TALLINN UNIVERSITY OF TECHNOLOGY
School of Engineering
Department of Electrical Power Engineering and Mechatronics

Tanel Treuberg 193761EAAB

Improving situational awareness of autonomous vessels using computer vision

Bachelor's Thesis

Supervisor: Heigo Mõlder
PhD

Co-Supervisor: Karl Janson
PhD

Tallinn 2023

TALLINNA TEHNIKAÜLIKOOL
Inseneriteaduskond
Elektroenergeetika ja mehhatroonika instituut

Tanel Treuberg 193761EAAB

Autonoomse laeva situatsiooniteadlikkuse arendamine masinnägemise abil

Bakalaureusetöö

Juhendaja: Heigo Mõlder
PhD

Kaasjuhendaja: Karl Janson
PhD

Tallinn 2023

Author's declaration of originality

I hereby certify that I am the sole author of this thesis and that this thesis has not been presented for examination or submitted for defense anywhere else. All used materials, references to the literature, and work of others have been cited.

Author: Tanel Treuberg

2023

THESIS ASSIGNMENT

Thesis topic:	**Improving situational awareness of autonomous vessels using computer vision**
Thesis topic in Estonian:	**Autonoomse laeva situatsiooniteadlikkuse arendamine masinnägemise abil**
Student:	**Tanel Treuberg, 193761EAAB**
Specialty:	**Electrical Power Engineering and Mechatronics**
Type of thesis:	**Bachelor's thesis**
Thesis supervisor:	**Heigo Mõlder, PhD**
Thesis co-supervisor:	**Karl Janson, PhD**
(organisation, position, contact)	**TalTech Department of Computer Systems, e-mail: karl.janson@taltech.ee**
Assignment validity period (set by the supervisor):	2022/2023 Spring
Thesis submission deadline:	**18.05.23**

Student (signature)

Supervisor (signature)

Programme director (signature)

Co-supervisor (signature)

1. Rationale

Within a joint project between the TalTech Department of Electrical Power Engineering and Mechatronics, the TalTech Centre of Excellence in Small-Craft Building, and the maritime company Baltic Workboats, the aim is to automate vessel sea trials that have until now been carried out by various captains with manual control. The results of the trials are therefore not objective; they depend on how each captain handles the vessel, and deviations can appear in the data on whose basis it is not possible to confirm the vessel's parameters with the desired accuracy. Conducting the trials is also very time-consuming, since a particular manoeuvre often has to be repeated several times because in some runs the vessel is steered with too large a radius or at the wrong speed – human error introduces unwelcome deviations in the trial results.

Within the cooperation project, these sea trials are planned to be performed with the help of a computer program, in which the vessel's situational awareness plays a key role. This thesis aims to develop the vessel's situational awareness by building a computer-vision-based solution for perceiving the surroundings.

2. Aim of the work

The aim of the thesis is to develop a computer-vision system suitable for the automated conduct of vessel sea trials. This means selecting suitable hardware and creating the software for the computer-vision system, with intermediate testing in the harbour and finally on a real vessel. It also includes investigating how the software can be optimised so as to make the most efficient use of the hardware, and as a result make the software's processing smoother and faster.

3. List of questions to be addressed

- Which control unit should be chosen for the computer-vision system?
- Which camera system should be chosen for computer vision?
- Which software solution should be used to solve the problem?
- How should information about objects detected by the computer-vision system be passed to the vessel's controllers?

- Which methods can be applied to optimise the software?
- Which neural network should be used for object detection, and why?

4. Source data

Source information is gathered from the following sources:

- Existing solutions previously built in the world will be studied.
- Device data sheets and schematics.
- Source information provided by the project team regarding the vessel's control system.
- Specialist literature, websites, scientific articles, etc.

5. Research methods

The theoretical part of the work is based on an analysis of sources (literature, web articles, scientific articles) and on the author's domain knowledge, together with consistency between them.

The practical part covers experiments, comparisons of the models used in computer vision and their capability, taking measurements and analysing them (both graphically and statistically), so that the results obtained enable the most optimal solution to the problem posed.

6. Graphical part

Charts, tables, and figures are predominantly in the main body of the work.

7. Structure of the work

(Minor changes to subtopics in the theoretical part are possible.)

- Thesis assignment
- Table of contents
- Foreword
- Glossary of terms, abbreviations, and symbols

- Introduction
    1. Problem statement
    2. Overview of solutions built around the world so far
    3. Project overview
- Hardware selection, with comparisons
    1. Control unit
    2. Cameras
    3. Control-tower assembly drawing / overall diagram
    4. Electrical diagram
- Computer vision and selection of computer-vision software, with comparisons
    1. Computer-vision software platforms – which one, suitable for what
    2. Object detection – what, how, and why
- Combining computer vision with other sensors for more precise information
    1. With radar
    2. With lidar
- Experiments
    1. Video-based experiments in a simulated environment to test computer vision
    2. Experiments in the harbour, control-tower test
    3. Experiments on the vessel
    4. Analysis of experimental results
- Conclusions
- Summary
- Future development work
    1. Integration with the AIS (Automatic Identification System)
- List of references
- Appendices

8. References used

Source material consists of specialist literature, maritime-themed articles, and – with Baltic Workboats' permission – information on the manoeuvring capabilities of the vessels under test.

- Nvidia Jetson AGX Xavier Developer Kit, https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit
- Marine Insight, https://www.marineinsight.com/
- OpenCV, https://opencv.org/
- TensorFlow, https://www.tensorflow.org/
- Baltic Workboats, https://bwb.ee/
- MindChip, https://mindchip.ee/

9. Thesis consultants

Possible consultation with Baltic Workboats' project manager / client / representative regarding installation of the computer-vision system on the prototype vessel and setting the required input parameters. Also for specifying parameters (turning radius, permitted heel, maximum speed, dimensions from bow to stern and from port to starboard).

10. Stages and schedule

- Theoretical part (11 March)
  - Browsing/collecting sources
  - Building chapter content
- Practical part (2 April)
  - Computer-vision model
    - Model selection
    - Implementation
    - Training
  - Experiments
    - Indoors
    - Harbour
    - On the vessel
    - Developing/refining model parameters
- Thesis draft version (23 April)
  - With the theoretical part
  - With the practical part

  - Submission to the supervisor (24 April)
  - Making revisions and additions (26 April)
- Thesis completed (8 May)

Conditions for a closed defence and/or restrictions on the publication of the thesis are formulated on the reverse.

Abstract

The aim of the thesis is to develop a computer vision system suitable for automated vessel maneuvering tests. This involves selecting appropriate hardware and computer vision model, developing software for the computer vision system, conducting system tests, and analysing the results. The analysis compares the accuracy and speed of different computer vision models. Additionally, research is conducted to determine ways to optimize the software in order to efficiently utilize the hardware and thereby make the system run smoother and faster.

Keywords: computer vision, machine vision, embedded system, optimisation, conteinerisation, virtual environment, neural network, machine learning, hardware acceleration.

The thesis is in English and contains 40 pages of text, 8 chapters, 24 figures, 4 tables.

Annotatsioon

Autonoomse laeva situatsiooniteadlikkuse arendamine masinnägemise abil

Lõputöö eesmärgiks on arendada välja masinnägemise süsteem, mis sobib laeva käigukatsete automatiseeritud läbiviimiseks. See eeldab sobiva riistvara ja masinnägemise mudeli valikut, masinnägemise süsteemi tarkvara loomist, süsteemi katsetusi ning tulemuste analüