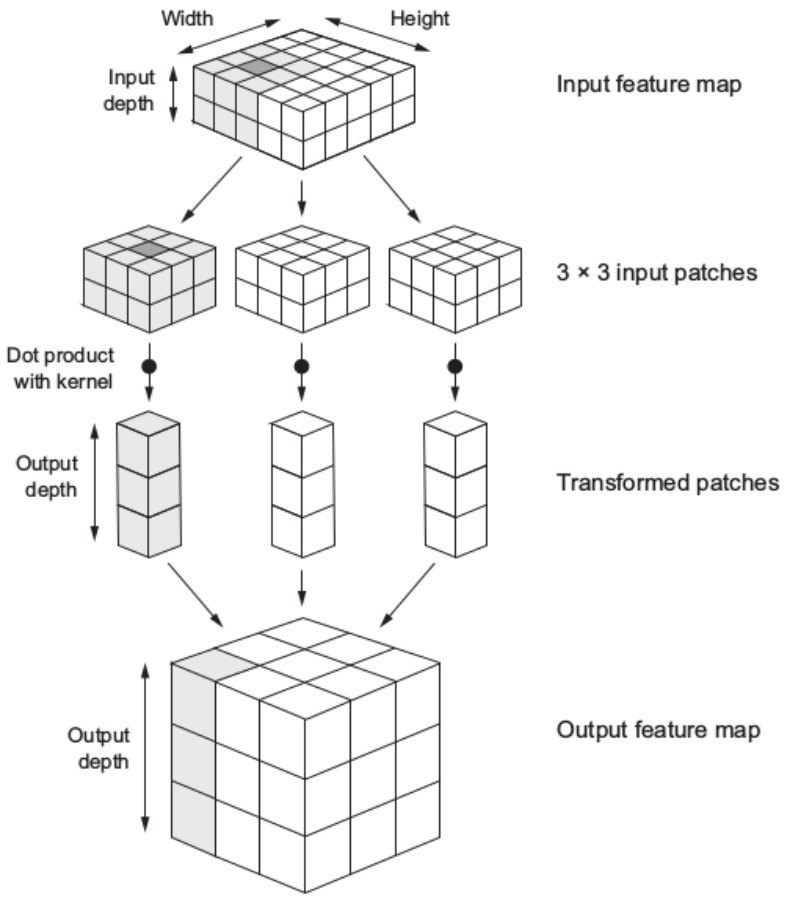

Figure 3.7 How convolution works

Note that the output width and height may differ from the input width and height. They may differ for two reasons:

- Border effects, which can be countered by padding the input feature map
- The use of strides.

3.3.1 Training a convnet

Having to train an image-classification, object-detection model using very little data is a common situation, which will likely encounter in practice when do computer vision in a professional context. A "few" samples can mean anywhere from a few hundred to a few tens of thousands of images[13]. In our case we had available almost 3.000 samples(images) well annotated from the COCO dataset[8]. A basic strategic to tackle this problem is to use data augmentation, which is a powerful technique for mitigating overfitting in computer vision. Although by alone this technique can't solve overfitting perfectly[13]. For this reason three more essential techniques for applying deep learning to small datasets can be used. They are feature extraction with a pretrained network, transfer-learning which we used and fine-tune a

pretrained network. These three techniques will be presented bellow briefly in order to obtain a more solid intuition about them and to clarify our selection of using a pretrained network.

There is the prevailing view that deep learning only works when lots of data is available. This is valid in part: one fundamental characteristic of deep learning is that it can find interesting features in the training data on its own, without any need for manual feature engineering, and this can only be achieved when lots of training examples are available. This is especially true for problems where the input samples are very high-dimensional, like images [13].

But what constitutes lots of samples is relative—relative to the size and depth of the network that is going to be trained, for start. It isn't possible to train a convnet to solve a complex problem with just a few tens of samples, but a few hundred can potentially suffice if the model is small and well regularized and the task is simple. Because convnets learn local, translation-invariant features, they're highly data efficient on perceptual problems. Training a convnet from scratch on a very small image dataset will still yield reasonable results despite a relative lack of data, without the need for any custom feature engineering [13].

What's more, deep-learning models are by nature highly repurposable: one can take, say, an image-classification or speech-to-text model trained on a large-scale dataset and reuse it on a significantly different problem with only minor changes. Specifically, in the case of computer vision, many pretrained models (usually trained on the ImageNet and/or COCO dataset[8]) are now publicly available for download and can be used to bootstrap powerful vision models out of very little data. That's what we did by using two different implementations, namely YOLOv3 and tiny-YOLOv3, which they will be presented in the next section.

3.3.2 Using a pretrained convnet

A common and highly effective approach to deep learning on small image datasets is to use a pretrained network. A pretrained network is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification, object-detection task. If this original dataset is large enough and general enough, then the spatial hierarchy of features learned by the pretrained network can effectively act as a generic model of the visual world, and hence its features can prove useful for many different computer-vision problems, even though these new problems may involve completely different classes than those of the original task. For instance, one might train a network on ImageNet (where classes are mostly animals and everyday objects) and then repurpose this trained network for something as remote as identifying furniture items in images. Such portability of learned features across different problems is a key advantage of deep learning compared to many older, shallow-learning approaches, and it makes deep learning very effective for small-data problems[13]. We took advantage of this technique by exploiting the pretrained YOLOv3 networks on COCO-object detection Dataset[8], which comprises of 80 classes of everyday objects, like cars, boats, persons and so on. This technique is thoroughly described bellow in transfer learning section.

3.3.3 Feature extraction

Feature extraction consists of using the representations learned by a previous network to extract interesting features from new samples. These features are then run through a new classifier, which is trained from scratch.

As it is known, convnets used for image classification – object detection comprise two parts: they start with a series of pooling and convolution layers, and they end with a densely connected classifier. The first part is called the convolutional base of the model. In the case of convnets, feature extraction consists of taking the convolutional base of a previously trained network, running the new data through it, and training a new classifier on top of the output (see figure 3.8).

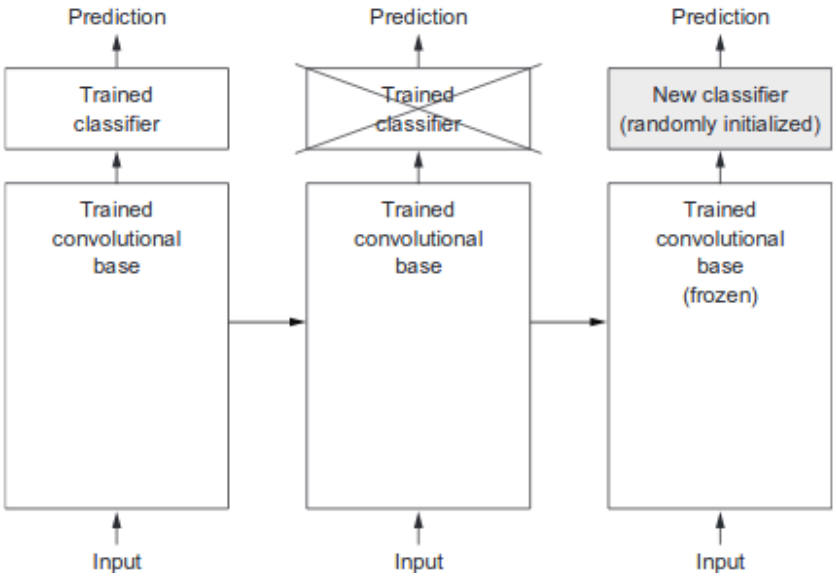

Figure 3.8 Swapping classifiers while keeping the same convolutional base

Why only reuse the convolutional base? Could also be reused the densely connected classifier as well? In general, doing so should be avoided. The reason is that the representations learned by the convolutional base are likely to be more generic and therefore more reusable: the feature maps of a convnet are presence maps of generic concepts over a picture, which is likely to be useful regardless of the computer-vision problem at hand. But the representations learned by the classifier will necessarily be specific to the set of classes on which the model was trained—they will only contain information about the presence probability of this or that class in the entire picture. Additionally, representations found in densely connected layers no longer contain any information about where objects are located in the input image: these layers get rid of the notion of space, whereas the object location is still described by convolutional feature maps. For problems where object location matters, densely connected features are largely useless.

Note that the level of generality (and therefore reusability) of the representations extracted by specific convolution layers depends on the depth of the layer in the model. Layers that come earlier in the model extract local, highly generic feature maps (such as visual edges, colors, and textures), whereas layers that are higher up extract more-abstract concepts (such as “cat ear”

or “dog eye”). So if the new dataset differs a lot from the dataset on which the original model was trained, it may be better off using only the first few layers of the model to do feature extraction, rather than using the entire convolutional base.

3.3.4 Fine-tuning

Another widely used technique for model reuse, complementary to feature extraction, is fine-tuning. Fine-tuning consists of unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the newly added part of the model (in this case, the fully connected classifier) and these top layers. This is called fine-tuning because it slightly adjusts the more abstract representations of the model being reused, in order to make them more relevant for the problem at hand[13].

3.3.5 Transfer Learning

The idea of transfer learning is inspired by the fact that people can intelligently apply knowledge learned previously to solve new problems. For example, learning to play one instrument can facilitate faster learning of another instrument. Transfer learning has gained attention since its discussion in the Neural Information Processing Systems 1995 workshop on Learning to Learn[35], which focused on the need for lifelong machine learning methods that retain and reuse previously learned knowledge. Another good analogy is with traditional software development: We almost never write a program completely from scratch; every application makes heavy use of code libraries that take care of common functionality. Maximizing code reuse is a best practice for software development, and transfer learning is essentially the machine learning equivalent[4].

Transfer learning is an artificial intelligence (AI) practice that uses data, deep learning recipes, and models developed for one task, and reapplies them to a different, but similar, task. In other words, it's a method in machine learning where a model developed for one task is used as a starting point for a model in a second task. Reuse of pretrained models allows improved performance when modeling the second task, hence achieving results faster.

Vast quantities of readily available data are great, but it isn't a prerequisite for success. With modern machine learning and deep learning techniques, knowledge acquired by a machine working on one task can be transferred to a new task if the two are somewhat related. This eventually helps to reduce training time significantly, thus improving productivity. In figure 3.9 the differences between traditional machine learning and with the use of transfer learning are presented.

There are different types of transfer learning which can be summarize into three categories as shown in the following table (3.1). The scope of this section is not to prevent deeply the different approaches of transfer learning but to state our selection of inductive transfer learning. The reason is that after careful consideration of different regression-object detection models, the YoLoV3 had the better response. We pause the explanation of YoLo V3 model till the next section. By looking to our feature space it was obvious that the fastest approach for Sea-Vessels

detection was to retrain the YoLo V3 to our dataset, exploiting the learning of the different features that learned previously by the network in a much bigger and diverse dataset. That gave also the ability to have better detection results in Sea-Vessels Detection.

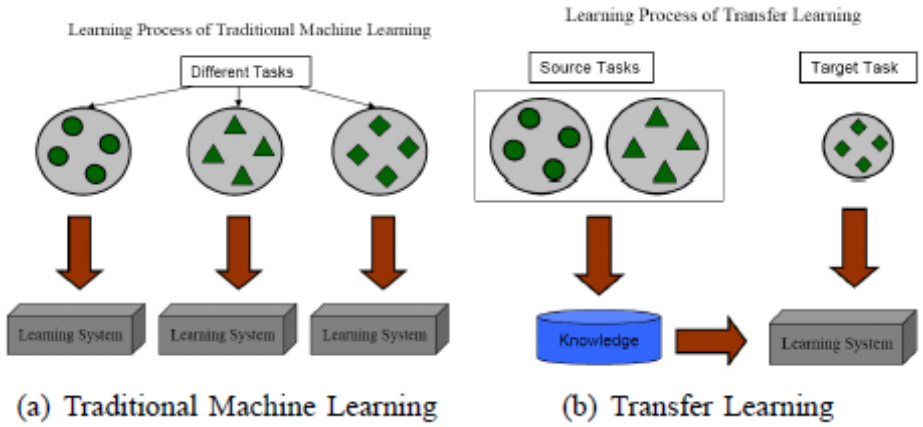

Figure 3.9 Different learning processes between traditional machine learning and transfer learning.

3.4 Customizing YOLOv3 for Sea-Vessels Detection

Before one can actually estimate 3D position of multiple objects from a single image, it is necessary first to be able to detect these objects of interest given a single image. As mention above we exploited the performance and accuracy of Yolo V3 object detector, customizing it although for our new task of Vessels detection. This section describes the Yolo V3 model architecture and continues with the steps that were performed before the retraining of the model till the first use of the final model and the first inference.

3.4.1 YOLO Model Architecture

The first YOLO model architecture in 2016 introduced a new approach to object detection[21]. Prior work on object detection repurposes classifiers to perform detection. Instead, Yolo model frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance. This gave Yolo model an extremely small processing time from end-to-end detection and outstanding performance compared with the model that where state-of-the art at that time[21].

What was genius about Yolo architecture was that it unified the separate components of object detection into a single neural network. Yolo network uses features from the entire image to predict each bounding box. It also predicts all bounding boxes across all classes for an image

simultaneously. This means the network reasons globally about the full image and all the objects in the image[21].

The YOLO design enables end-to-end training and realtime speeds while maintaining high average precision. Yolo system divides the input image into an SxS grid. If the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts. Formally it is defined confidence as Pr(Object) * IOUtruthpred . If no object exists in that cell, the confidence scores should be zero. Otherwise, the confidence score to equal the intersection over union (IOU) between the predicted box and the ground truth[21].

Each bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x; y) coordinates represent the center of the box relative to the bounds of the grid cell. The width and height are predicted relative to the whole image. Finally the confidence prediction represents the IOU between the predicted box and any ground truth box. Each grid cell also predicts C conditional class probabilities, Pr(Classi|Object). These probabilities are conditioned on the grid cell containing an object.[21]

Yolo only predicts one set of class probabilities per grid cell, regardless of the number of boxes B. At test time the conditional class probabilities and the individual box confidence predictions are multiplied,

$$\Pr(\text{Classij}|\text{Object}) * \Pr(\text{Object}) * \text{IOUtruthpred} = \Pr(\text{Classi}) * \text{IOUtruthpred} \quad (1)$$

which gives the class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object.

The Yolo architecture presented in Figure 3.11 implement this model as a convolutional neural network and evaluate it on the PASCAL VOC detection dataset [25]. The initial convolutional layers of the network extract features from the image while the fully connected layers predict the output probabilities and coordinates. Yolo network architecture is inspired by the GoogLeNet model for image classification [32]. Yolo network has 24 convolutional layers followed by 2 fully connected layers as shown bellow.

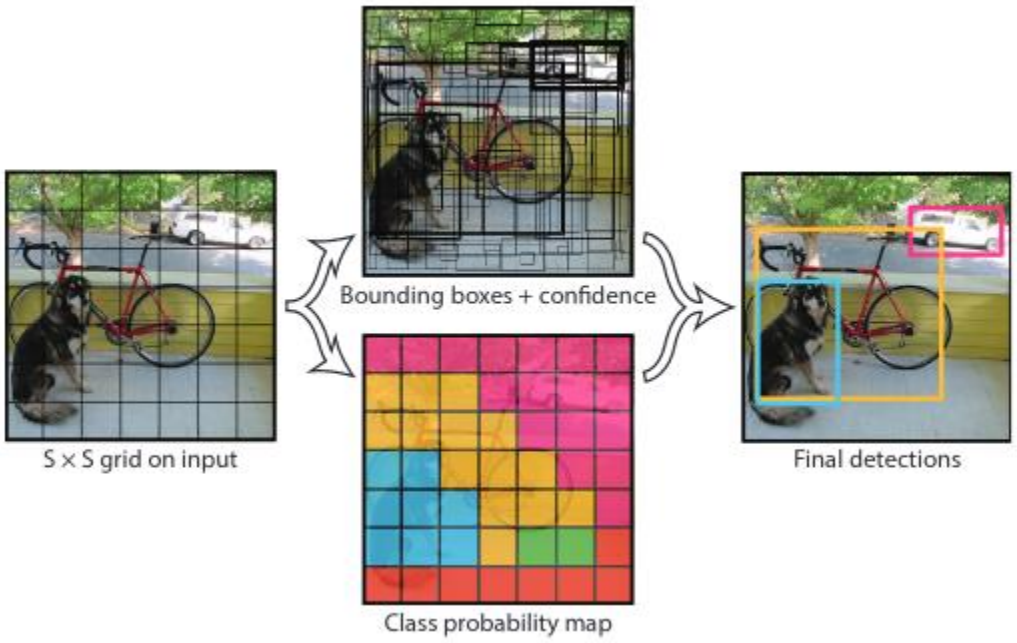

Figure 3.10 Yolo system models detection as a regression problem. It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B ∗5 + C) tensor.[21]

During training Yolo optimize the following, multi-part loss function:

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2
\end{aligned}
$$

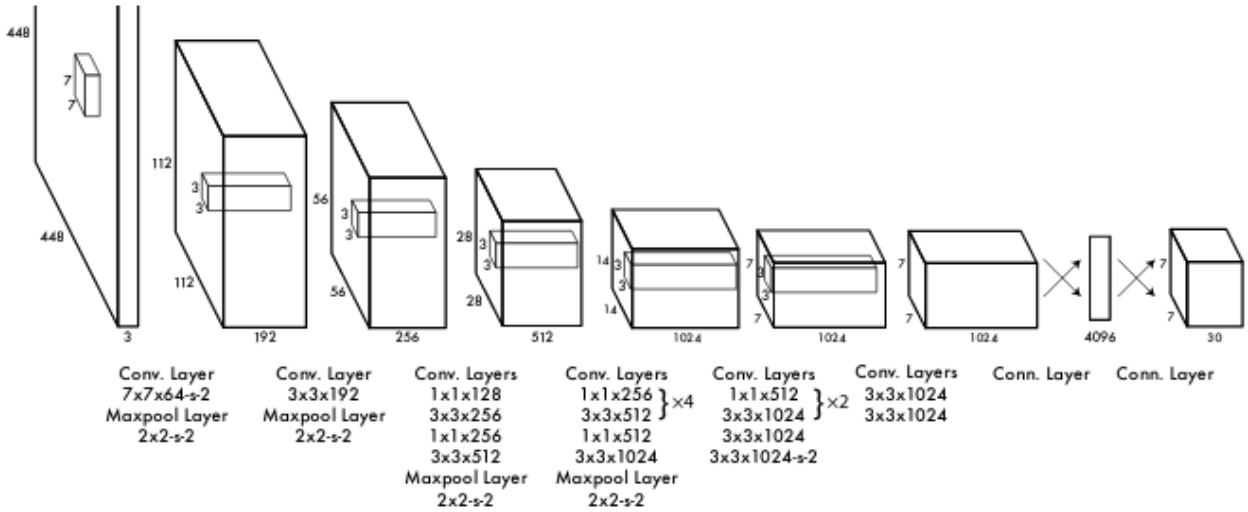

Figure 3.11 Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1 × 1 convolutional layers reduce the features space from preceding layers.[21]

3.4.2 Yolov3 Model Architecture

Based on the above described architecture authors of Yolo did some impressing changes in model architecture which gave it an outstanding performance and detection accuracy. First of all Yolov3[20] is a little bigger than before having more layers as is shown in figures 3.12 & 3.13.

	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 3.12 Yolo V3 feature extractor network architecture Darknet-53[20]

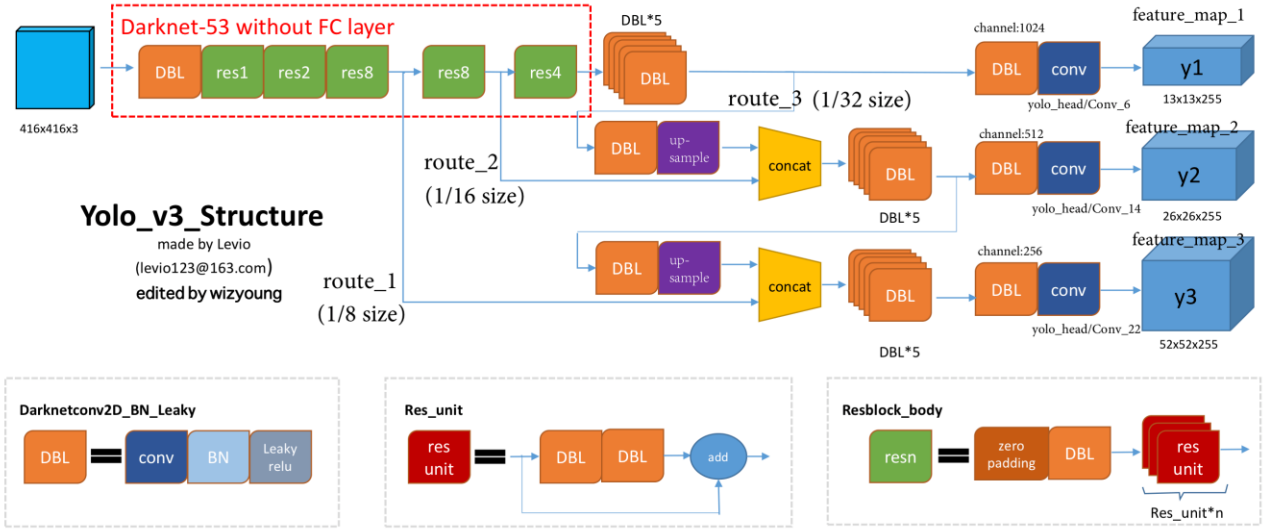

Figure 3.13 The completely Yolo v3 model architecture. After the feature extractor (Darknet-53) one can see that feature maps with different sizes are upsampled and used for prediction, yield three different prediction feature maps. This is called feature pyramid.

Furthermore the system predicts bounding boxes using dimension clusters as anchor boxes (figure 3.14). The network predicts 4 coordinates for each bounding box, tx ,ty , tw , th . If the cell is offset from the top left corner of the image by (cx , cy) and the bounding box prior has width and height pw, ph , then the predictions correspond to:

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
\qquad \text{(Eq. 1)}
$$

During training sum of squared error loss function is used.

YOLOv3 predicts an objectness score for each bounding box using logistic regression. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. If the bounding box prior is not the best but does overlap a ground truth object by more than some threshold the prediction is ingored, following [32]. Threshold of .5 is used. Unlike [32] Yolo system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness. Finally, each box predicts the classes the bounding box may contain using multilabel classification[20].

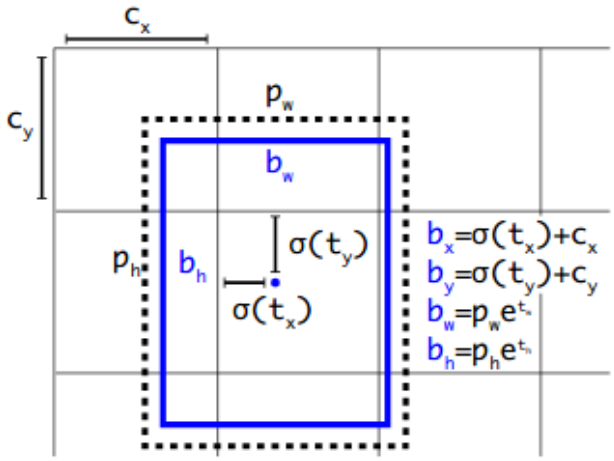

Figure 3.14 Bounding boxes with dimension priors and location prediction. We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [20]

A key role characteristic of Yolov3 model architecture is that it makes predictions across three differen scales. Meaning that features are extracted from those scales using a similar concept to feature pyramid networks [38]. From the base feature extractor of Yolov3 model, several convolutional layers were added. The last of these predicts a 3-d tensor encoding bounding box, objectness, and class predictions. Yolov3 originally created for COCO [8], predicts 3 boxes at each scale so the tensor is $N \times N \times [3 * (4 + 1 + 1)]$ for the 4 bounding box offsets, 1 objectness prediction, and 1 class prediction. The real YoloV3 model calculates 80 class predictions, meaning that the output tensor is $N \times N \times [3 * (4 + 1 + 80)]$ for the COCO.

Next the feature map from 2 layers previous is taken and is upsampled it by 2×. Also a feature map from earlier in the network is taken and is merged with the upsampled features using concatenation (Figure 3.12). This method allows Yolov3 model to get more meaningful semantic information from the upsampled features and finer-grained information from the earlier feature map. Then a few more convolutional layers to process this combined feature map are added, and eventually a similar tensor is predicted, although now twice the size. The same design is performed one more time to predict boxes for the final scale. Thus the predictions for the 3rd scale benefit from all the prior computation as well as finegrained features from early on in the network[20].

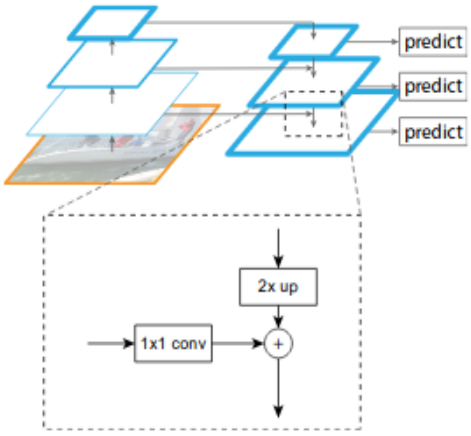

Figure 3.12 Yolov3 structure but leverages it as a feature pyramid, with predictions made independently at all levels

3.4.3 Multiple Sea-Vessels Detection

It is time now to present the steps that were executed in order to adjust necessarily the Yolov3 model architecture to our task of Sea-Vessel detection. Specifically we modified and retrained the Yolov3 as well as the Yolov3_tiny model, which is a smaller version of Yolov3, exploiting the learned features that these networks learned previously.

For our task we used the dataset from COCO object detection competition[8], which comprises of more than 150 thousands of well annotated images from 80 different classes. We kept the images that contained only Sea-Vessels, yielding more than 3.000 images. In order to achieve sufficient detection accuracy we had to change the anchor boxes that the original network used.

In order to determine the anchor boxes of the Yolov3 models based on the new task of Sea-Vessels detection, we implemented a k-means clustering system[23]. For Yolov3 we used 9 clusters and for Yolov3-tiny we used 6 clusters. The k-means algorithm computed the new centers of the anchor boxes based on the our dataset (see figure 3.13 & 3.14). The K-mean clustering system was implemented in python.

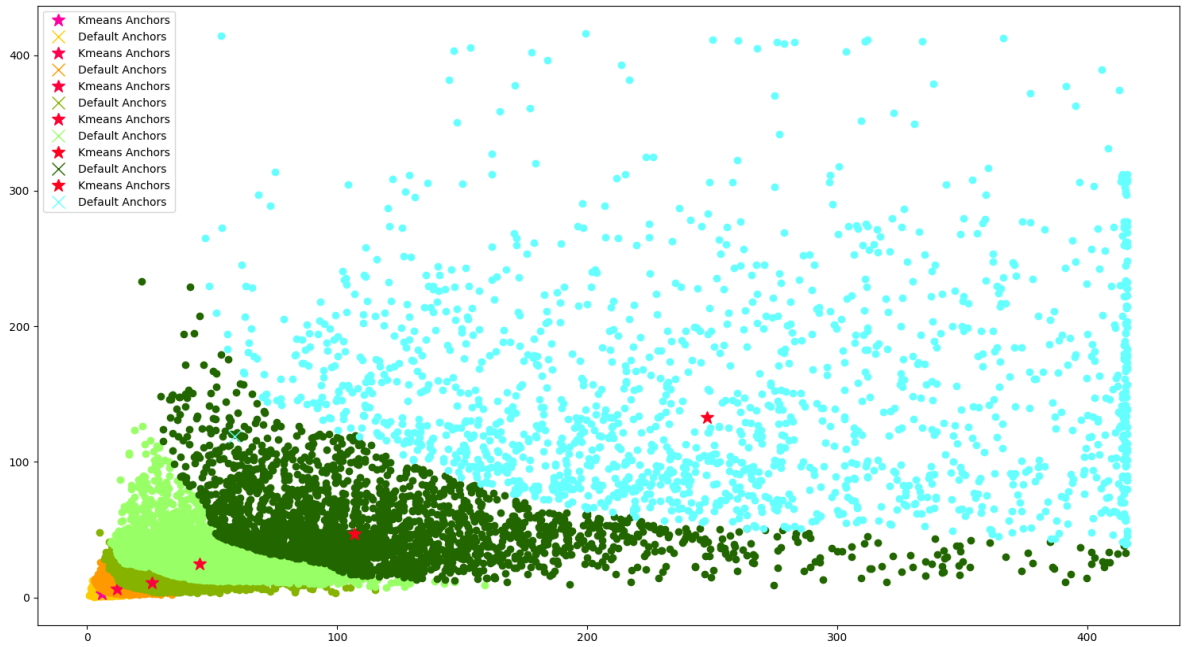

Figure 3.13 The different clusters and the centroid of each cluster is depicted for Yolov3-tiny. With red star the centroid of each cluster and with colored dots, the corresponding anchor boxes that belongs to this cluster are presented.

Having the new anchor boxes determined it is time to retrain our models. We used the framework from the authors of Yolo, which is Darknet. Every training needed a whole week in a medium-sized computer running on a Nvidia 1060 6Gb graphic card. We trained every network for more than 40.000 epochs. The resulting average loss function is in figures 3.15 & 3.16 depicted. From the figures one can see that after 5000 epoch model is starting to learn the new feature.

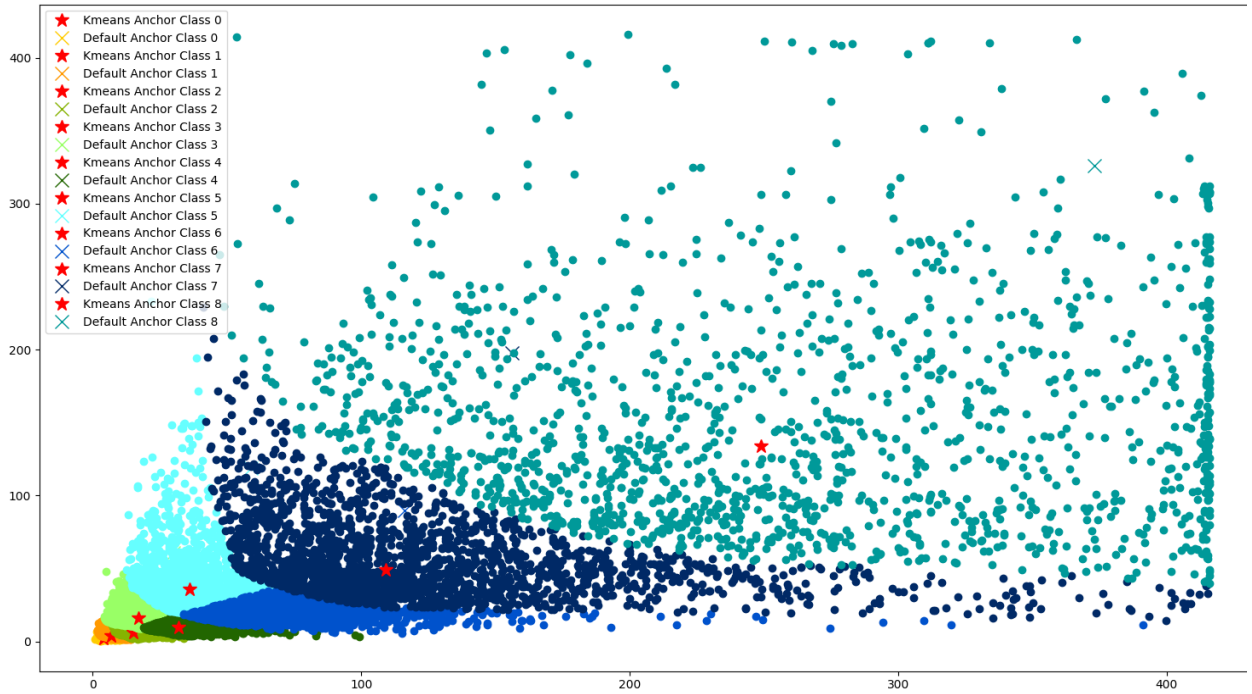

Figure 3.14 The different clusters and the centroid of each cluster is depicted for Yolov3. With red star the centroid of each cluster and with colored dots, the corresponding anchor boxes that belongs to this cluster are presented.

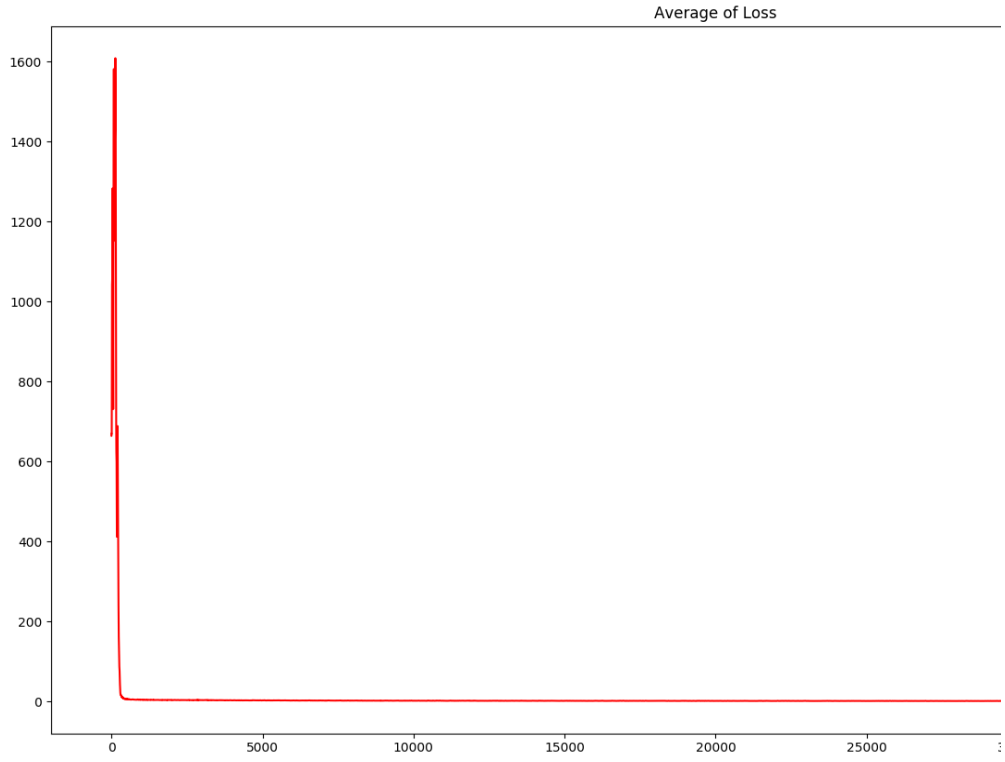

Figure 3.15 The average loss function is depicted

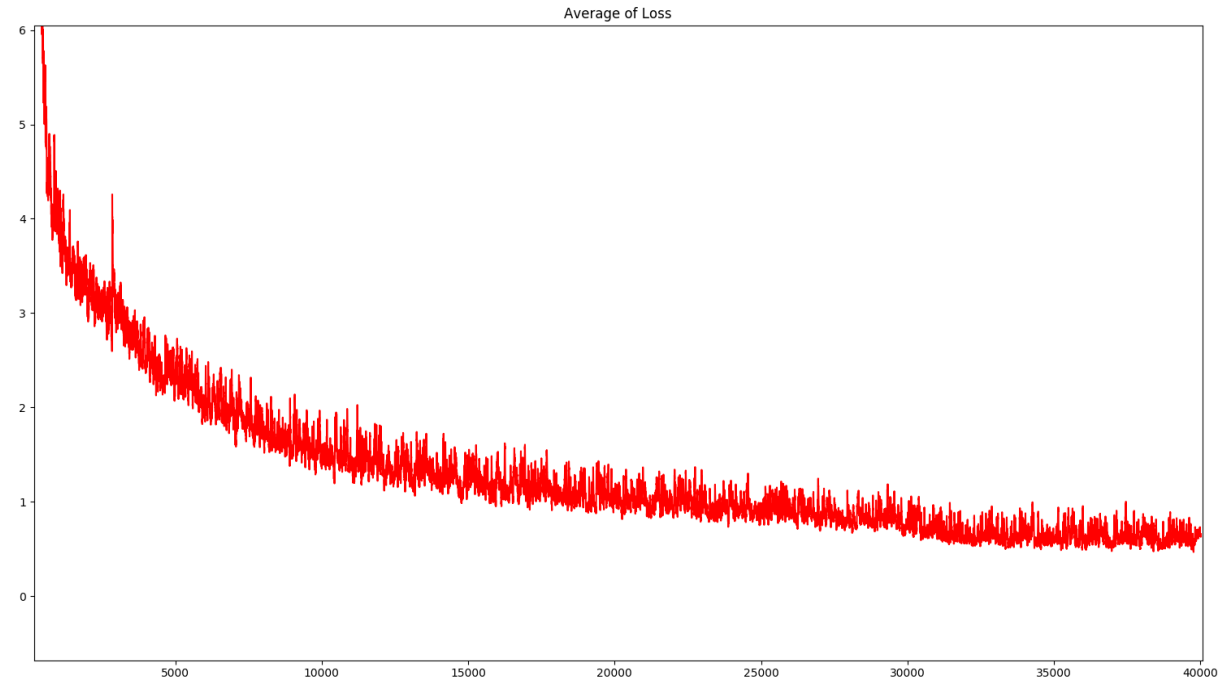

Figure 3.16 The average loss function is depicted. As is shown after 5000 epoch the loss function starts to converge to a steady value near 1.

For retraining the models a variable learning rate was used. This training strategy is well known to machine learning community and tries to tackle the fact that at the begging of training big loss values will be encountered causing to big weight changes of the pretrained network. This causes the network to forget the learned features that are previously learned, a phenomenon which is not desired. For this reason at the begging of the training a learning rate starting almost from 0 till 0.001 progressively, is used for 1000 epochs. This is called burn-in or warm-up period in deep learning community (figure 3.17).

In the following table the performances of the two different implementations namely YoloV3 and YoloV3_tiny are presented

Model	**Precision**	**Recall**	**Average Precision**
Yolo V3 @ 608	0.982	0.948	0.838
Yolo V3 @ 416	0.975	0.917	0.824
Yolo V3 tiny @ 416	0.979	0.786	0.763

Table 5.1 Performance of Yolo V3 different implementations at Sea-Vessels detection

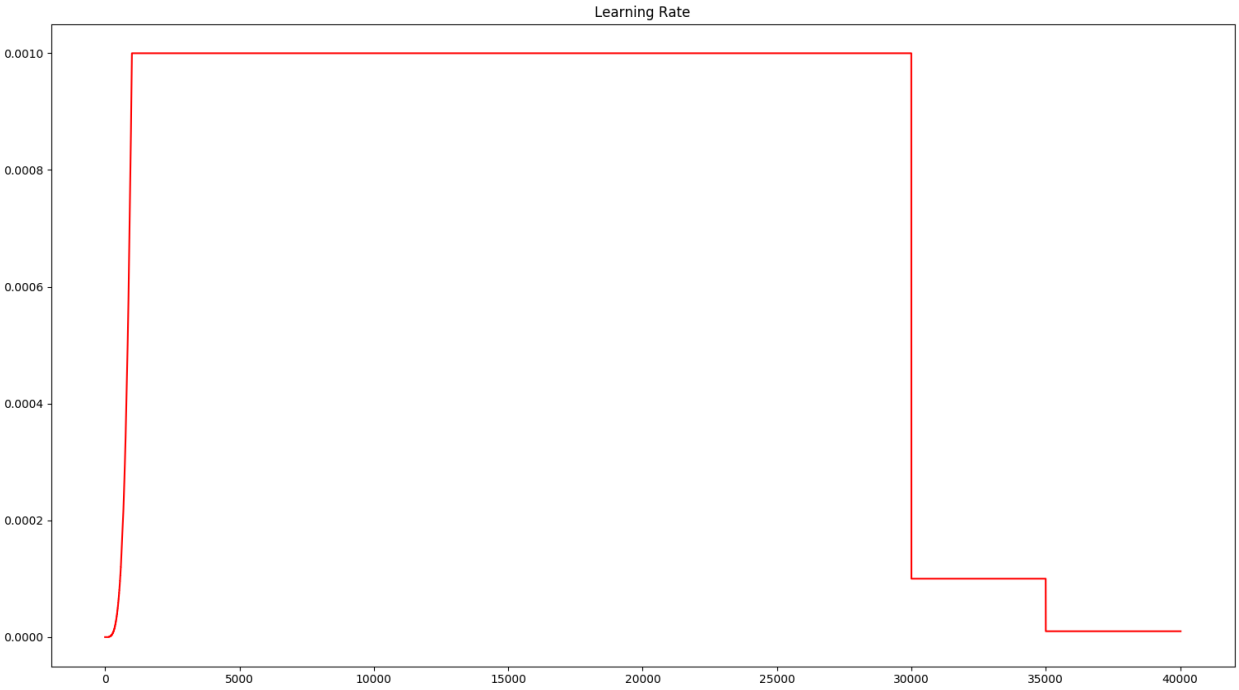

Figure 3.17 Variable learning rate training.

The resulting networks performed extremely well on detecting Sea-Vessels in different weather and lighting conditions. Exemplary images are shown bellow in different lighting conditions , proving the robustness of the networks.

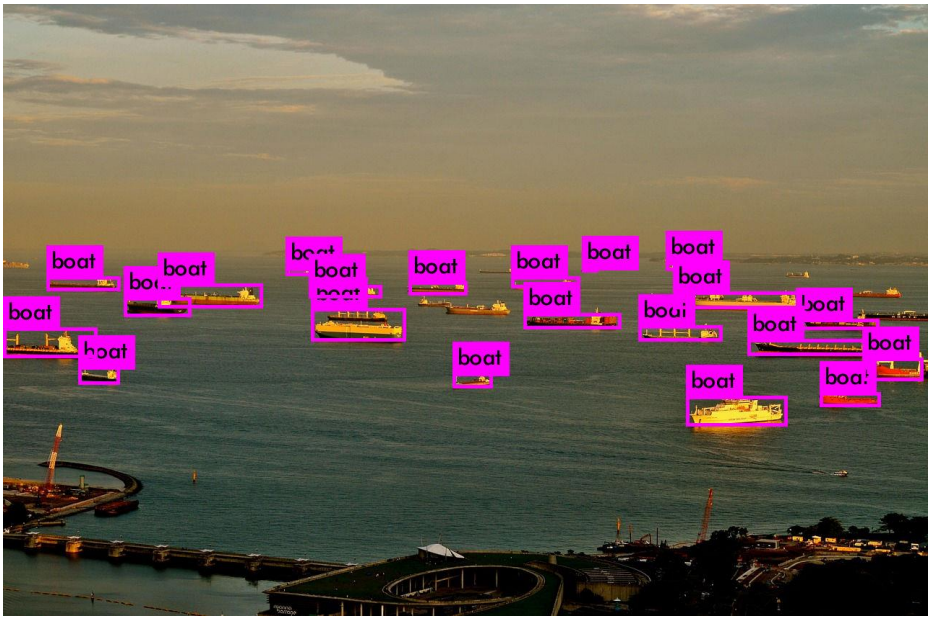

Figure 3.18 a) An image of a busy port and the detected Sea-Vessels within the oat annotated boxes

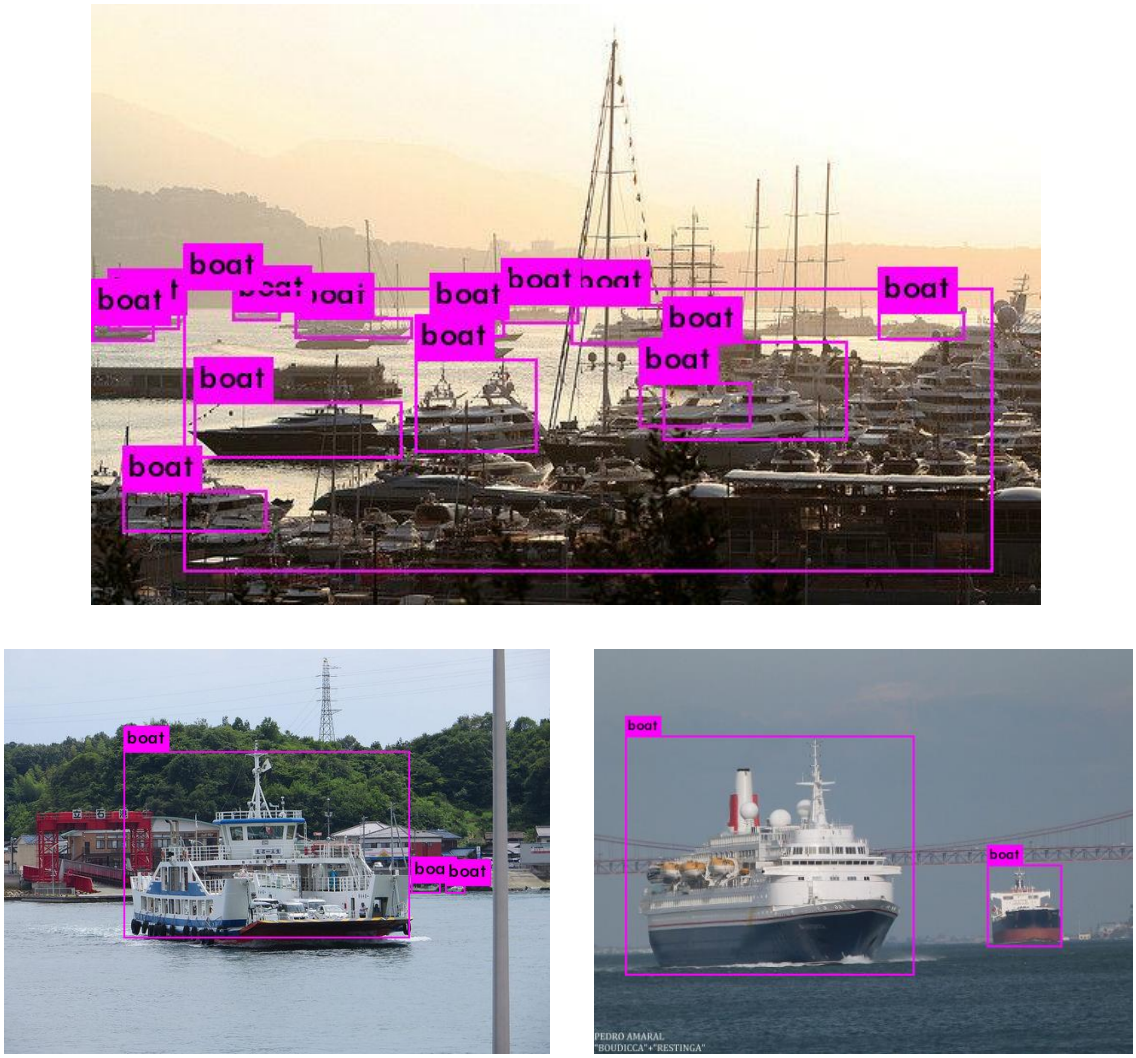

Figure 3.18 b) Images of detected Sea-Vessels with different light conditions.

Chapter 4

4 Fast Horizon Line Detection Algorithm

This chapter present the fast horizon line detection algorithm exploiting computer vision techniques . First the theory of the computer vision tools that were used is presented. Then the fast horizon line detection algorithm is further studied. Finally the performance of the proposed detection algorithm is thoroughly investigating.

4.1 The need for Horizon Line detection

As explained in the previous section, with the help of Sea-Vessel object detector, one can find multiple Sea-Vessels of interest in a single image. The question is, due to determined object detection precision, how do we decide for a bad detection. One simple but yet powerful technique is to use the horizon line of the open sea to discard false Sea-Vessel detections.

As is shown by the general diagram pipeline (Fig. 4.1) from images to 3D Sea-Vessel detections , first we take an image frame then we detect bounding boxes containing Sea-Vessels. We exploit that information in our horizon line detector in order to form a ROI (Region Of Interest) in the image frame, where the horizon detection would take place and then we discard false positives. In order to compute the ROI we took the information of the highest and lowest detected Sea-Vessel in the image and after applying a threshold we cutted the image and the ROI was directly formed.

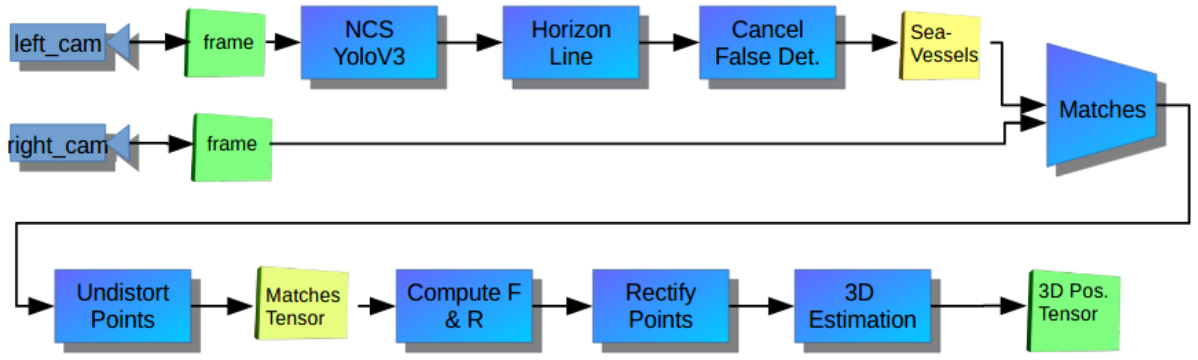

Figure 4.1 General Diagram of Sea-Vessels detection and 3D position estimation algorithm pipeline.

One key characteristic between Sea and Sky surface is that these areas have different color and light intensity distribution which we exploit it in order to detect the horizon line. We make the assumption that the horizon line is generally represented as a straight line in a maritime scenario. Based on this assumption we employ a series of computer vision techniques but combined with such a way that the overall time execution is not affecting the overall system performance.

4.2 Fast Horizon Line Detector

As described previously we begin by cutting the initial image and form a ROI. After that we use a feature detector in order to detect the horizon line (Figure 4.2). As a feature detector we used the simple but yet powerful Canny edge detector. It was developed by John F. Canny in 1986[19]. It is a multi-stage algorithm. The first step is to filter the image with a smoothing filter like Gaussian and then apply Canny edge detector.

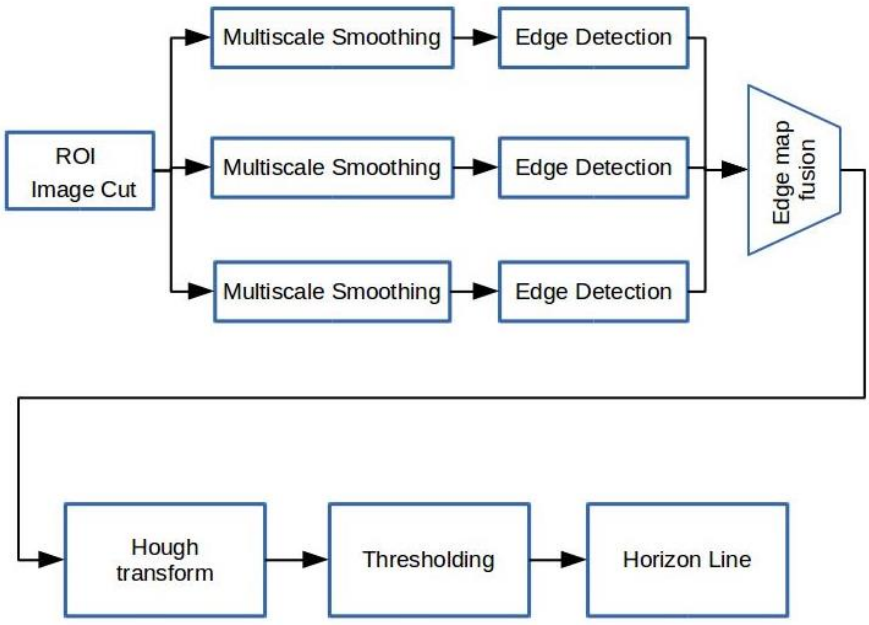

Figure 4.2 Horizon line detection pipeline. The steps for detecting the horizon line are depicted in this diagram

4.2.1 Edge Detection

After that Canny edge detector tries to find intensity gradient of the input image. This is achieved by applying Sobel kernel filter in both horizontal and vertical direction to get first

derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

$$EdgeGradient(G) = \sqrt{G_x^2 + G_y^2}$$
$$Angle(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \quad \text{(Eq. 1)}$$

Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions. After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient, as shown in the figure 4.3 bellow.

Point A is on the edge (in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for the next stage, otherwise, it is suppressed (putted to zero).

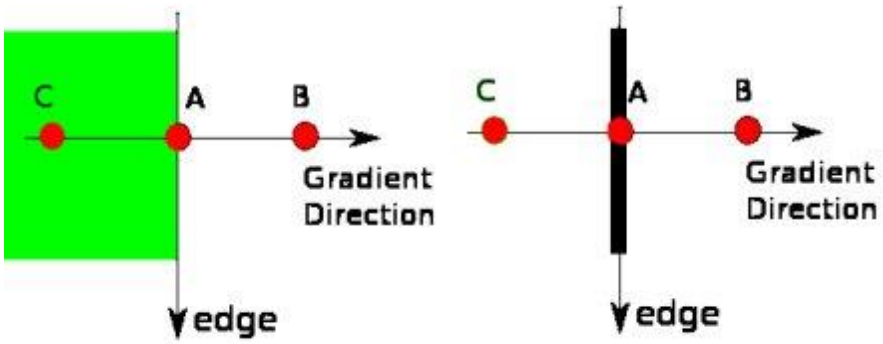

Figure 4.3 Sobel Non-maximum Suppression method. Left the point A is not local maximum and it is not considered as an edge point. In the right image although point A is local maximum and that's why it is consider as an edge point.

The next stage decides which edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels, they are considered to be part of edges. Otherwise, they are also discarded. See the figure 4.4 below:

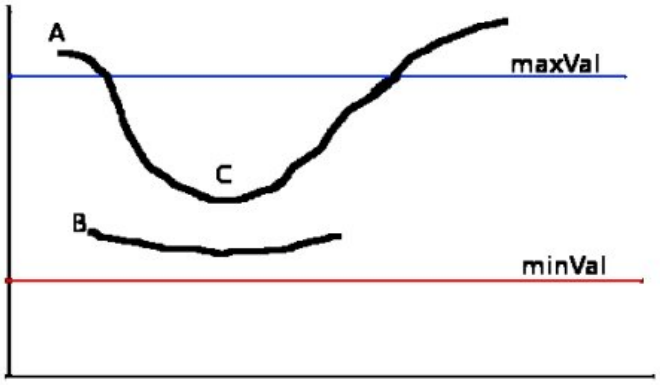

Figure 4.4 Edge decision threshold.

The edge A is above the maxVal, so considered as “sure-edge”. Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any “sure-edge”, so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result. This stage also removes small pixels noises on the assumption that edges are long lines. So what we finally get is strong edges in the image.

4.2.2 Multi-Scale edge detection

Maritime scenes contain many edges generated by wakes from ships, sunlight, and waves. This makes the detection of horizon line ambiguous, because of too many generated edges. In order to tackle this problem, we used multi-scale edge detection. The scale in edge detection is related to the size of the smoothing filter applied before edge detection[6].

Large-scale edge detection can identify reliable edges related to the horizon, but it loses detailed structures. In addition, recent research pointed out that detecting edges at multiple scales can reduce the inherent ambiguity of edge detection at a single scale. Therefore, several methods, [24] adopting a multi-scale approach, have been proposed to mitigate sensitivity to parameters of edge detection and to reduce the effect of noisy edges[6].

Multi-scale edge detection is distinguishable by the method of analyzing the information detected at different scales.[24] Previous works have independently processed edge information detected on different scales, so that horizon line estimation is applied to the number of scales. This was one reason for increasing the processing time of the horizon detection method, when adopting multi-scale edge detection. In addition, MusCoWERT [30] detected edges on different scales and analyzed their length, because it suffers from processing times requiring an order of tens of seconds.

This Paper [6] proposed a method which used three different scales of smoothing filter and then merging the resulting edges based on a threshold intensity value. In our implementation, the proposed method applies different smoothing values in ROI but for every scale applies the Canny edge detector with different threshold values, giving more importance to scales with bigger smoothing filters. This gave us the ability to isolate more accurately the dominant lines.

The proposed method detects edges from the images by applying a smoothing filter of various sizes. Then, it applies the Canny edge detector with different low and high threshold values of edges, based on intensity distribution of the ROI. More specifically before we compute the edges in each scale, we compute the median intensity value of the cropped image (ROI). Then we compute the threshold values for the edge detector using the median value of the ROI as a central value. Using the type

lower = int(max(0, (1.0 - sigma) * v))

upper = int(min(255, (1.0 - sigma) * v))

Where the sigma parameter is consider as normal deviation. Then the edge images at different scales are synthesized to a single edge map.

Detecting a horizon by analyzing a combined edge map can reduce the inherent ambiguity of edge detection using a single scale, while increasing the processing speed of horizon detection. The proposed method uses the Gaussian filter as a smoothing filter. The Canny edge detector is applied to multi-scale images independently. Then, the weighted edge map is synthesized using the edge maps to which the Canny edge detector is applied, as follows

$$W(x, y) = \sum_{s=1}^{N} E_s(x, y) \qquad \text{(Eq. 2)}$$

where N is the number of median filters, of the scale s, and Es is the edge maps of the scale s.

The edges related to the horizon were consistently detected on edge maps at various scales because of the strong brightness changes near the horizon. Thus, the proposed method applies thresholding to the weighted edge map to suppress noisy edges, while keeping the edges associated with the horizon. The thresholding to the weight edge maps is applied as follows

$$W_T(x, y) = \begin{cases} 255 & \text{if } W(x, y) \geq t \\ 0 & \text{if } W(x, y) \leq t \end{cases} \qquad \text{(Eq. 3)}$$

where t is the threshold for filtering the noisy edges; we set the threshold to 170.

The example image of the edge maps generated from the multi-scale images and the weighted edge map applying the thresholding are shown in figure 4.5. To improve the readability of the edge maps in figure 4.5, a dilation filter with a 5 x 5 rectangular structuring element is first applied to the edge maps.

Figure 4.5 shows that the proposed multi-scale edge detection can preserve the edges associated with the horizon while suppressing the noisy edges. The proposed method reduces edges unrelated to the horizon, but there still exist outlier edges. Therefore, a method is necessary to reduce the effect of out-lier edges when estimating the horizon line.

Figure 4.5.1 The final edge-map after applying three different size smooth Gaussian filters to the original image bellow and thresholding.

Figure 4.5.2 Above the final edge-map after applying three different size smooth Gaussian filters to the original image bellow. Then using thresholding the three different edge maps are concatenated into one final.

4.2.3 Horizon Line Estimation

The representative method for estimating the horizon from the edge image are the Hough transform. The method using Hough transform can robustly estimate the parameter of horizon, even when there are small numbers of edges related to the horizon or there are many noisy edges. This is because Hough transforms tend to be most successfully applied to line finding.

A line is easily parameterized as a collection of points (x, y) such that

$$x \cos\theta + y \sin\theta + r = 0$$

Now any pair of (θ, r) represents a unique line, where $r \geq 0$ is the perpendicular distance from the line to the origin, and $0 \leq \theta < 2\pi$. We call the set of pairs (θ, r) line space; the space can be visualized as a half-infinite cylinder. There is a family of lines that passes through any point token. In particular, the lines that lie on the curve in line space given by $r = -x_0 \cos\theta + y_0 \sin\theta$ all pass through the point token at (x_0 , y_0).

Because the image has a known size, there is some R such that we are not interested in lines for $r > R$, these lines will be too far away from the origin for us to see them. This means that the lines we are interested in, form a bounded subset of the plane, and we discretise this with some convenient grid figure 4.6. The grid elements can be thought of as buckets, into which we will sort votes. This grid of buckets is referred to as the accumulator array. Now for each point token we add a vote to the total formed for every grid element on the curve corresponding to the point token. If there are many point tokens that are collinear, we expect that there will be many votes in the grid element corresponding to that line.

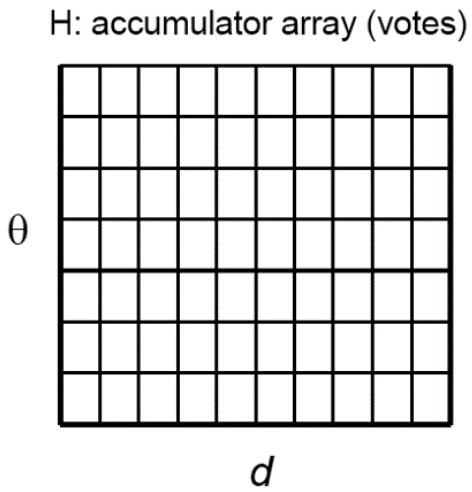

Figure 4.6 The accumulator array where the lines are voted for a every θ, d based on equation 1

After applying the Hough Transform in our multi-scale edge map, we get many candidate horizon lines. In order to decide the right one we discard lines that having a slope bigger than 45 degrees. This is not sufficient to cancel all the outliers and present only the dominant horizon line. In order to tackle this problem, we exploited the difference in color distribution and

intensity of the sea-sky. The horizon line lies on the boundary of these two spaces and we exploit those differences by computing the deviation of the intensity value of two boxes above and bellow the horizon line in each end point of the line (figures 4.7).

$$Criterion = \frac{\sum_{x\ y} \left(I(x\ \ y)_{down} - I_{up}^{mean} \right)^2}{N^2} \qquad \text{(Eq. 4)}$$

where N is the size of image window to be checked and in our application was 10pixels

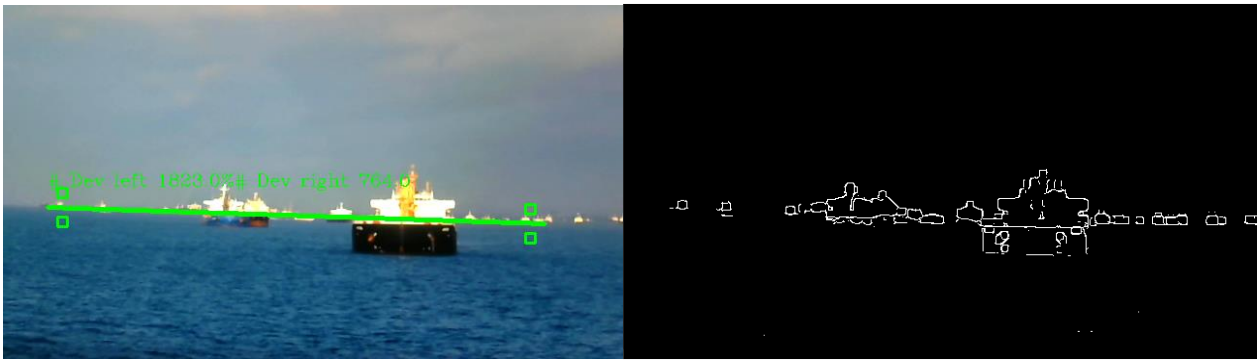

Figure 4.7.1 Images of horizon line detected using square windows to compare the two different areas of sky and see. It is also written the deviation values computed in each side of the line. In the right the computed finale-edge map from which the lines where computed is shown.

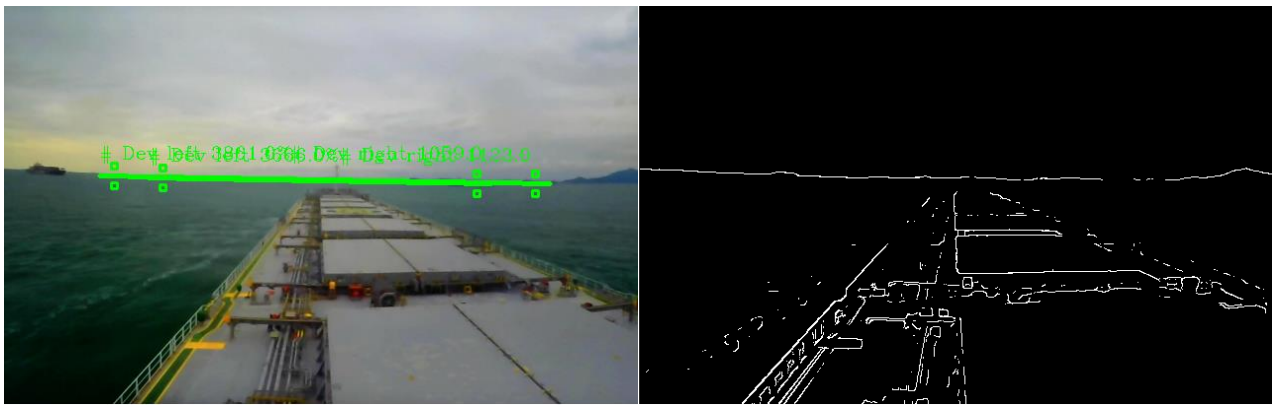

Figure 4.7.2 Horizon line detected left and the correspondent final edge-map right, before thresholding, are depicted. In left there are two lines as horizon line detected. This facilitates the need of thresholding the computing deviation values.

The threshold of the deviation values in each side of the horizon line should be adjusted properly in order to discard false detections as shown above (figure 4.7.2). Not only the deviation threshold values should be adjusted, but also the window size. Window size plays a crucial role in discarding false detections and also in speed of algorithm. Because the bigger the size more reliable are the results but more slowly runs the algorithm. So there exist a trade-off which the engineer should be carry on. One final parameter that should be taken into account, is the size of the smoothing Gaussian windows applied at the step before computing edges.

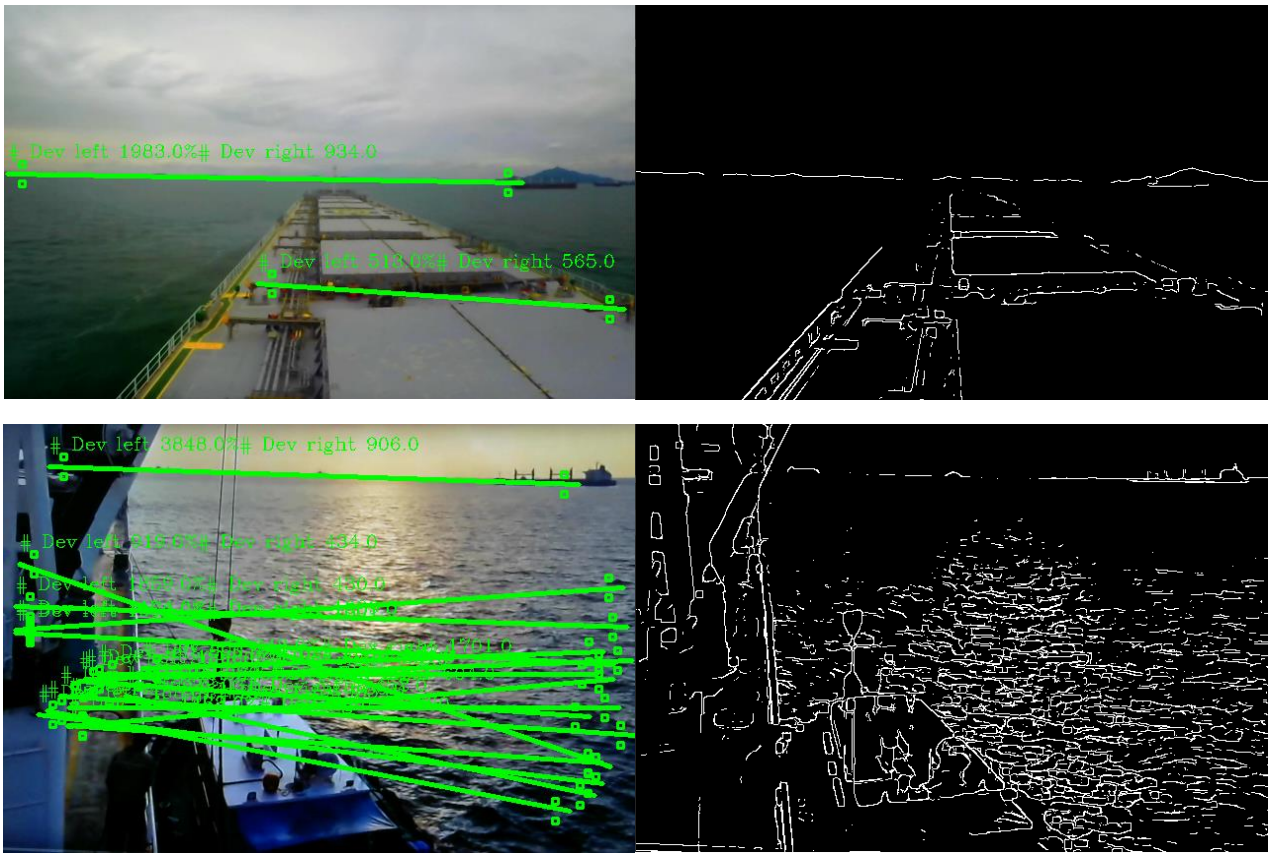

Figure 4.7.3 Horizon line detected left and the correspondent final edge-map right, before thresholding, are depicted. As it is clearly shown false detections due to waves or patterns like the horizon line occurs very often. The need of properly adjusting all the parameters of the algorithm is clearly depicted.

After thresholded and configured properly the window size at 10 pixels square, we were able to discard false horizon line detection, keeping only the dominant horizon line (Fig. 4.8). This technique works also when the horizon line is not expanded in the whole image, but in a small segment. For example when there is another objects like See-Vessel or islands. Although this technique is giving more robustness to overall horizon line detection algorithm, the horizon line should be existed as a segment for more than 50% of the width of the image. The horizon line was programmed in python using OpenCV. The related code is at appendix.

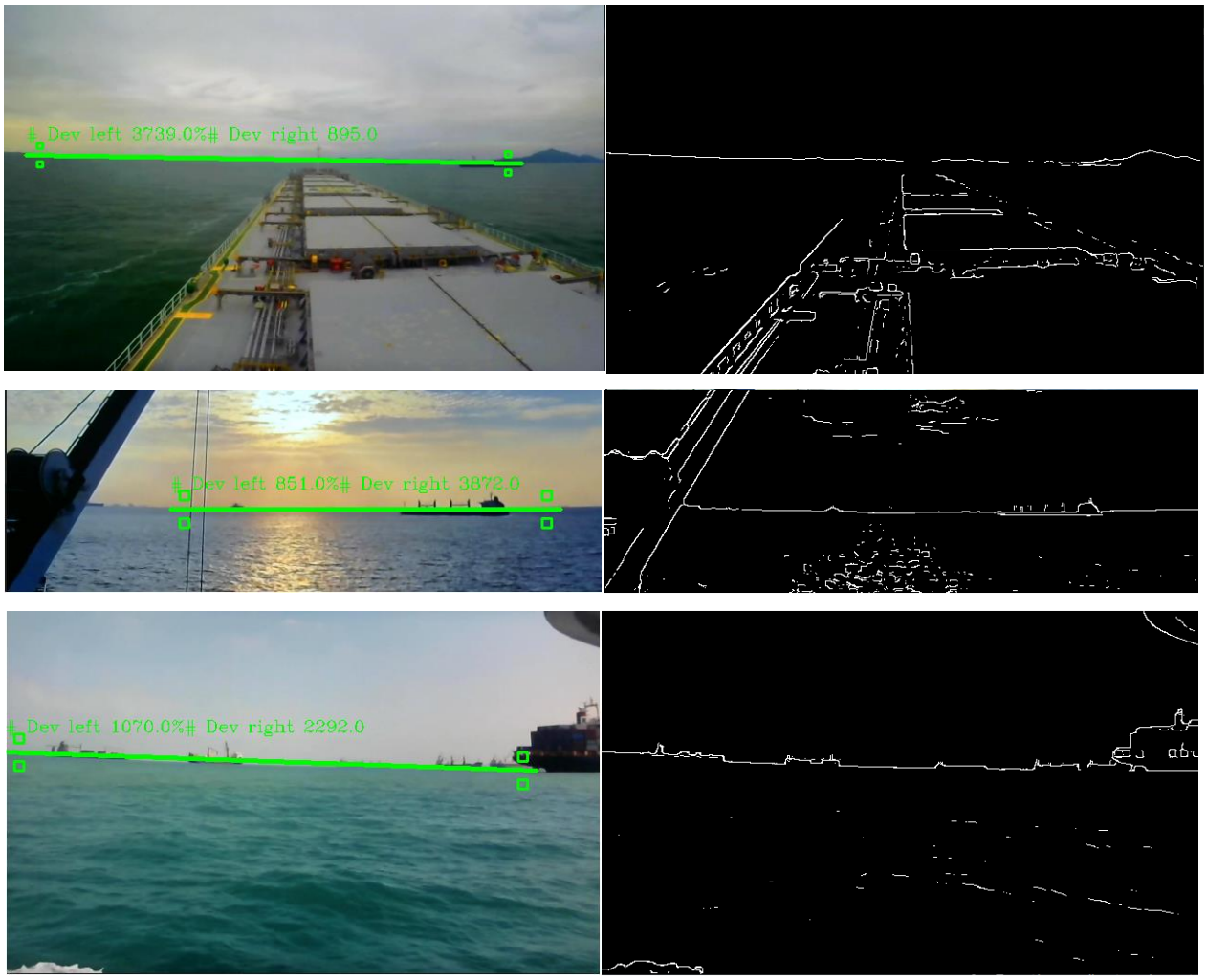

Figure 4.8.1 The resulting Horizon line detection algorithm. In the left, images of the horizon line as well as the small window for comparing the intensity values above and under the line are presented. It is also written the squared deviation value of the two windows in each side. In the right are shown the correspondent final edge map of the left images

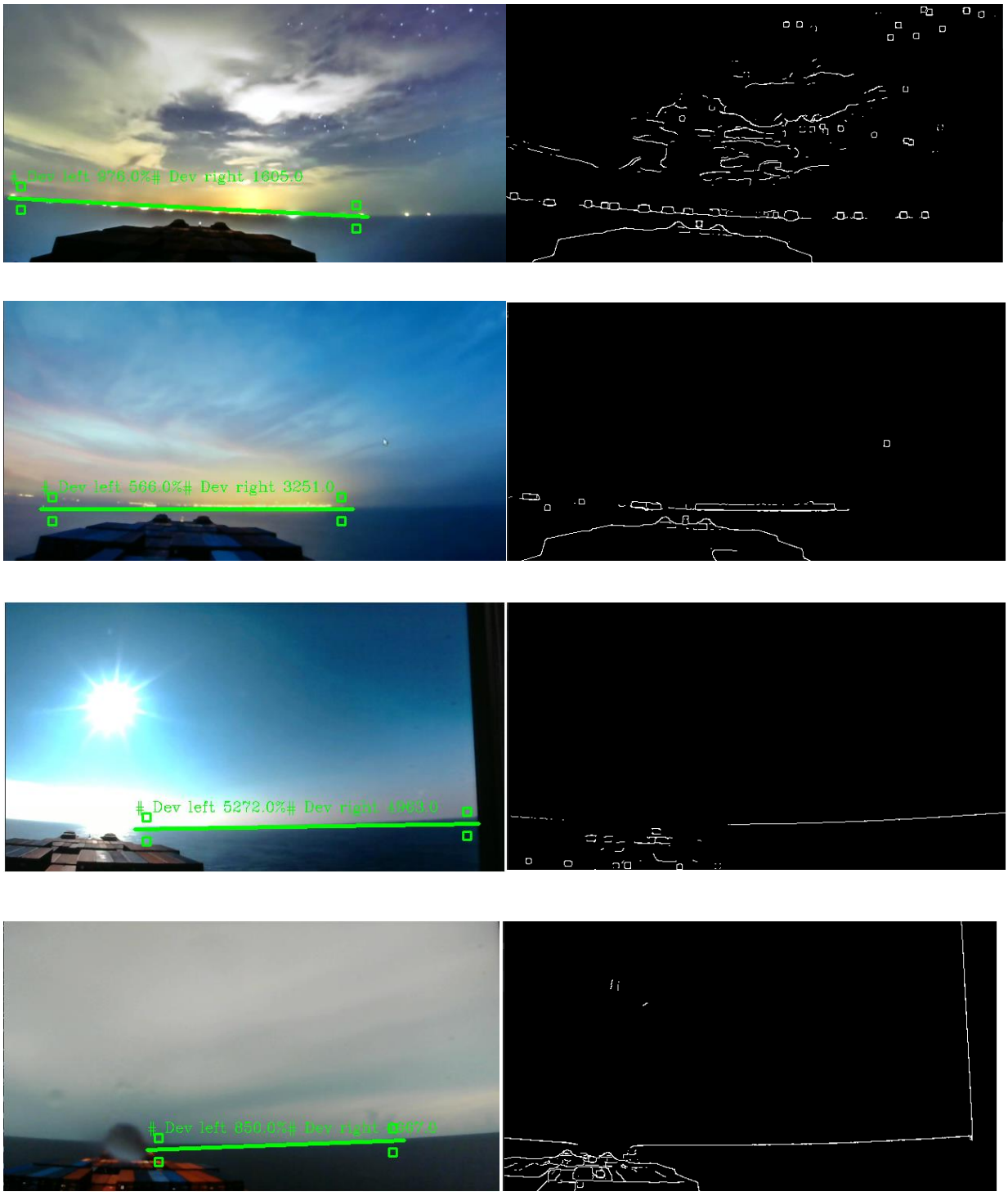

Figure 4.8.2 Images of detected horizon lines left and the corresponding final edge-map right. The images depicted the capability and robusteness of the proposed algorithm to detect horizon lines in different light conditions

Chapter 5

5 Stereo Fussion & 3D Position Estimation

This chapter is focused on the estimation of the 3D position of the detected object. This is accomplished by the use of a second camera, which enhances our system with more optical information, necessary to estimate the 3D position of the See-Vessels corresponding to the cameras. After describing the fundamentals of stereo vision, which are essential for our application, the method of fussing the images from the two camera sensors is analyzed. Then a proposed method for noise reduction and better position estimation of the detected See-Vessels is presented. This method is capable of correcting the error in camera orientation, compare to each other, in every frame and without the need of use of external fix-points (markers) calibrated with respect to the stereo system, as it is done in the majority of stereo based 3D position estimation system.

5.1 Camera Model

Most of the time cameras are model using the pinhole camera model. This is because of simplicity of this camera model. A pinhole is an imaginary wall with a tiny hole in the center, that blocks all rays except those passing through the tiny aperture in the center. In this section, we will start with a pinhole camera model to get a handle on the basic geometry of projecting rays. Unfortunately, a real pinhole is not a very good way to make images because it does not gather enough light for rapid exposure. This is why human eyes and cameras use lenses to gather more light than what would be available at a single point. The downside, however, is that gathering more light with a lens not only forces us to move beyond the simple geometry of the pinhole model but also introduces distortions from the lens itself [10].

5.1.1 Pinhole Camera Model – Intrinsics & Extrinsics

In pinhole camera model, light is envisioned as entering from the scene or a distant object, but only a single ray enters from any particular point. In a physical pinhole camera, this point is then “projected” onto an imaging surface. As a result, the image on this image plane (also called the projective plane) is always in focus, and the size of the image relative to the distant object is given by a single parameter of the camera: its focal length. For our idealized pinhole camera, the distance from the pinhole aperture to the screen is precisely the focal length. This is shown in Figure 5.1, where f is the focal length of the camera, Z is the distance from the camera to the object, X is the length of the object, and x is the object’s image on the imaging plane[14]. In the figure 5.1, we can see by similar triangles that $-x/f = X/Z$, or

$$x = \frac{f \cdot X}{Z}$$

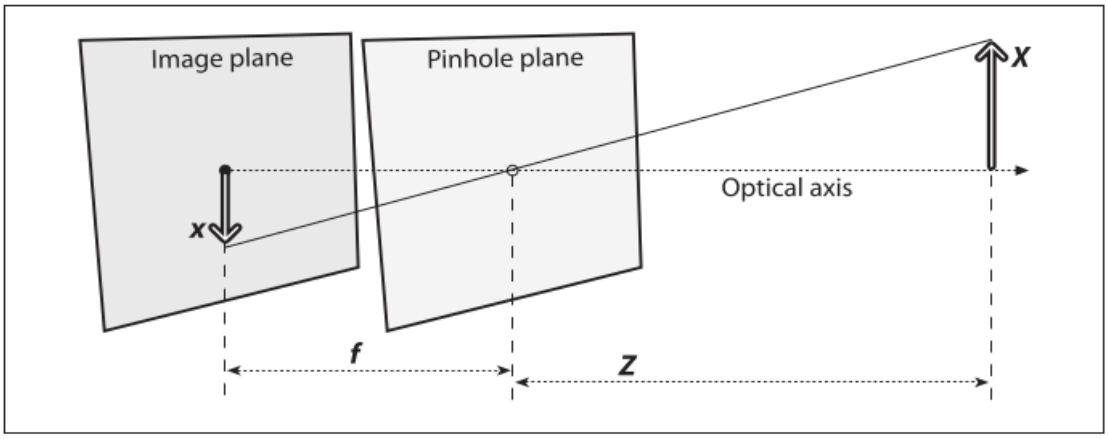

Figure 5.1 Pinhole camera model. The image plane corresponds to the pixel array of a typical camera sensor. The optical rays passing through the pinhole aperture from the pinhole plane, which is focal length away from the image plane[14].

In figure 5.2, the pinhole and the image plane change position. The main difference is that the object now appears as it is, right side up. The point in the pinhole is reinterpreted as the center of projection. Using this interpretation, every ray leaves a point on the distant object and heads for the center of projection. The point at the intersection of the image plane and the optical axis is referred to as the principal point. On this new frontal image plane (see Figure 5.2), which is the equivalent of the old projective or image plane, the image of the distant object is exactly the same size as it was on the image plane in figure 5.1 [14].

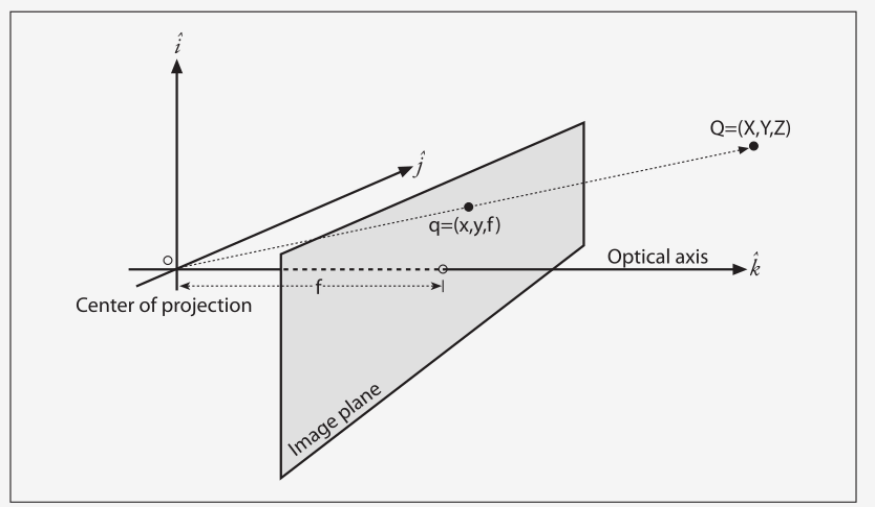

Figure 5.2 The projection model of a pinhole camera is presented. A point Q = (X, Y, Z) is projected onto the image plane by the ray passing through the centre of projection, and the resulting point on the image is q = (z, y, f) The image is generated by intersecting these rays with the image plane, which happens to be exactly a distance f from the center of projection.

This makes the similar triangles relationship x/f = X/Z more directly evident than before. The negative sign is gone because the object image is no longer upside down.

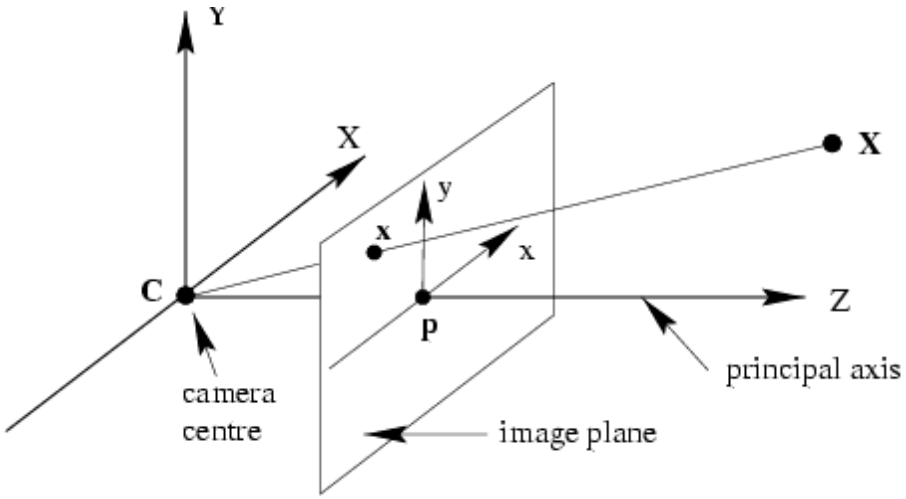

Figure 5.3 Basic Projection model of a pinhole camera.

As discussed earlier, a point X in 3D space can be mapped (or projected) into a 2D point x in the image plane Π′ (Fig. 5.3). This R3→R2 mapping is referred to as a projective transformation. This projection of 3D points into the image plane does not directly correspond to what we see in actual digital images for several reasons. First, points in the digital images are, in general, in a different reference system than those in the image plane. Second, digital images are divided into discrete pixels, whereas points in the image plane are continuous. Finally, the physical sensors can introduce non-linearity such as distortion to the mapping. To account for these differences, we will introduce a number of additional transformations that allow us to map any point from the 3D world to pixel coordinates. Image coordinates have their origin P at the image center where the Z axis intersects the image plane (Fig. 5.3). On the other hand, digital images typically have their origin at the lower-left corner of the image [22]. Thus, 2D points in the image plane and 2D points in the image are offset by a translation vector $[c_x,c_y]T$. To accommodate this change of coordinate systems, the mapping now becomes

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} f\frac{x}{z} + c_x \\ f\frac{y}{z} + c_y \end{bmatrix} \quad (1)$$

The next effect we must account for is that the points in digital images are expressed in pixels, while points in image plane are represented in physical measurements (e.g. centimeters). In order to accommodate this change of units, we must introduce two new parameters k and l. These parameters, whose units would be something like pixels*m^{-1}, correspond to the change

of units in the two axes of the image plane. Note that k and l may be different because the aspect ratio of the unit element is not guaranteed to be one. If k=l, we often say that the camera has square pixels [22]. The previous mapping is adjusted to be

$$P' = \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} fk\frac{x}{z} + c_x \\ fl\frac{y}{z} + c_y \end{bmatrix} = \begin{bmatrix} \alpha\frac{x}{z} + c_x \\ \beta\frac{y}{z} + c_y \end{bmatrix} \quad (2)$$

In order to form better this non linear transformation from world points (x,y,z) to discrete camera points (x', y'), we attempt to rewrite this equation using matrix multiplications between a matrix and the input vector P = (x,y,z). However, from Equation 2, it is obvious that this projection P→P′ is not linear, as the operation divides one of the input parameters (namely z). Still, representing this projection as a matrix-vector product would be useful for future derivations [22].

One way to get around this problem is to change the coordinate systems. A new coordinate is introduced, such that any point P′=(x′,y′) becomes (x′,y′,1). Similarly, any point P=(x,y,z) becomes (x,y,z,1). This augmented space is referred to as the homogeneous coordinate system. As it is known, to convert some Euclidean vector (v1,...,vn) to homogeneous coordinates, we simply append a 1 in a new dimension to get (v1,...,vn,1). Note that the equality between a vector and its homogeneous coordinates only occurs when the final coordinate equals to one. Therefore, when converting back from arbitrary homogeneous coordinates (v1,...,vn, w), we get Euclidean coordinates (v1/w,...,vn/w, 1)[22]. Using homogeneous coordinates, we can formulate

$$P'_h = \begin{bmatrix} \alpha x + c_x z \\ \beta y + c_y z \\ z \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P_h \quad (3)$$

From this point on, assume that we will work in homogeneous coordinates, unless stated otherwise. We will drop the h index, so any point P or P′ can be assumed to be in homogeneous coordinates [22]. As seen from equation 3,we can represent the relationship between a point in 3D space and its image coordinates by a matrix vector relationship:

$$P' = \begin{bmatrix} x' \\ y' \\ z \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x & 0 \\ 0 & \beta & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} P = MP \quad (4)$$

This transformation can be decomposed into:

$$P' = MP = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} I & 0 \end{bmatrix} P = K \begin{bmatrix} I & 0 \end{bmatrix} P \tag{5}$$

The matrix K is often referred to as the camera matrix. This matrix contains some of the critical parameters that are useful to characterize a camera model. Two parameters are currently missing from our formulation: skewness and distortion. We often say that an image is skewed when the camera coordinate system is skewed. In this case, the angle between the two axes are slightly larger or smaller than 90 degrees. Most cameras have zero-skew, but some degree of skewness may occur because of sensor manufacturing errors [22]. Deriving the new camera matrix accounting for skewness is outside the scope but is shown bellow:

$$K = \begin{bmatrix} x' \\ y' \\ z \end{bmatrix} = \begin{bmatrix} \alpha & -\alpha \cot \theta & c_x \\ 0 & \frac{\beta}{\sin \theta} & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{6}$$

So far, it is described a mapping between a point P in the 3D camera reference system to a point P′ in the 2D image plane. But what if the information about the 3D world is available in a different coordinate system? Then, there is need to include an additional transformation that relates points from the world reference system to the camera reference system. This transformation is captured by a rotation matrix R and translation vector T [22]. Therefore, given a point in a world reference system Pw, we can compute its camera coordinates as follows:

$$P = \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} P_w \tag{7}$$

Substituting this in equation (5) and simplifying gives:

$$P' = K \begin{bmatrix} R & T \end{bmatrix} P_w = M P_w \tag{8}$$

This completes the mapping from a 3D point P in an arbitrary world reference system to the image plane. As is shown the projection matrix M consists of two types of parameters: intrinsic and extrinsic parameters. All parameters contained in the camera matrix K are the intrinsic parameters, which change as the type of camera changes. The extrinsic parameters include the

rotation and translation, which do not depend on the camera's build. Overall, one can find that the 3×4 projection matrix M has 11 degrees of freedom: 5 from the intrinsic camera matrix, 3 from extrinsic rotation, and 3 from extrinsic translation[22].

5.1.2 Lens Distortion Models

Real world cameras are using lens. Lens are not perfect and therefore they introduce distortion. The reason is mainly because of the manufacturing process. Perfect lens need to be parabolic whether in reality, mathematically ideal parabolic scheme is not easy achievable. Most of the times real world camera lens are more spherical and not perfectly align to camera sensor. This means that the center of the Len is not align exactly with the center of the image sensor (CCD or CMOS). Taking account those inaccuracies the pinhole model that were described previously lacks. It is proofed that for simply applications, lens distortions can be as Radial and Tangential [14][22].

Radial distortion, distort the rays of light that are near the edges of the imager. This bulging phenomenon is the source of the "barrel" or "fish-eye" effect. Figure 5.4 gives some intuition as to why radial distortion occurs. With some lenses, rays farther from the center of the lens are bent more than those closer in. A typical inexpensive lens is, in effect, stronger than it ought to be as you get farther from the center. Barrel distortion is particularly noticeable in cheap web cameras but less apparent in high-end cameras, where a lot of effort is put into fancy lens systems that minimize radial distortion [14].

As far as radial distortion is concerned, the distortion is 0 at the (optical) center of the imager and increases as we move toward the periphery. In practice, this distortion is small and can be characterized by the first few terms of a Taylor series expansion around r = 0. For cheap web cameras, the first two such terms are used; the first of which is conventionally called k1 and the second k2. For highly distorted cameras such as fish-eye lenses one can use a third radial distortion term k3. In general, the radial location of a point on the imager will be rescaled according to the following equations [14]:

$$x_{\text{corrected}} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{\text{corrected}} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

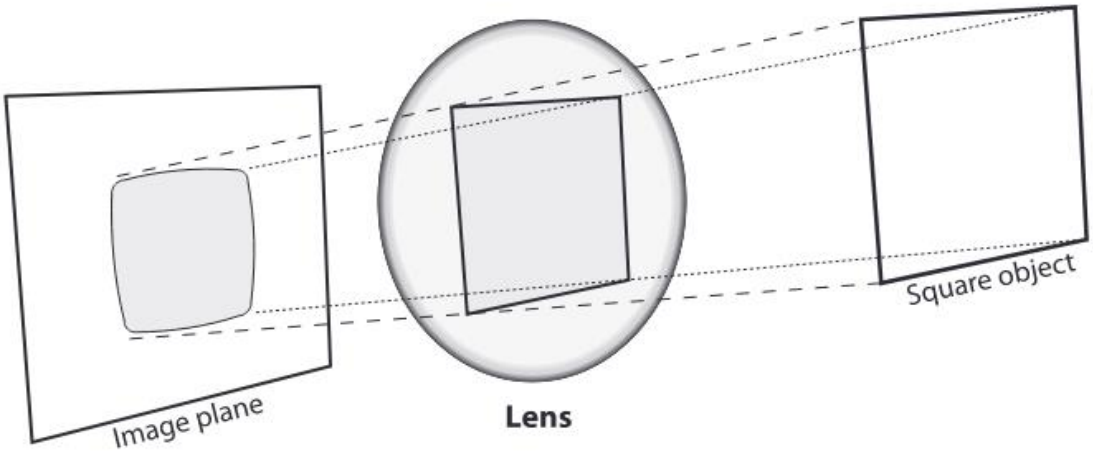

Figure 5.4 Radial distorting is presented. Objects rays that passes away from the center of the lens have more radial distortion than the others passing closer from the lens center[14].

Where x,y correspond to the original location (on the imager) of the distorted point and (xcorrected, ycorrected) correspond to the new location as a result of the correction. Generally there are two types of radial distortion. The radial distortion is increasing as the radial distance of the point (x,y) in image plane, from the optical center increases [14].

One can safely classify the radial distortion as pincushion distortion when the magnification increases and barrel distortion when the magnification decreases as shown in figure 5.5. Radial distortion is caused by the fact that different portions of the lens have differing focal lengths [14].

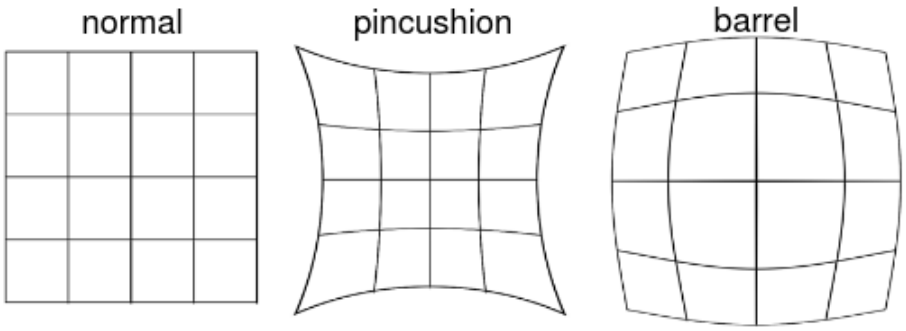

Figure 5.5 Showing the two different cases of barrel distortion [22]

Tangential distortion is the second-largest common distortion figure 5.6. Tangential distortion is due to manufacturing defects resulting from the lens not being exactly parallel to the imaging plane. Tangential distortion is minimally characterized by two additional parameters, p1 and p2, such that:

$$x_{\text{corrected}} = x + [2p_1 y + p_2(r^2 + 2x^2)]$$

$$y_{\text{corrected}} = y + [p_1(r^2 + 2y^2) + 2p_2 x]$$

Finally, there are in total 5 distortion coefficients that are required for a proper distortion elimination.

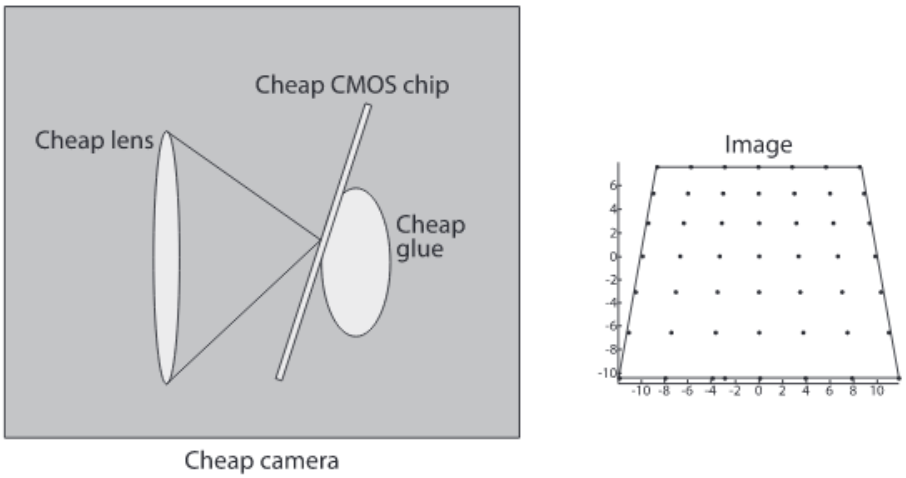

Figure 5.6 Tangential distortion due to manufacturing process error in correctly positioning and aligning of lens and image sensor.

At this point a complete mathematically model for a digital camera is formed. As is presented above the model parameters can be divided into intrinsic, extrinsic and distortion parameters. In order to compute all these parameters, there is a technique calling Calibration. Calibration uses some known 3D points corresponding to a world frame (Figure 5.7). Then after taking image frames of these points, the number of the frame and the number of points differ for every calibration method, the desired parameters are estimated. The majority of these methods are using iterative algorithms, minimizing a cost function and forming an optimization problem, in order to compute all the camera parameters. It is not intended to describe the calibration technique and its different calibration methods. One can be referred to [41].

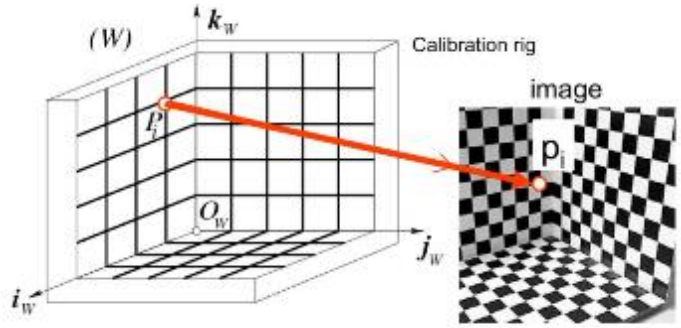

Figure 5.7 Calibration rig and corresponding image point. From point correspondeces, iterative algorithms find the parameters of the camera.

We used the calibration method proposed from OpenCV. The algorithm that OpenCV uses to solve for the focal lengths and offsets is based on Zhang's method [41], but OpenCV uses a different method based on Brown [12] to solve for the distortion parameters. OpenCV algorithm uses a plane chessboard. This chessboard is captured in image frames in different angles and based on those frames all the camera parameters are computed [14].

For our system implementation we did several calibrations and took the average of those values for a more accurate result. In figure 5.8 one can see the detected corners of the chessboard.

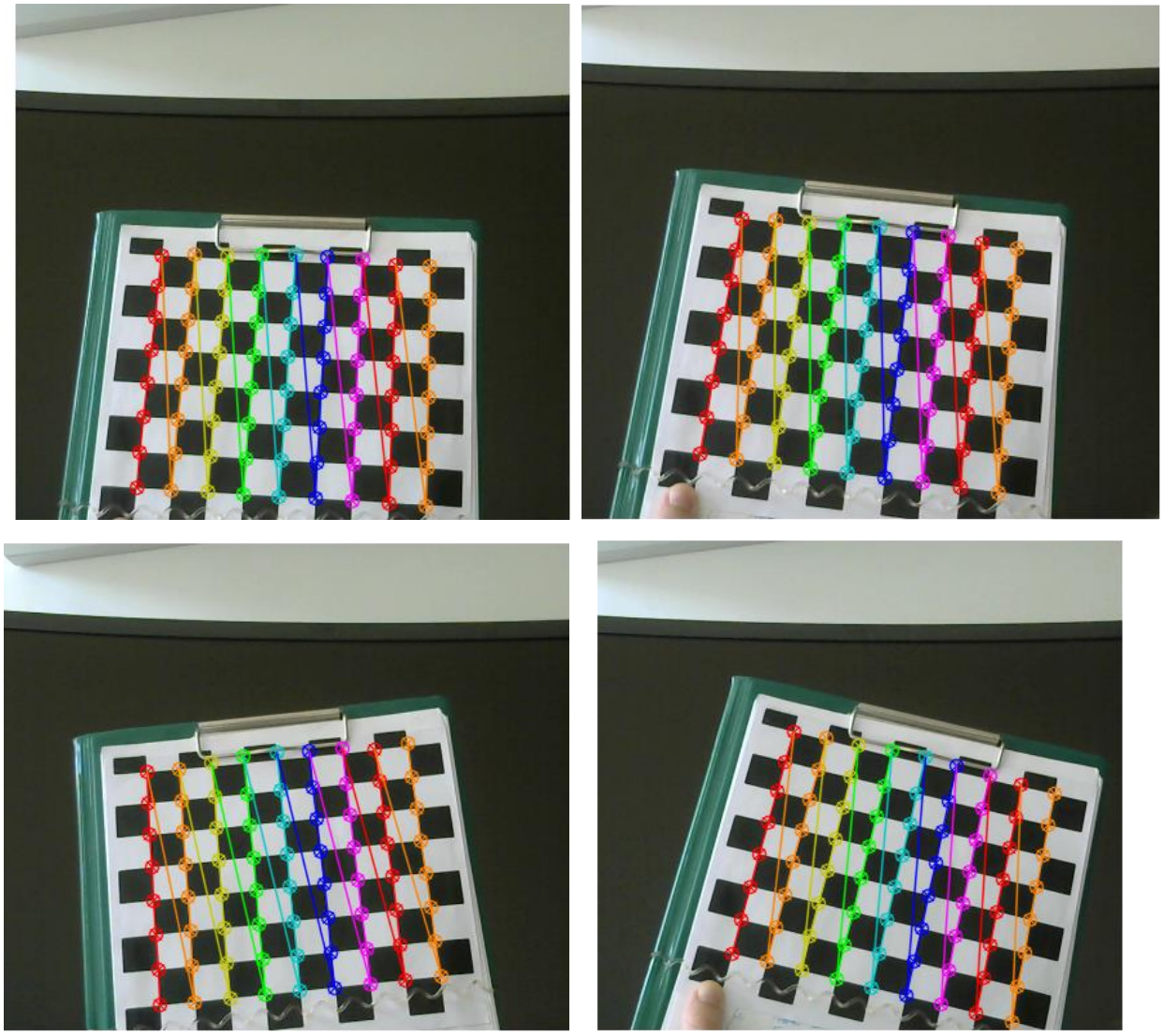

Figure 5.8 Calibration board and different position of the calibration board.

In the following images (Figure 5.9) a comparison between the raw image as it is taken from the system's camera affected to lens distortion and the undistorted version of the same image is presented. As it can be show in the undistorted image, some black regions are created in the edges of the image as a result of missing pixels dues to lens distortion.

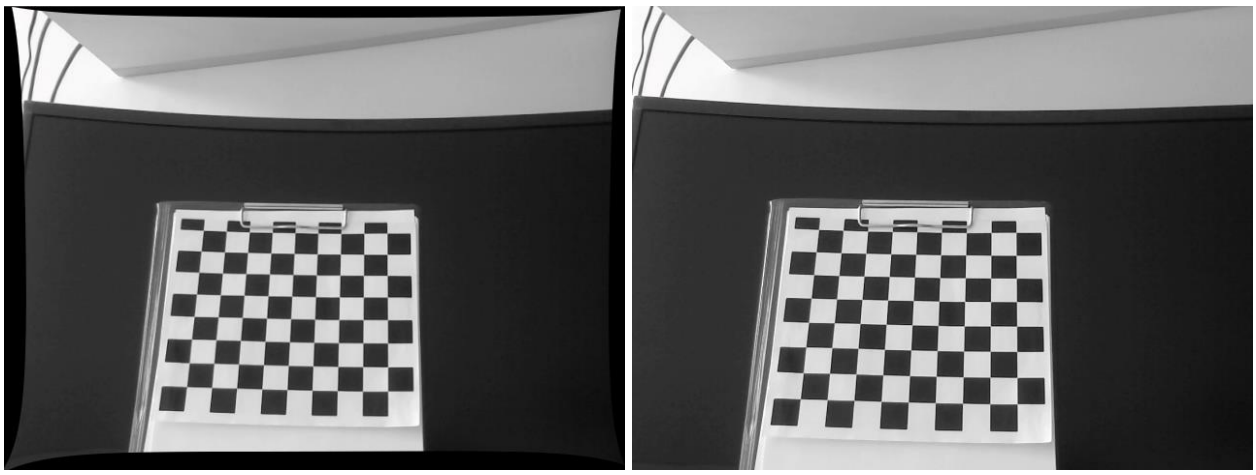

Figure 5.9 Left the undistorted image of the right raw image as it was taken directly from the camera. As it is shown in the left the radial distortion is dominating the distortion of the image. At the edges of the left image one can see the affect of the lens distortion. After un-distortion curves that were curves remain and lines that were curved due to distortion are corrected.

5.2 Stereo Imaging and 3D Pose Estimation

Let's swift to stereo imaging and position estimation of object of interest in an image. Despite the wealth of information contained in an image frame, the depth of a scene point along the corresponding projection ray is not directly accessible in a single image. With at least two pictures, on the other hand, depth can be measured through triangulation. This is of course one of the reasons why most animals have at least two eyes and/or move their head when looking for friend or foe, as well as the motivation for equipping autonomous robots with stereo or motion analysis systems. Before building such a program, we must understand how several views of the same scene constrain its three-dimensional structure as well as the corresponding camera configurations [10].

5.2.1 Epipolar Geometry

Epipolar geometry is the basic geometry of a stereo imaging system. In essence, this geometry combines two pinhole models (one for each camera) and some interesting new points called the epipoles (see Figure 5.10). Before explaining what these epipoles are good for, we will start by taking a moment to define them clearly and to add some related terminology. When we are done, we will have a concise understanding of this overall geometry and will also find that we

can narrow down considerably the possible locations of corresponding points on the two stereo cameras [10]. This added discovery will be important to our stereo implementation.

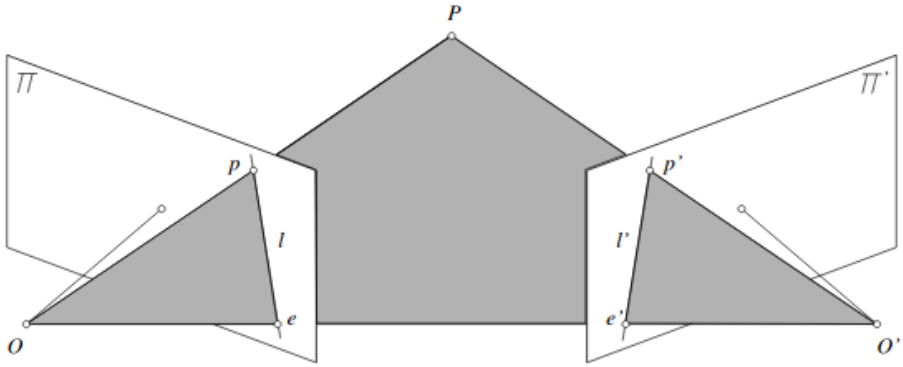

Figure 5.10 The Epipolar Plane formed from the points P, O and O' of the epipolar geometry of the cameras, is colored gray. In this plane belongs the center of the cameras (O & O') , the two image points p , p' of P and the world point P.

Consider the images **p** and **p′** of a point **P** observed by two cameras with optical centers **O** and O′. These five points all belong to the epipolar plane defined by the two intersecting rays **OP** and **O′P** (Figure 5.10). In particular, the point **p′** lies on the line **l′** where this plane and the retina **Π′** of the second camera intersect. The line **l′** is the epipolar line associated with the point p, and it passes through the point **e′** where the baseline joining the optical centers **O** and **O′** intersects **Π′**. Likewise, the point **p** lies on the epipolar line **l** associated with the point **p′**, and this line passes through the intersection e of the baseline with the plane **Π** [10].

The points e and e′ are called the epipoles of the two cameras. The epipole **e′** is the (virtual) image of the optical center **O** of the first camera in the image observed by the second camera, and vice versa. As noted, before, if **p** and **p′** are images of the same point, then **p′** must lie on the epipolar line associated with **p**. This epipolar constraint plays a fundamental role in stereo vision and motion analysis [10].

The epipoles and the epipolar constrain help us founding correspondences between the images in the stereo configuration. Assuming that our cameras are calibrated and the intrinsic and extrinsic parameters are known, as well as the distortion coefficients, a point view from the left camera (**x**) can be found to the right camera (**x′**) lining to the epipolar line (**l′**) in the right camera, that correspond to the point in the left. The above description will be given in a mathematically form bellow. So, the epipolar constraint greatly limits the search for these correspondences (figure 5.11). For choosing the best match, more constraints are necessary in order to decide [10].

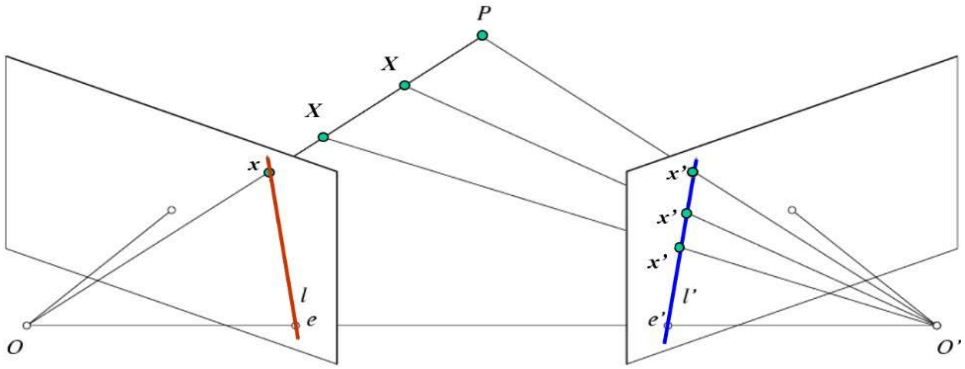

Figure 5.11 Epipolar geometry and epipolar constrains

5.2.2 Essential Matrix

Let's formulate the epipolar constraint that was described above with the assumption that our cameras are calibrated, meaning that the intrinsic and extrinsic parameters are known and the distortion parameters as well (Figure 5.12). This implies that $\mathbf{x} = \hat{\boldsymbol{x}}$. Clearly, the epipolar constraint implies that the three vectors **Ox, O′x′,**and **OO′** are coplanar [10]. Equivalently, one of them must lie in the plane spanned by the other two, or

$$\vec{Ox} \quad \left[\vec{OO'} \times \vec{O'x'}\right] = 0 \qquad \text{(Eq. 9)}$$

Rewriting this coordinate-independent equation in the coordinate frame associated to the first camera as:

$$x \quad \left[t \times Rx'\right] = 0 \qquad \text{(Eq. 10)}$$

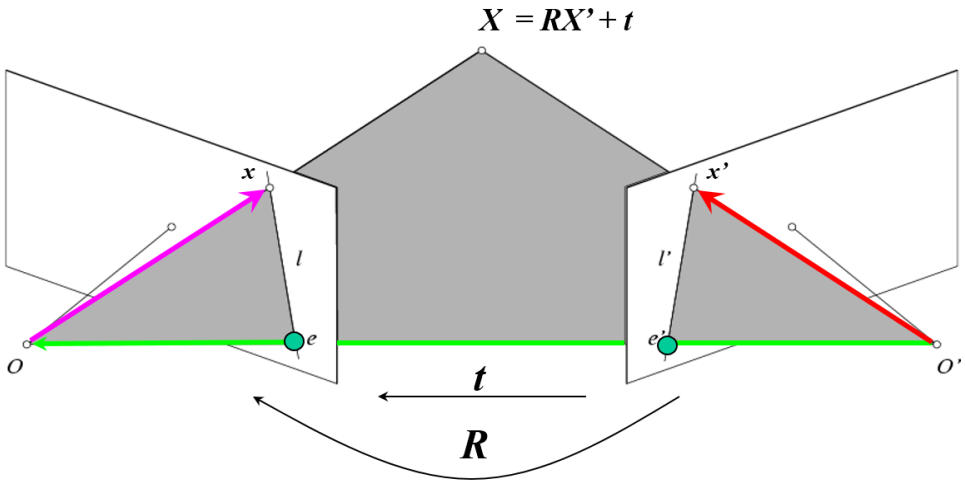

Figure 5.12 Epipolar Geometry.

where $x=(u,v,1)^T$ and $x'=(u',v',1)^T$ denote the homogenous image coordinate vectors of **x** and **x′**, t is the coordinate vector of the translation **OO′** separating the two coordinate systems, and **R** is the rotation matrix such that a free vector with coordinates **w′** in the second coordinate system has coordinates **Rw′** in the first one (in this case the two projection matrices are given in the coordinate system attached to the first camera by **(Id 0)** and $\mathbf{(R^T, -R^T t)}$ [10]. Equation (10) can finally be rewritten as:

$$x^T E x' = 0 \qquad \text{(Eq. 11)}$$

where $\mathbf{E=[t_\times]R}$, and $\mathbf{[a_\times]}$ denotes the skew-symmetric matrix such that $\mathbf{[a_\times]x=a\times x}$ is the cross-product of the vectors **a** and **x**. The matrix **E** is called the essential matrix, and it was first introduced by Longuet-Higgins. Its nine coefficients are only defined up to scale, and they can be parameterized by the three degrees of freedom of the rotation matrix **R** and the two degrees of freedom defining the direction of the translation vector **t** [10].

Note that

- **Ex′** can be interpreted as the coordinate vector representing the epipolar line associated with the point **x′** in the first image: indeed, an image line **l** can be defined by its equation **au+bv+c=0**, where **(u,v)** denote the coordinates of a point on the line, **(a,b)** is the unit normal to the line, and c is the (signed) distance between the origin and **l**. Alternatively, we can define the line equation in terms of the homogeneous coordinate vector $\mathbf{x=(u,v,1)^T}$ of a point on the line and the vector $\mathbf{l=(a,b,c)^T}$ by **l·x= 0**, in which case the constraint $\mathbf{a^2+b^2= 1}$ is relaxed since the equation holds independently of any scale change applied to **l**.
- Eq. (11) expresses the fact that the point **x** lies on the epipolar line associated with the vector **Ex′**.
- By symmetry, it is also clear that **Ex′** is the coordinate vector representing the epipolar line associated with x in the second image.
- Epipoles belongs also to the epipolar **E· e =0** lines so for the left epipole and $\boldsymbol{e'^T \cdot E = 0}$
- Essential matrix **E** is singular, having **rank=2**

5.2.3 Fundamental Matrix

The essential matrix E uses camera coordinates. This implies that the intrinsic parameters of the cameras are known, because by looking to pixel coordinates one can use the following equations and compute the image coordinates. But what happens when the cameras are not calibrated and the intrinsic parameters (Camera Calibration Matrix -K) are not known? The solution is given by the Fundamental matrix. The Fundamental matrix encapsulates the change

of coordinates and gives us a method to compute the epipolar lines and to work directly to image pixel coordinates [10].

$$\begin{aligned} \widehat{x} &= K_l \Longleftrightarrow x = K_l^{-1}\widehat{x} \\ \widehat{x'} &= K_r \Longleftrightarrow x' = K_r^{-1}\widehat{x'} \end{aligned} \qquad \text{(Eq. 12)}$$

By substituting Eq. 12 to Eq. 11 we compute:

$$\begin{aligned} \left(K_l^{-1}\hat{x}\right)^T E\left(K_r^{-1}\hat{x'}\right) &= 0 \\ \hat{x}^T\left(K_l^{-1\,T} E K_r^{-1}\right)\hat{x'} &= 0 \\ \hat{x}^T F \hat{x'} &= 0 \end{aligned} \qquad \text{(Eq. 13)}$$

Where:

$$F = K_l^{-1\,T} E K_r^{-1} \qquad \text{(Eq. 14)}$$

is the Fundamental matrix, which gives us the ability to work with the images without the need to calibrate our cameras. It has rank 2 and depends on the intrinsic and extrinsic parameters (f, R, t) [10].

Likewise with essential matrix the fundamental matrix is defined by seven independent coefficients and can in principle be estimated from seven point correspondences. Methods for estimating the essential and fundamental matrices from a minimal number of parameters indeed exist, but they are far too involved to be described here [10].

One simple estimation of fundamental matrix could be found by forming the Eq.13 into the following form, using at least 8 points. It is also called the 8-point algorithm [10].

$$(u,v,1)\begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix}\begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = 0 \Leftrightarrow (uu', uv', u, vu', vv', v, u', v', 1)\begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = 0. \qquad \text{(Eq. 15)}$$

Since (Eq. 15) is homogeneous in the coefficients of F, one can for example set F33= 1 and use eight point correspondences xi↔x′i(i=1,..,8) to setup an 8×8 system of non-homogeneous linear quations:

$$\begin{pmatrix} u_1u'_1 & u_1v'_1 & u_1 & v_1u'_1 & v_1v'_1 & v_1 & u'_1 & v'_1 \\ u_2u'_2 & u_2v'_2 & u_2 & v_2u'_2 & v_2v'_2 & v_2 & u'_2 & v'_2 \\ u_3u'_3 & u_3v'_3 & u_3 & v_3u'_3 & v_3v'_3 & v_3 & u'_3 & v'_3 \\ u_4u'_4 & u_4v'_4 & u_4 & v_4u'_4 & v_4v'_4 & v_4 & u'_4 & v'_4 \\ u_5u'_5 & u_5v'_5 & u_5 & v_5u'_5 & v_5v'_5 & v_5 & u'_5 & v'_5 \\ u_6u'_6 & u_6v'_6 & u_6 & v_6u'_6 & v_6v'_6 & v_6 & u'_6 & v'_6 \\ u_7u'_7 & u_7v'_7 & u_7 & v_7u'_7 & v_7v'_7 & v_7 & u'_7 & v'_7 \\ u_8u'_8 & u_8v'_8 & u_8 & v_8u'_8 & v_8v'_8 & v_8 & u'_8 & v'_8 \end{pmatrix} \begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \end{pmatrix} = - \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \quad \text{(Eq. 16)}$$

which is sufficient for estimating the fundamental matrix. This is the eight-point algorithm proposed by Hugh Christopher Longuet-Higgins in 1981.

We used the embedded function from OpenCV [14], in order to compute the Fundamental matrix, using although more than 8 points. OpenCV function for estimating the fundamental matrix form a linear least square problem as :

$$\min \sum_{i=1}^{n} \left(x_i^T F x_i' \right)^2 \quad \text{(Eq. 17)}$$

with respect to the coefficients of F under the constraint that the vector formed by these coefficients has unit norm. Additionally we exploited the RANSAC functionality that this function provide in order to discard outliers. Points that are false matched. This is done by RANSAC algorithm which computes the dominant affine transformation of the points in left frame that are matched to the right. Points from left frame that are not correspond more than a distance threshold to the initial right matches, are considered false matches and are discarded. With this technique, the computation of Fundamental matrix is more precise and this is very important in 3D position estimation of the detected See-Vessels.

5.2.4 Stereo Calibration

Stereo calibration is the process of computing the geometrical relationship between the two cameras in space, namely the matrices **R, t.** Using the same technique as with the single camera calibration method, one can use a chessboard visible from both cameras and after taking several

pictures in different positions the rotation matrix and the translation vector of the two cameras are computed. The process is the same as per single camera calibration, with the difference that the correspondences of the same points, projected in the two images, are exploited.

In our system, there is need at the beginning of the configuration of the system to calibrate the cameras. We did that by using a chessboard and taking several frames in different positions. In the following images one can see some of the stereo calibration images that were used. We used the OpenCV function cv2.stereocalibrate() in order to compute the rotation matrix R and the translation vector t of the coordinate systems of the two cameras figure 5.13.

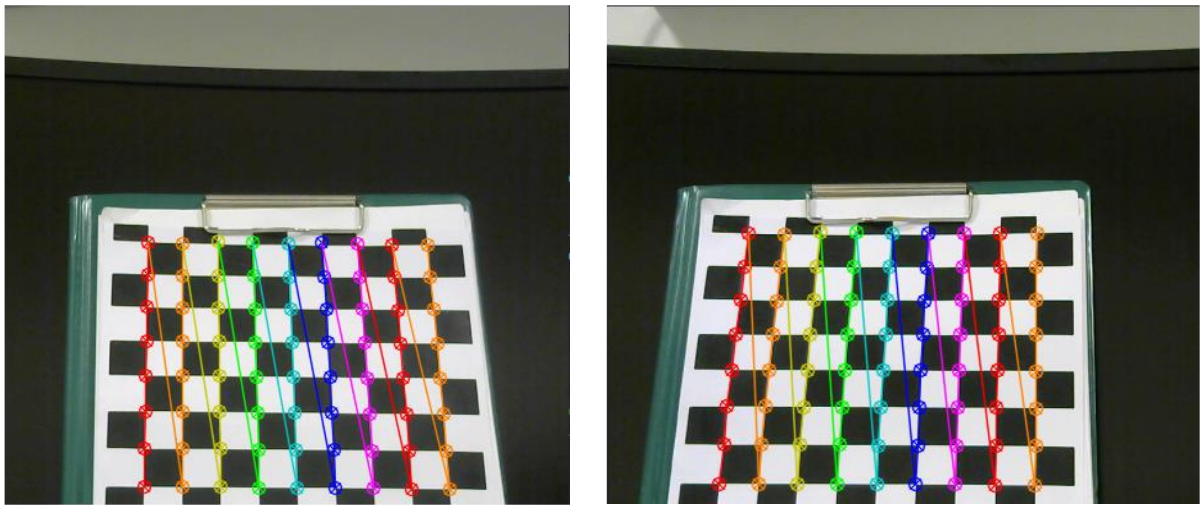

Figure 5.13 Left the calibration block and the founded blocks from the left camera. Right image the correspondent image from the right camera and the founded blocks of the calibration block.

5.2.5 3D Position Estimation

Estimating the position of object-points from the coordinate system of the camera is a well-established problem, which is often called in literature scene reconstruction. The aim is from images to reconstruct the scene that are captured in those images. This is not an easy task because of the discretization that is introduced from digital cameras, the distortion that produce on images not perfectly used lens and the inaccuracy between the cameras in stereo configuration, as well as of the matched points in two images. Although there are methods to estimate the depth of the scene and and as a result the 3D position of desired object-points, using a single camera or one frame, we focused on stereo configuration because of the accuracy and robustness that is provided [14].

The method to estimate the 3D position of object-points from images is by triangulating those points, with the correspondent points in the two frames and by exploiting the stereo configuration, as shown in the next figure 5.14. As is shown in the figure 5.14 bellow, in this simplified case, taking xl and xr to be the horizontal positions of the points in the left and right

frames (respectively), allows us to show that the depth is inversely proportional to the disparity between [14]

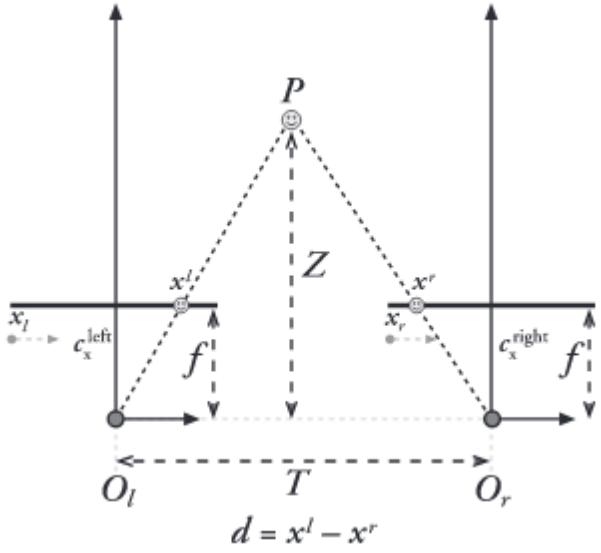

Figure 5.14 Depth estimation from perfectly undistorted, align stereo rig. [14]

these views, where the disparity is defined simply by $d=x_l - x_r - (c_l - c_r)$. Then by using similar triangles one can compute:

$$\frac{T - (x^l - c^l - x^r + c^r)}{Z - f} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{d - (c^l - c^r)} \qquad \text{(Eq. 18)}$$

where c_l and c_r are the coordinates of the principal points of the two image sensors, computed from the calibration process. We used this approach because in real world cheap cameras the two centers are not in the center of the sensor array (CCD /CMOS) as described in the lens distortion section above. A more helpful figure shows this case (figure 5.15)

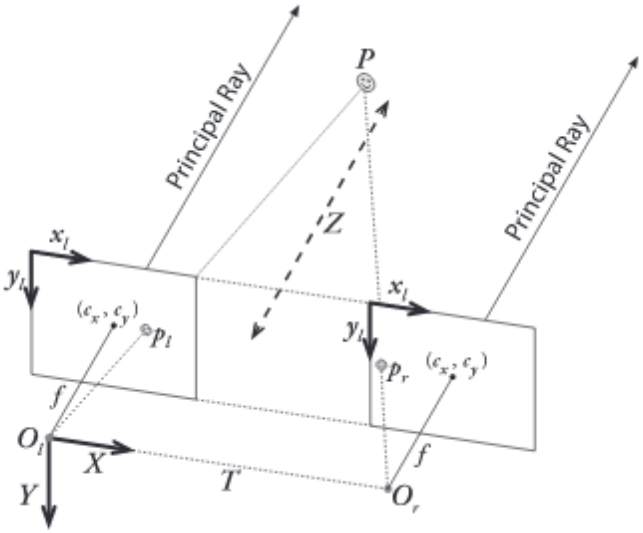

Figure 5.15 Stereo rig with undistorted and rectified images. It is also prevented the reference coordinate system for our system attached to the left camera. [14]

5.2.6 Stereo Rectification

As mentioned previously many times the images are thought as rectified. This means that their image planes are parallel-coplanar to each other and their epipolar lines are parallel lines with the images x-axis. This means that for every point in one image, one searchs for correspondences in the other image by looking at the line parallel to the image x-axis and with the same y value. This means that correspondences in right image lies to the line y = yl figure 5.16.

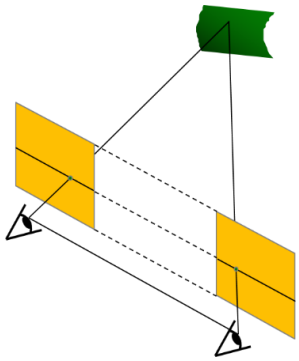

Figure 5.16 horizontal scan lines- epipolar lines of corrspondent points in two images

This is the reason, most stereo vision application are using parallel configured cameras. This simplified the correspondence search problem and the computation of the disparity and as a result the depth of the scene. When this configuration is not applied then one have to construct a plane, in which the two stereo images will be co-planar and row-align (figure 5.17). This is

done by using image transformations, in order to transfer the images into this configuration [14]. There are several methods that achieve this and can be mainly categorize into two main categories depending on the type of stereo configuration:

- Uncalibrated stereo rectification
- Calibrated stereo rectification

It is not intended, to be described the different rectification methods. The only thing we have to mention is that we used the rotation matrix R that relates the right camera coordinate system with the left camera coordinate system in each frame in order to form the co-planar and to row-alignment configuration that depicted in figure 5.17. This gave us better 3D position estimation results despite that the two cameras were intended to be as much as possible in a parallel configuration. One have not to do this step, if the parallel configuration of the cameras of the stereo rig is well established and the estimated 3D position accuracy is acceptable. But our system was though that it will be suffer from vibration due to ship movements. In our application we had to guarantee that small deviations will be eliminated. Those vibrations could happen, because of the heavy strikes of the Sea-Vessel (Ship) to diverse waves from small to big and/or on-board propulsion engine vibrations, causing instantaneous deviation from the desired configuration [14].

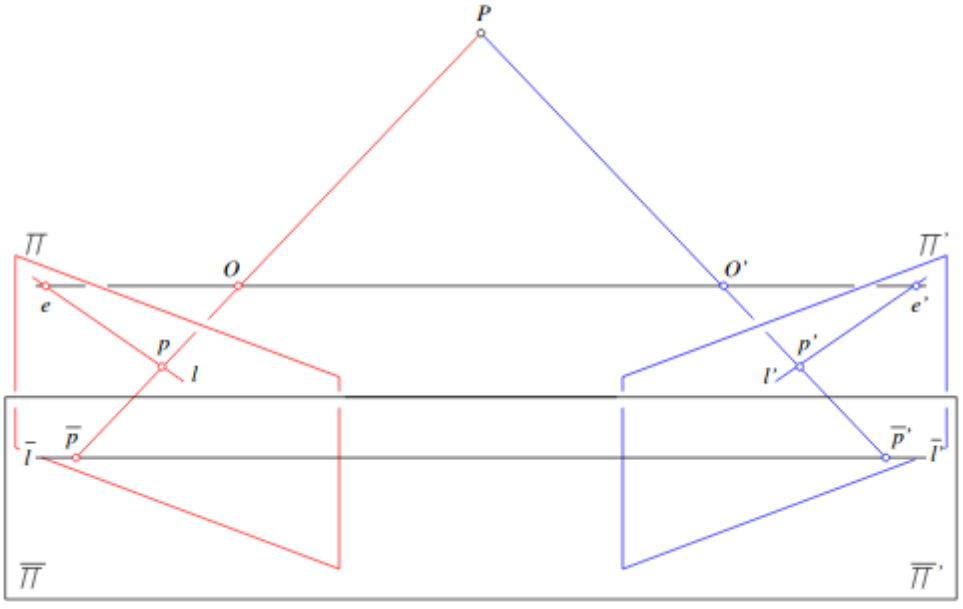

Figure 5.17 A rectified stereo pair: the two image planes Π and Π'are reprojected ontoa common plane Π= Π'parallel to the baseline. The epipolar lineslandl'associated withthe pointspandp'in the two pictures map onto a common scanline l= l'also parallelto the baseline and passing through the reprojected points p̄and p̄'. The rectified imagesare easily constructed by considering each input image as a polyhedral mesh and usingtexture mapping to render the projection of this mesh into the plane Π= Π'.

5.3 Stereo Fusion

As described our system uses a stereo rig (two cameras) in configurable position to each other and detects Sea-Vessels and estimates their 3D position corresponding to the left camera. More specifically we attach in each camera a coordinate system as depicted in figure 5.15. Also the coordinate system of the left camera is taken as the reference coordinate system of the whole system and the position of the detected Sea-Vessels are related with this. It is so far known that after detecting the Sea-Vessel objects from the left image using deep neural networks (YoloV3 or YoloV3tiny) we attempt then to compute the position of them.

For this reason we fuse the information coming from the right camera with this information coming from the left. This is done by computing the correspondences of the left detected object to the right. It is worth mentioned that special care was given to take as much as possible frames at the same time instant from both cameras. Thats why in our embedded system separated threads was established for each camera in order the signal, which starts the camera to take a frame, to have not to wait till the first camera finish the data transfer and then the second camera to start caputing and sending the image. The cameras were connected into the same serial bus via USB3 port on the Raspberry Pi4.

Having the right frame and knowing the object's vertical (y) position from the left camera, we exploited the fact that our stereo rig is coplanar and row-aligned and computed the matches on the right frame in each horizontal scan-line using as matching criterion the maximization of the normalized correlation of gray scale intensity values of the window object to each position of the right's frame scan-line .

$$R(x, y) = \frac{\sum_{x', y'} \left(T(x', y') \cdot I_r(x + x', y + y')\right)}{\sqrt{\sum_{x', y'} T(x', y')^2 \cdot \sum_{x', y'} I_r(x + x', y + y')^2}} \quad \text{(Eq. 20)}$$

Where T(x',y') corresponds to the template (detected Sea-Vessel) that we want to find at the right frame, I(x,y) the intensity values of the right frame.

The searching for correspondences of every Sea-Vessel object of the left frame was done along a horizontal line in the right image with height 10 to 20 pixels more than the height of the Sea-Vessel box. This is done because we don't undistort the frames in each cycle run of the algorithm, because it's a time consuming operation and we don't care for the rest of the information containing in the image.

So after finding the correspondent boxes with the Sea-Vessel in each frame, we undistort those points. This help us to optimize more the computational time.

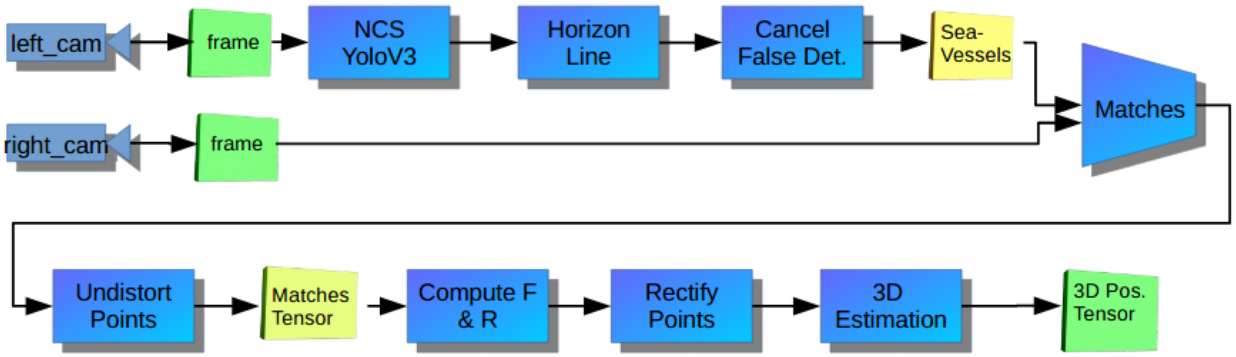

Figure 5.18 The overall Sea-Vessel detection and 3D position estimation algorithm pipeline.

5.4 Improved 3D Position Estimation

In the previous section a simple yet efficient method to estimate the depth of the scene and as a result the desired object points was presented. It was also mentioned what one should do when the pair of cameras in the stereo rig is not perfectly coplanar and row aligned. In this section it will be described the steps that was fallen in order to compensate the instant error of rotational position of the right camera related to the left. This error occurs very often in Sea environment, where different types of waves and Ship movements causes instantaneous vibrations.

The assumption of fixed, rigid position of each camera was used. In order to rectify the two images, the rotational matrix of the right camera related to the left is needed. In many systems and scientific works this is done by measuring the relative position of each camera from fixed, well calibrated points (markers) that are rigidly attached to a fixed distance from the cameras. This approach, although is more simpler and easy to algorithmic be implemented, it's difficult in practice because one have to precisely position those points in Ship Hull and quarantined that they are not affected from any type of vibrations as the cameras.

Our approach in the other hand, simply exploits the correspondences in each frame. After the points of bounding-boxes containing Sea-Vessels are undistorted, then the fundamental matrix is computed and then using SVD the rotational matrix of the right camera related to the the left is computed. Finally, we rectify the matched points of the right image and the left image and then the position of every detected See-Vessel is computed. This approach is far more robust and doesn't need special configuration and expensive position sensors for perfectly calculating the position of fixed markers. Finally, it is worth mention that the false matched points are filter through RANSAC algorithm at the stage of fundamental matrix computation.

The computation of the rotation matrix from the fundamental matrix done using singular value decomposition of the essential matrix and exploiting its rank 2 property.

$$E = K' F K \quad \text{(Eq. 21)}$$

Where K' and K are the calibration matrices of each camera.

It is already known that the cameras coordinate systems expressed to left camera's coordinate system [10]. Thus, the projection matrix of each camera is:

$$P = K (I \; 0) \quad \text{and} \quad P' = K' (R \; t)$$

Essential matrix has two equal singular values (eigenvalues) and third one that is zero. Based on the fact that E=SR, with S = [t]x we define:

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad and \quad Z = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \text{(Eq. 22)}$$

S can be decomposed as: S = kUZUT with UЄO(3)

We have Z=±diag(1, 1, 0) W and thus:

$$S = U diag(1 \; 1 \; 0) W U^T \quad \text{(Eq. 23)}$$

A Singular Value Decomposition of E is thus, by using the equations E= SR and (23):

$$E = U diag(1 \; 1 \; 0)(W U^T R) \quad \text{(Eq.24)}$$

The Singular Value Decomposition of E is:

$$E = U diag(1 \; 1 \; 0) V^T \quad \text{(Eq.25)}$$

by comparing equations (24) and (25) we get the following two possible solutions for R

$$R = U W V^T \quad \text{(Eq. 26)}$$

$$R = U W V^T \quad \text{(Eq. 27)}$$

Only one solution is feasible

After computing the rotation matrix of the right camera related to the left we rectified the points of the right camera by multiplying with the rotation matrix and then we used the left and right points to compute the 3D position of the detected See-Vessels

$$x_{left}^{rect} = R \quad x_{left} \qquad \text{(Eq. 28)}$$

then the disparity is computed:

$$d = x_{right} - x_{left}^{rect} \qquad \text{(Eq. 29)}$$

After computing the disparity and in order to increase the accuracy of the measuring distance we use subpixel accuracy of the disparity computation by using parabola fitting. In order to compute disparity with sub-pixel accuracy we. The sub-pixel estimation is based on the similarity measures of three-pixel locations – related pixel, previous pixel and next pixel– calculated by the normalized correlation. After calculating the normalized correlation values, parabola fitting is carried out,the centerline location of which is the estimated sub-pixel position, as shown in Figure 5.19. The following equations used for the calculation of the parabola fitting

$$d_{sub} = \frac{Cor(d-1) - Cor(d+1)}{2Cor(d-1) - 4Cor(d) + 2Cor(d+1)} \qquad \text{(Eq. 30)}$$

where Cor(d-1), Cor(d) and Cor(d+1) are the simillarity measures of the previous pixel, related pixel and next pixel locations respectively. Then the disparity is: d + dsub

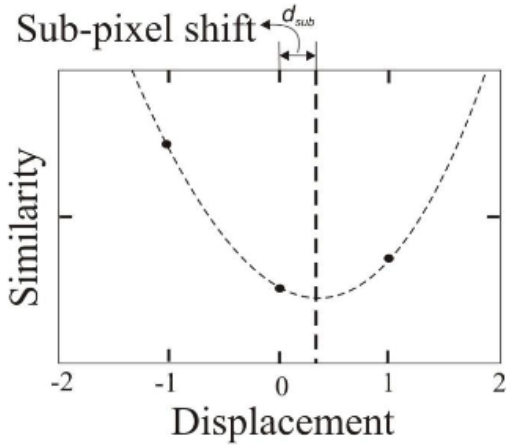

Figure 5.19 Parabola fitting of similarities in order to compute the sub-pixel position of the disparity [15]

Finally by using the reprojection matrix:

$$Q = \begin{pmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & -1/T_x & (c_x - c_x^{left})/T_x \end{pmatrix} \quad \text{(Eq. 31)}$$

then the computation of the position of the object in homogeneous coordinates is:

$$Q \begin{bmatrix} x \\ y \\ d \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} \quad \text{(Eq. 32)}$$

Where the 3D coordinates with respect to the left camera is

$$P_o = \begin{pmatrix} X/W \\ Y/W \\ Z/W \end{pmatrix} \quad \text{(Eq. 32)}$$

In order to demonstrate the benefits of the proposed algorithm to better 3D position estimation, we present following some images (Figure 5.20, 5.21, 5.22) of the results as it was measured without stereo rectification and with stereo rectification. In some cases, the measured depth of the un-rectified method was double of that computed with rectified stereo images, at the same frame. The innovation that this algorithm introduce, is that it percept the rotational position of each camera in each frame by simple finding correspondences in images. The last is done by the neural network, which detects Sea-Vessels. This eliminates the need of precise marker positioning on the hull of the Ship, as well as precise positioning cameras in a parallel frontal plane to build a perfectly stereo rig.

Figure 5.20 In the left, the image from the left camera is presented and the detected Sea-Vessel is in green box. With red numbers the depth (0.56 meters) of the object from the left camera with the use of all the above steps is shown and with blue numbers at the bottom of the box the corresponding depth (0.88 meters) as it was measured without rectification. In the right the corresponding frame from the right camera. The two cameras was positioned 0.6 meters away from the monitor and in almost parallel configuration.

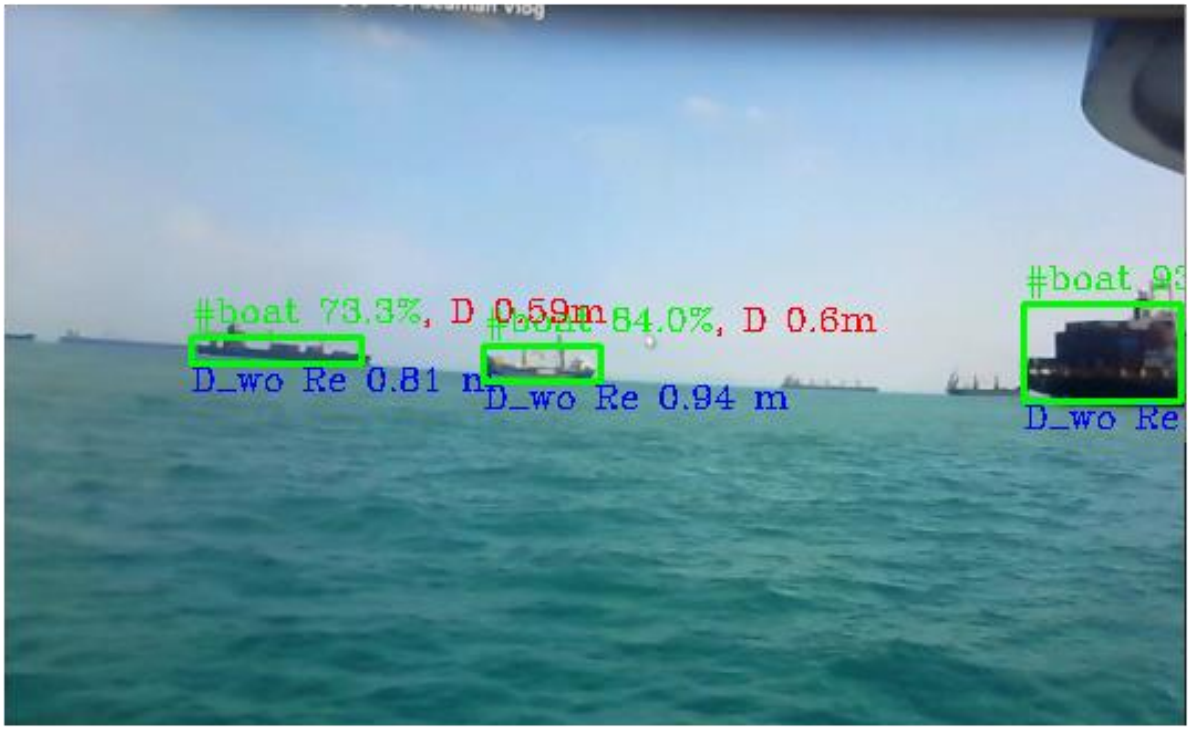

Figure 5.21 a) Image from the left camera . With red numbers are the depth estimated with rectification of the stereo images and with blue numbers at the bottom of each green box the depth measure without stereo rectification. As we can see the error in depth measuring without the rectification technique is not acceptable. The two cameras was positioned 0.6 meters away from the monitor and in almost parallel configuration.

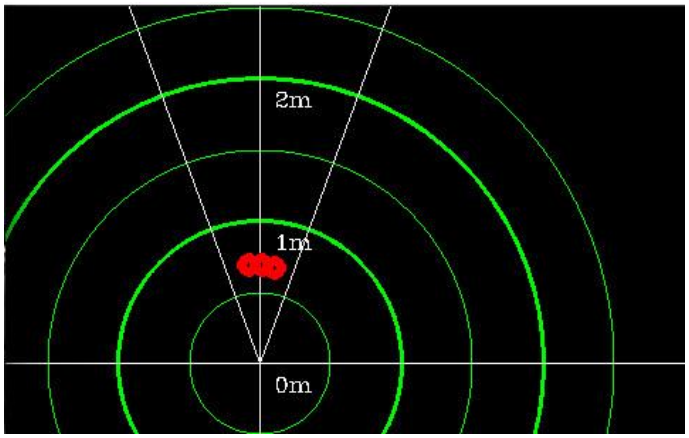

Figure 5.21 b) Presents the corresponding radar-like image of the estimated 3D position of the detected Sea-Vessels from our algorithm. This screen is part of our detection-estimation algorithm

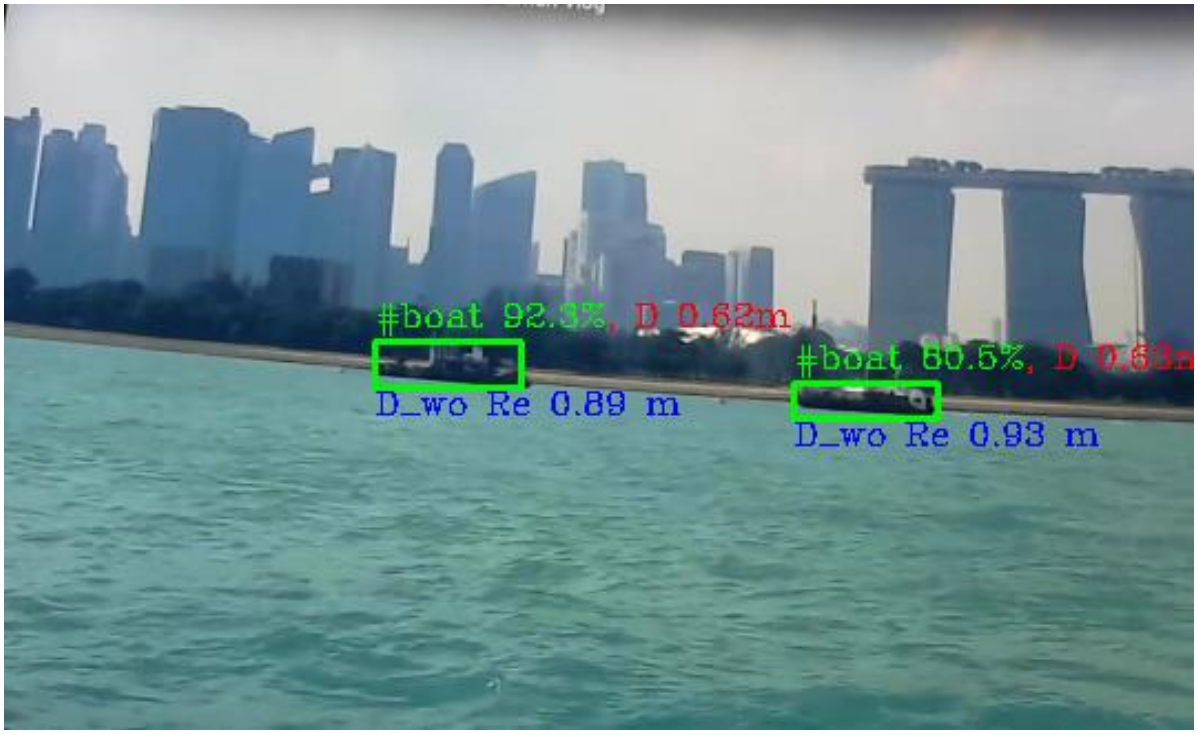

Figure 5.22 a) Image from the left camera . With red numbers are the depth estimated with rectification of the stereo images and with blue numbers at the bottom of each green box the depth measure without stereo rectification. As we can see the error in depth measuring without the rectification technique is not acceptable. The two cameras was positioned 0.6 meters away from the monitor and in almost parallel configuration.

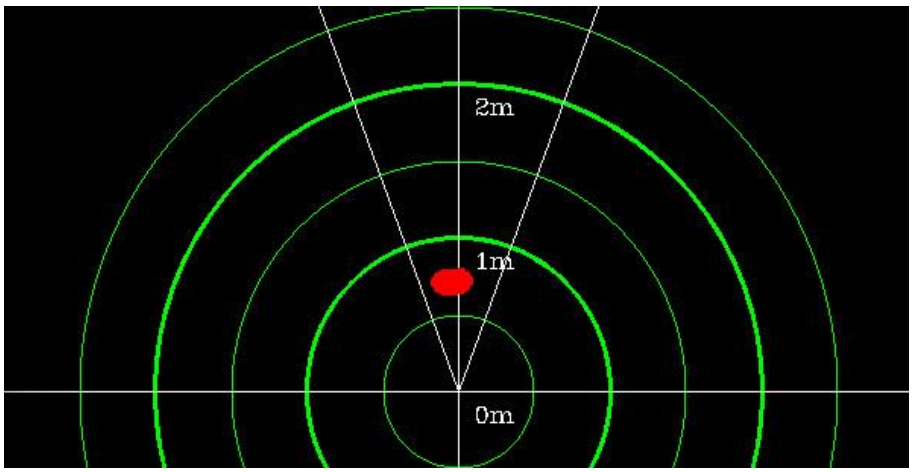

Figure 5.21 b) Presents the corresponding radar-like image of the estimated 3D position of the detected Sea-Vessels from our algorithm. This screen is part of our detection-estimation algorithm

Chapter 6

6 Experiments

This chapter describes the experimental procedure. Fist system requirements and configurations are described. Then the experimental procedure as well as the results are prevented. The aim of this chapter is to present the novel system architecture and the engineering techniques that were used to construct the system algorithm in order to minimize the overall run-time. Special care was given to optimize the data flow and the inference time, as it would be seen bellow.

6.1 Application Requirements

6.1.1 Neural Network Pre-Requirements

For training the neural network there is need of a host machine with enough computing power. We used as a host machine a desktop, which could have one of the following requirements installed:

- Windows 10 or Linux, preferred Ubuntu 16.04 or 18.04, preferred 16.04
- Danrknet, a C++ library for training Yolo
- CUDA kernels if GPU computation is preferred.
- OpenCV for image plots, it's not necessary.
- Tensorflow compatible with CUDA kernels

It is worth noticing that the Yolo network in order to be trained need a station with enough computing power and when a CUDA GPU exists, then the training procedure exploits the computational power of parallel processing that GPU provides.

6.1.2 Inference at Edge – Movidius Stick NCS2 Pre-requirements

For installing and running the Intel Movidus NCS stick in the base machine as well as in edge-Raspberry Pi4 the following packages are necessary:

- Windows 10 or Linux Ubuntu 16.04 or 18.04, preferred 16.04
- Intel OpenVino at least R1, it's an Intel library for hardware optimization
- OpenCV 4

6.2 System Configuration

By using the Intel Movidius NCS2 stick, a sequence of steps are needed in order to be ready for an inference. In appendix these steps are describing in more detail. Here we will use a diagram in order to depict quickly those steps (figure 6.1).

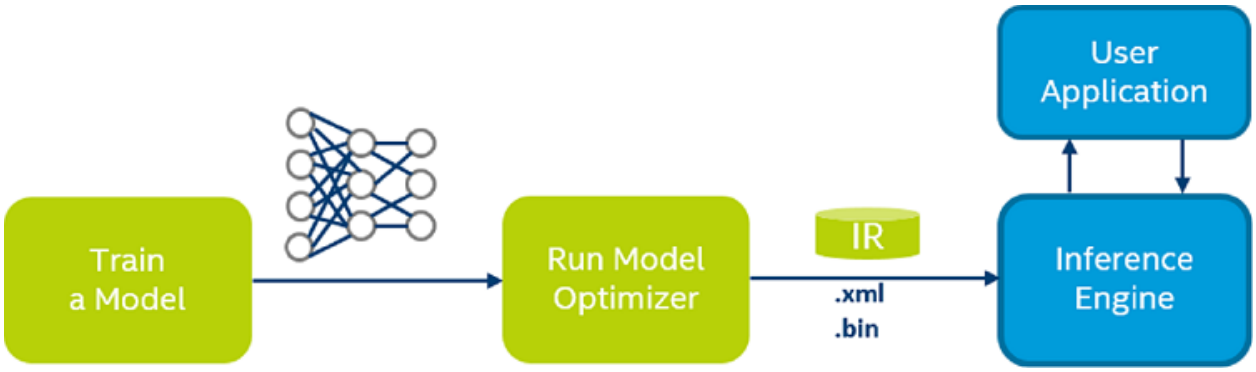

Figure 6.1 The main steps for prepairing a neural network to run in Movidius NCS2 are presented

In an embedded system with constraint power resources, like Raspberry Pi the OpenVino library doesn't include the Model Optimizer Toolkit. So, a host-based machine is needed for the training, as described above.

After we trained the Yolov3 network in Darknet framework, on our host machine, we followed the instructions for converting Yolov3 implemented in tensorflow to a format compatible with the Intel NSC2 (see appendix). Then two files are generated, which are the model architecture in .xml format and a binary file containing the trained weights of the network. From this point one can use it for the desired task. We used it for our task of Sea-Vessels 3D position estimation.

Steps for deploying a neural network in Intel NSC2:

- Train the network in an external library, we used Darknet
- Convert Yolo model from Darknet to Tensorflow library

- Create the two files of the network for deploying in NCS2, by running OpenVINO model optimizer
- Download the two files to the embedded machine, in our case the Raspberry Pi4
- Load the two files of the network in the code and initialize the network in Intel NCS2
- Make an inference by providing an image

Steps for configuring the system in raspberry pi:

- Download the network files to the raspberry
- Connect the two cameras on the USB port
- Calibrate the cameras
- Run the application

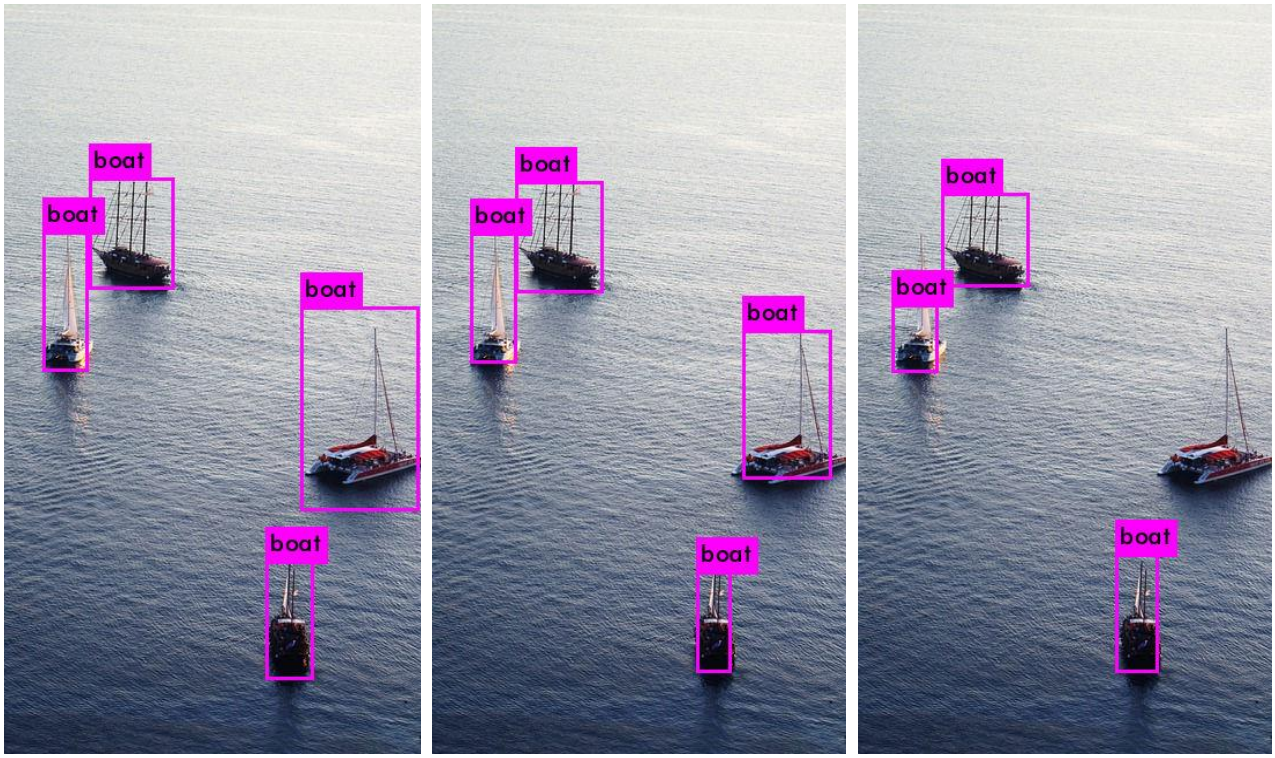

Figure 6.2 At left the detection results of the Yolov3 with input size at 608x608x3 tensor size. Middle the detection results of the Yolov3 with input size at 416x416x3 tensor size. Right the detection results of the Yolov3-Tiny implementation with input size at 416x416x3 tensor size. The encapsulation of fewer knowledge for Sea-Vessels from the Yolov3-Tiny model is clearly depicted from the lack of the 4th Ship detection. This is expected due to less neural layers of the tiny model.

6.3 Experiments

In this section, we will pay attention of the overall throughput of the algorithm and especially the accuracy of the 3D position estimation of the detected Sea-Vessels. In the previous chapters the robustness and the accuracy of each of the sub-modules was extensive analyzed and discussed. As a result we will not pay attention of each of the sub-modules.

Figure 6.2 illustrates 3 images after were inferred from the three different Yolov3 neural network implementations. There is an interesting observation. In the left and the middle we see the detection results from the Yolov3 architecture for 608x608x3 input tensor size and for 416x416x3 input tensor size corresponding. As we see there are not big differences between the output of those networks. The networks have correctly detected all the Sea-Vessels. This not the case for the Yolov3-Tiny for 416x416x3 input tensor size implementation at the right. As we can see it felts to detect all the objects. This is expected as it is much smaller compared with the Yolov3 and as a result encapsulates less knowledge about Sea-Vessel objects. On the other side, is much faster and can be used when the highest accuracy of detection is not first priority.

6.3.1 3D position accuracy

Apart from color information, one of the most crucial aspect of the pipeline is the accuracy of the 3D pose. Since this whole approach tries to solve an ill-posed problem by using additional information, coming from the detected object of the right camera, it is necessary to compare the accuracy of the pipeline. For this reason we compare of our system with a simple meter of 1mm accuracy. We run indoor tests by using a monitor showing frames with Sea-Vessels from different viewing angles and lighting conditions. The purpose was to estimate the 3D position and especially the distance-depth of the detected objects from the stereo cameras, which were viewing the videos on the monitor. The stereo cameras were moving in the space and placing in different positions corresponding to the monitor. After running the whole detection and position estimation algorithm in real time, the results were collected and are depicted in the following diagrams.

Figure 6.3 illustrates the two different Z-distance (depth) measurements of the stereo system from the monitor, as well as the real one with red color. The stereo system was placed successive to different distances from the monitor, starting from lower to bigger distance values. The distance range that we could use was from 0.5m to 2m approximately. With blue points the estimated distance of the stereo cameras from the monitor is depicted. As it is shown the results are steady and instant disturbances of the orientations of one camera compared to the other is eliminated, thought the process of image rectification. With green points the estimated distance of the stereo cameras from the monitor is depicted. We can observe that although the results are steady for the bigger amount of measurements, there is although an oscillation at the measured distance of 1.1m. This is because without rectification the results are sensitive to instant camera orientations disturbances.

Figure 6.4 illustrates the error in estimating the distance of each method. With blue color the error between the estimated distance with stereo rectification and the real one is presented.

With green color the error between the estimated distance without stereo rectification and the real one is presented. Finally with red color the difference between the estimated distance of the two methods, with and without rectification is depicted. We can clearly observe that the best depth accuracy comes from the method with stereo rectification, which achieves a maximum distance -depth error of +- 8cm. At the same time (green color) the sensitivity of the method without stereo rectification is clearly shown, which achieves a maximum error of 0.3m. This is because instant disturbances in camera's orientation to each other change the stereo configuration and this change affect the measured distance-depth. At the end a comparison of the two estimated methods, with and without stereo rectification is depicted with red color. As a conclusion the superiority of the method with stereo rectification against the one without is clearly illustrated.

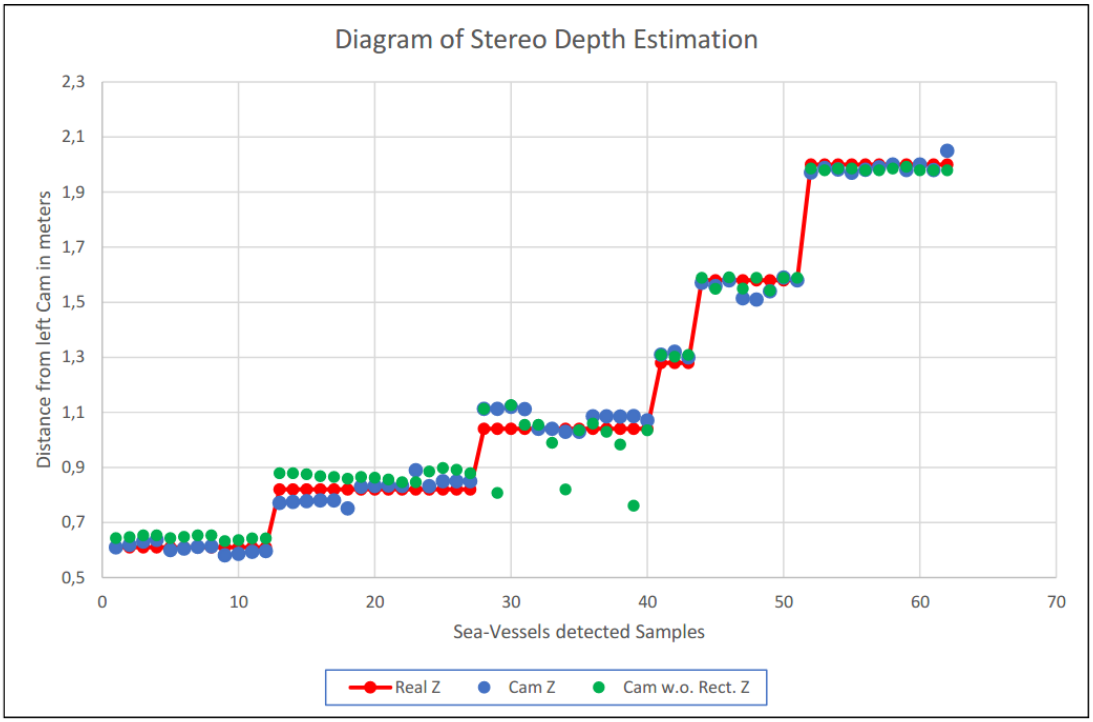

Figure 6.3 Depth distance between left cameras coordinate frame of the stereo system and real one as estimated from the meter. With red color the reference Z-distance-depth is depicted and with blue points the depth estimated from the stereo system by using stereo rectification. With green the points of the Z-distance from the left camera of the stereo system, as it was estimated without stereo rectification. As we can see the two estimated distances of the detected Sea-Vessels are pretty close to each and to the real one, but with more oscillations as far as is concerned the estimation method without image rectification.

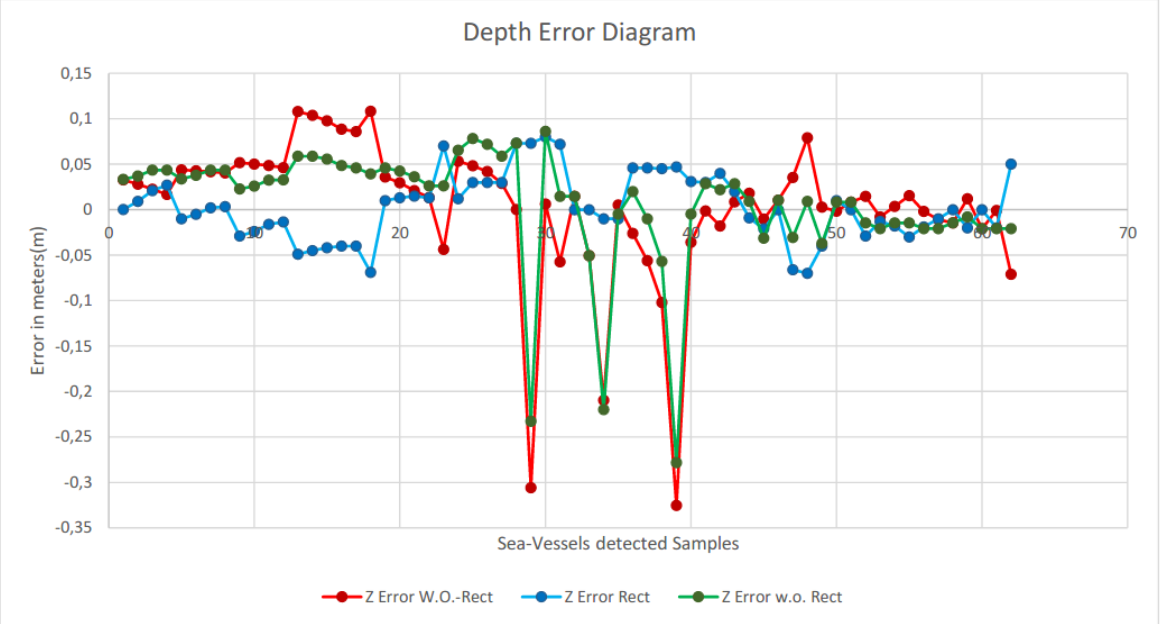

Figure 6.4. In this diagram the error in estimating the distance of each method is depicted. With blue color the error between the estimated distance with stereo rectification and the real one is presented. With green color the error between the estimated distance without stereo rectification and the real one is presented. Finally with red color the difference between the estimated distance of the two methods, with and without rectification is depicted. We can clearly observe that the best depth accuracy comes from the method with stereo rectification, which achieves a maximum distance -depth error of +- 8cm.

In Fig. 6.5,6.5 two diagrams of the estimated distance between the real one and the estimated distance from the stereo system is depicted. The x-axis represents the depth in meters of the detected Sea-Vessels from the stereo cameras and in y-axis the measured distance from the meter. The points correspond to the points that were presented in the previous diagrams. The perfect measurements of the depth-distance of the detected Sea-Vessels would be them which follow the line y=x that depicted with red color. As we see from the figure 6.5 compared with the figure 6.6, the estimated distance of the points in figure 6.5 have smaller error and thus smaller distribution around the line y=x of the perfect estimations. On the other hand, without using stereo rectification the results are more vulnerable to external disturbances in rotational position of the stereo cameras as is presented in figure 6.6.

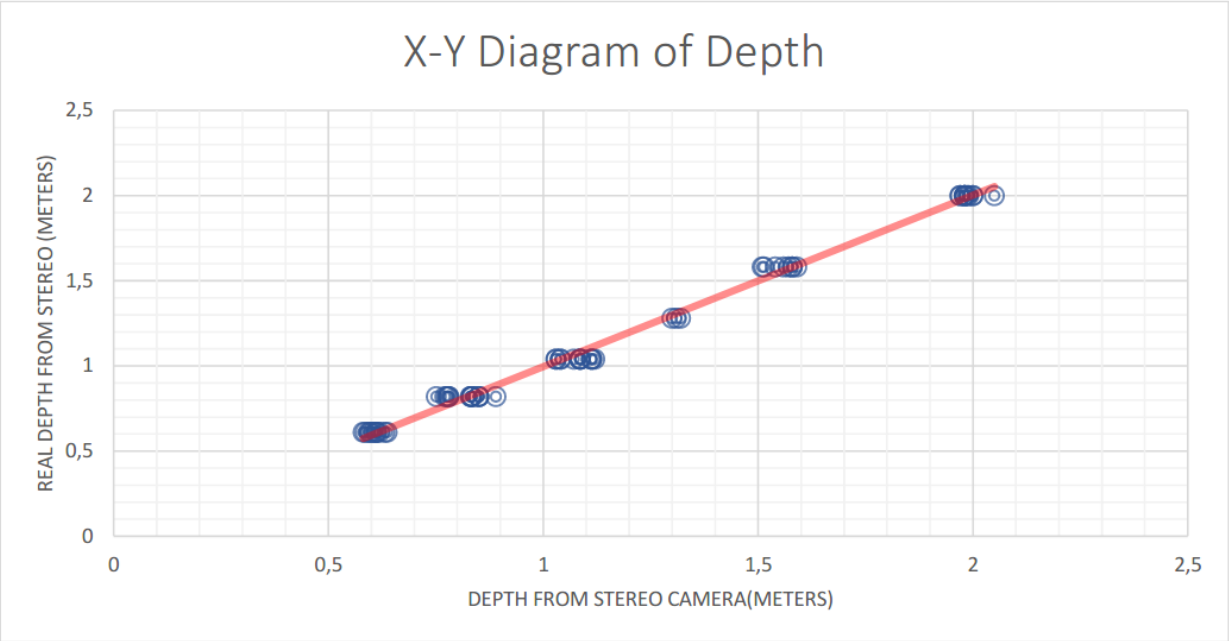

Figure 6.5 The estimated depth-distance of the detected Sea-Vessels from the stereo cameras computed with the stereo rectification method are depicted in X-Axis. In Y-Axis the depth of the Sea-Vessels from the stereo cameras as measured with the meter. With red line the perfect estimations are depicted.

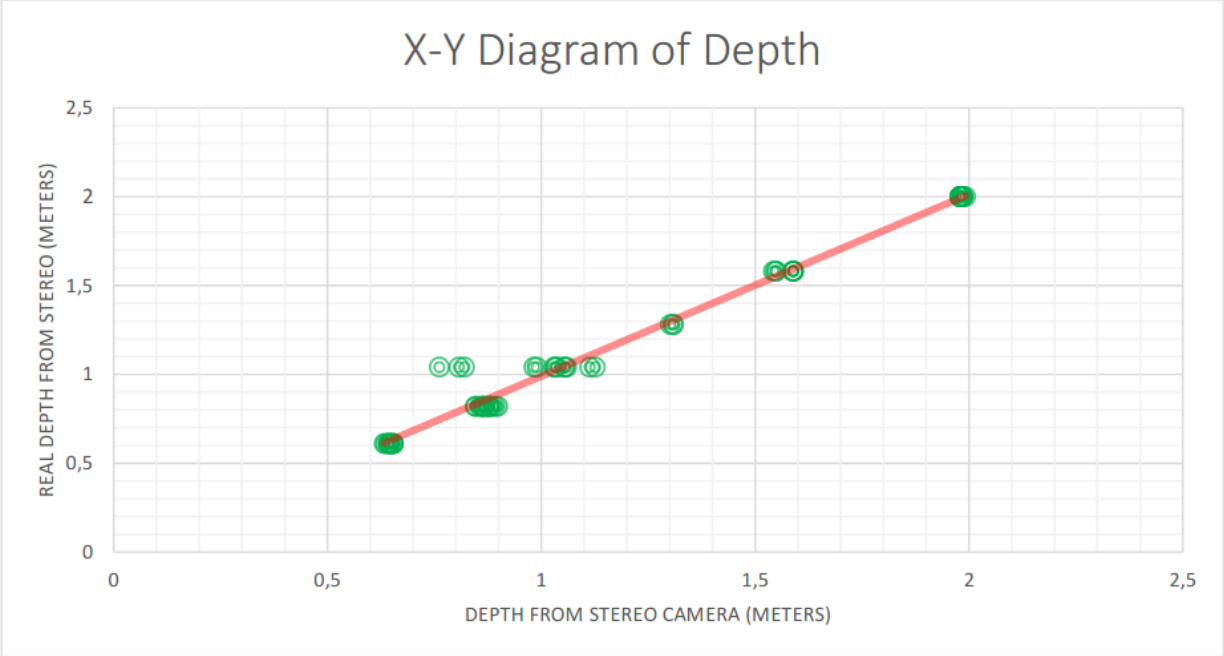

Figure 6.6 The estimated depth-distance of the detected Sea-Vessels from the stereo cameras computed without the stereo rectification method are depicted in X-Axis. In Y-Axis the depth of the Sea-Vessels from the stereo cameras as measured with the meter. With red line the perfect estimations are depicted. For 1.1m distance of the stereo cameras we observe the bigger error that is introduced to the measurements, compared with the results from the previous figure.

6.3.2 Processing Time and Delays of the Pipeline

One of the crucial parts during designing and developing the overall system architecture was the execution time from a single image shot to 3D position estimation results. For this reason our embedded system comprises a specific external processing unit for the execution of the neural network as described in chapter 2. After measurements in each sub-module time execution, found that the USB cameras that were used introduced an increased time delay compared with the other sub-modules. Both cameras introduced a time delay of 160ms-200ms. The right camera frame was always 150ms approximately back from the left camera frame. It was surprisingly discovered that the execution of the neural network on the NSC2 was extremely fast compared with the two cameras, 10ms maximum for tiny Yolov3, causing the overall system pipeline operating at a maximum of 6 FPS! when there were not Sea-Vessels detected. When there were Sea-Vessels detected the rest of the sub-modules were introduced all-together approximately 100ms-300ms causing a minimum overall execution time of the system reaching 2FPS. One of the sub-modules with greater processing time demand was the Horizon-Line detection pipeline which execution time varies from 10ms to 150ms relative with the ROI (Region Of Interest). Next the sub-module with hight execution time demands was the correspondences matching sub-module between the two image frames, needing approximately 10-35ms for each Sea-Vessel, depending on the Sea-Vessel detected size. The conclusion of

these results is that hardware which provides parallel execution like GPUs and TPUs are beneficial for image processing because it is a inherent parallel computational problem.

Finally, our system at the fastest implementation, using the YoloV3-tiny implementation was capable of delivering 5FPS. For a real-time application with fast moving objects this is not sufficient, but for maritime industry, where the speeds are far less lower compared with automobile industry for example, this system could fit in very efficiently (Figure 6.7).

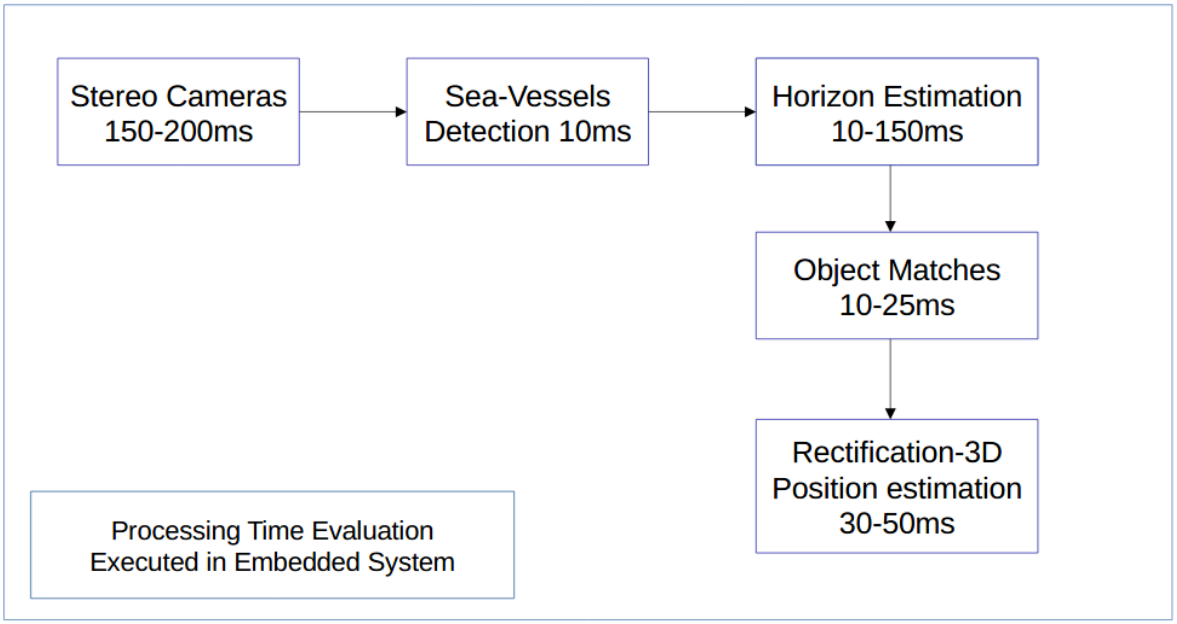

Figure 6.7: Data flow and time delays of the sub-modules of the system pipeline.

Chapter 7

7 Conclusions and Future Work

7.1 Thesis Contributions

According to Section 1.2, two major engineering challenges hinder the development process of a Sea-Vessel detection system capable of estimating the 3D position of each Vessel using stereoscopic images. First, it is challenging to build a compact embedded system which will detect and localize Sea-Vessels in real time without the need of huge computational power, consumption power and cost effective. Second, it is unclear what kind of detection algorithm must be used in order to achieve the best results in detection accuracy and execution time in a conventional embedded system. And finally, although the current huge evolution of hardware technology there is still a huge research in developing faster and more efficient hardware for big data processing.

The work accomplished in this thesis has (i) demonstrated the feasibility of developing a stereoscopic detection and localization system of Sea-Vessels, (ii) proposed and implemented an effective detection pipeline and (iii) demonstrated the accuracy of detecting and localizing Sea-Vessels, making the system ideal for autonomous Shipping and early collision awareness system for maritime industry. These contributions can be further elaborated as follows:

- Hardware design

  Setting up a perception pipeline on a real-time system involves a coherent and clever interplay between software and hardware. The embedded hardware architecture comprises of a tensor processing unit (Intel NSC2) for accelerating inference of our neural network model, connected to a Raspberry Pi4, an ARM based microcomputer and with two common usb-cameras comprises our stereo based vision system. These selections based on the trade-off between computational power, consumption power and low-cost.

- Sea-Vessels detection pipeline

  From early developing stages of the detection pipeline it was chosen to be used a stereoscopic system based on deep learning for Sea-Vessel detections. For this reason the Yolov3 model architecture was used and retrained – fine tuned suitably for our task. A K-means clustering system for

setting up correctly the bounding box anchors of our model was created[20][23]. Three different types of the neural network model architecture were developed and evaluated, with the results being more than satisfactory. One run of the smallest neural-network was accomplished in synchronous mode in 10ms maximum time in the neural compute stick (Intel NCS2). For delivering fastest results, in the code we used asynchronous execution of the detection pipeline from the main thread of the application, since the inference of the model was executed on the Intel NSC2. With this technique there was no lag in the main application thread for waiting the external module to reply.

- Fast Horizon-Line detection pipeline

  In order to eliminate false detections of the neural network, a fast horizon line detection algorithm was developed, based in computer vision techniques. Standard engineering techniques were used in order to boost the accuracy and the time execution of the horizon-line detection pipeline, since it was running in the CPU of the embedded system. A combination of detected boxes information from the previous stage of neural network and the color information of the two regions at the boundary of sea surface was used in order to detect the horizon line. Again special effort was given in order to combine all these informations and to provide accurate and robust horizon-line estimation result.

- 3D position estimation of the detected Sea-Vessels

  Exploiting the information coming from the second camera and stereo geometry, a novel position estimation technique was developed. This technique is capable of detecting and correcting instant small rotational disturbances of the right camera coordinate frame corresponding to the left, by using RANSAC algorithm and the detected Sea-Vessels from the previous stages. The robustness and accuracy of the algorithm were proofed by several experiments.

7.2 Future Work

Before our perception and Sea-Vessels localization system can be ready for general used in Maritime industry, several hardware and software improvements are necessary. First the current camera sensors that were used are very slow and have small resolution, causing the system running slow. More importantly, it is necessary to reduce the time from capturing a frame till this frame is transferred to the main application thread. Second, as were proofed the neural compute module is extremely fast compared with the CPU throughput of analyzing

images, for this reason it is proposed the fast horizon-line detection pipeline to be implemented with a neural network. By this way the overall execution time it is believed that will decrease, since the CPU needs more time to process the images. Third for a better time execution results it is suggested the whole software system architecture to be written in C++, since the python that were used for prototyping is an interpreted language and far more slowly from the C++.

Appendix A

A.1 Pre-Requirements for deploying Yolov3 based model on Intel NCS2

In this section the steps for deploying Yolov3 neural network model to the Intel Neural Compute Stick2 are presented. We made this procedure on a Linux host machine, so the steps correspond to a Linux host machine configuration.

Machine host pre-configuration:

- Ubuntu Linux 16.04
- CUDA GPU kernels 10.1
- DarkNet Framework for training/testing the Network
- TensorFlow versions between 1.11.0 and 1.13.0
- Intel OpenVino 2019_R1.1

After properly configuration of the host machine, one has to follow the bellow describing steps for deploying Yolov3 model on the Intel NCS2. It is worth noting that there is no need to have a configured machine precisely as follow, because new versions of each driver/library are coming. The above steps configurations were used for these thesis.

A.2 Configuration of the Intel NCS2

Configurations for deploying the Yolov3 model on Intel NSC2:

- Convert Yolov3 model from DarkNet framework to a TensorFlow representation
- Within TensorFlow create a frozen model file of .pb format
- Navigate to openvino install directory and run model optimizer for creating the two necessary files of the network to be run on Intel NCS2

The above described steps need some explanation. After we finished with the training and testing of our models on DarkNet framework, we had to use TensorFlow framework in order to run our model on Intel Movidius NCS2. This step is needed because OpenVino library which interfaces with the Intel NCS2, it is not supporting DarkNet format based networks. On the

other hand TensorFlow provides a compatibility with the DarkNet framework and the trained weights of the Yolov3 model could be inserted in TensorFlow framework. Additionally OpenVino library is compatible with TensoFlow implemented networks. This helps because TensorFlow provides more functionalities as the DarkNet framework.

Within TensorFlow we save our model, creating a frozen .pb file. A .pb file is a model representation of the Yolov3 in TensorFlow format. It comprises all the information of the neural network (topology, activation functions, weights). This format of file is acceptable by the model optimizer of the OpenVino library, which will convert-optimize the model appropriate for running on the Neural Compute Stick 2. The model optimizer should be provided with the .pd file of the Yolov3 model and then it creates two files, one .xml file which contains the graph topology of model and a binary file of the trained weights of each neuron in the neural network. Those two files can then be loaded in the neural compute stick by using the appropriate classes of the OpenVino framework.

Finally we provide a link from the intel's official website which explains the converting procedure and additionally provides the necessary statements

Bibliography

[1] Aayush Grover, Shashi Kumar, Anil Kumar, Ship Detection Using Sentinel-1 SAR Data, ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume IV-5, 2018

[2] Adrien Bartoli CNRS – LASMEA, Søren I. Olsen DIKU: Lecture 16 – Camera Motion From Two-View Relationships (Section 9.6)

[3] Ankit Dhal: Real-time 3D Pose Estimation with a Monocular Camera Using Deep Learning and Object Priors On an Autonomous Race car, arXiv 1809.10548v1 ,27 Sep 2018

[4] Beenish Zia, Ramesh Illikkal and Bob Rogers, Use Transfer Learning for Efficient Deep Learning Training on Intel® Xeon® Processors, White Paper

[5] Benjamin RM, Curcio J, Leonard JJ, Newman PM (2006). A Method for Protocol-Based Collision Avoidance Between Autonomous Marine Surface Craft. J. Field Robot.23(5):333-346.

[6] Chi Yoon Jeong , Hyun S Yang and KyeongDeok Moon: Fast horizon detection in maritime images using region-of-interest, International Journal of Distributed Sensor Networks 2018, Vol. 14(7)

[7] Christoph Alexander Thieme, Ingrid Bouwer Utne, Stein Haugen: Assessing ship risk model applicability to Marine Autonomous Surface Ships, Ocean Engineering 165 (2018) 140–154

[8] COCO, cocodataset.org

[9] Cosimo Rubino, Student Member, IEEE, Marco Crocco , Member, IEEE, and Alessio Del Bue , Member, IEEE:"3D Object Localisation from Multi-View Image Detections" IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 40, No. 6, june 2018

[10] David A. Forsyth, Jean Ponce: Computer Vision a Modern Approach, Second Edition, Pearson

[11] Domenico D. Bloisi, Fabio Previtali, Andrea Pennisi, Daniele Nardi, Michele Fiorini, Senior Member, IEEE: Enhancing Automatic Maritime Surveillance Systems with Visual Information, IEEE Transactions on Intelligent Transportation Systems

[12] Duane C. Brown: Close-Range Camera Calibration, DBA Systems, Inc. Melbourne, Flu. 32901, 1971

[13] François Chollet: Deep Learning with Python. 2018 by Manning Publications Co

[14] Gary Bradski and Adrian Kaehler: Learning OpenCV, O'REILLY

[15] Gazi Kocak, Shigehiro Yamamoto and Takeshi Hashimoto: Detection and tracking of ships using a stereo vision system, Scientific Research and Essays Vol. 8(7), pp. 288-303, 18 February, 2013 http://www.academicjournals.org/SRE

[16] Gazi Kocak, Shigehiro Yamamoto, Takeshi Hashimoto: Analyzing Influence of Ship Movements on Stereo Camera System Set-up on Board Ship, Journal of the JIME Vol. 47, No. 6(2012)

[17] Intel Neural Toolkit, OpenVINO,

[18] Intel® Neural Compute Stick 2,

[19] John Canny Member IEEE: ”A Computational Approach to Edge Detection”, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. Pami-8, No. 6, November 1986

[20] Joseph Redmon, Ali Farhadi, University of Washington: Yolov3: An incremental Improvement, arXiv 2018 - https://pjreddie.com/darknet/yolo/

[21] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, University of Washington, Allen Institute for AI, Facebook AI Research: You Only Look Once: Unified, Real-Time Object Detection, CVPR 2016, http://pjreddie.com/yolo/

[22] Kenji Hata and Silvio Savarese: CS231A Course Notes 1: Camera Models

[23] Lloyd, Stuart P. "Least squares quantization in PCM." Information Theory, IEEE Transactions on 28.2 (1982): 129-137

[24] Lopez-Molina C, Baets BD, Bustince H, et al. Multiscale edge detection based on Gaussian smoothing and edge tracking. Knowl-Based Sys 2013; 44(Suppl. C): 101–111.

[25] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. “The pascal visual object classes challenge: A retrospective.” International Journal of Computer Vision, 111(1):98–136, Jan. 2015.

[26] Martins A, Almeida JM, Ferreira H, Silva H, Dias N, Dias A, Almeida C, Silva EP (2007). Autonomous surface vehicle docking maneuver with visual information. Proc. IEEE Int. Conf.Robot.Autom., Roma, Italy.

[27] OpenCV opencv.org/

[28] Paolo Di Febbo, Carlo Dal Mutto, Kinh Tieu, Stefano Mattoccia Aquifi Inc. University of Bologna: KCNN: Extremely-Efficient Hardware Keypoint Detection with a Compact Convolutional Neural Network, CVPR 2018

[29] Perera LP, Carvalho JP, Guedes Soares C (2011). Fuzzy logic based decision making system for collision avoidance of ocean navigation under critical collision conditions. J.Mar. Sci. Technol. 16:84-89.

[30] Prasad DK, Rajan D, Rachmawati L, et al. MuSCoWERT: multi-scale consistence of weighted edge radon transforms for horizon detection in maritime images. J Opt Soc Am A 2016; 33: 2491.

[31] Remote and Autonomous Ship, Ship Intelligence Marine, Rolls-Royce Aawa White Paper

[32] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: “Towards real-time object detection with region proposal networks”. arXiv preprint arXiv:1506.01497, 2015.

[33] Shimpo M, Hirasawa M, Oshima M (2005). Detection and tracking method of moving ships through navigational image sequence. J. Japan Inst. Navigation 113:115-126.

[34] Shimpo M, Shu R, Yamamoto S (2008). Investigation of Image Processing Methods Capable of Supporting Navigational Lookout. Proceedings of Asia Navigation Conference.

[35] Sinno Jialin Pan and Qiang Yang Fellow: A Survey on Transfer Learning, (Volume: 22 , , Oct. 2010)

[36] Statheros T, Howells G, McDonald-Maier K (2008). Autonomous Ship Collision Avoidance Navigation Concepts, Technologies and Techniques. J. Navigation61:129-142.

[37] TensorFlow, www.tensorflow.org

[38] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie: Feature Pyramid Networks for Object Detection, arXiv 1612.03144v2 [cs.CV] 19 Apr 2017

[39] World Ocean Review, “Global shipping - a dynamic market.” [Online]. Available: worldoceanreview.com/en/wor-1/transport/global-shipping

[40] Yan Yan, Bok-Suk Shin, Xiaozhengmou, Wei Mou, Han Wang, School Of Electrical And Electronic Engineering, Nanyang Technological University, Singapore: Efficient Horizon Detection on Complex Sea for Sea Surveillance, International Journal Of Electrical, Electronics And Data Communication, ISSN: 2320-2084 Volume-3, Issue-12, Dec.-2015

[41] Zhengyou Zhang, Senior Member, IEEE, A Flexible New Technique for C, 1971amera Calibration, IEEE Transactions On Pattern Analysis And Machine Intelligence, Vol. 22, No. 11, November 2000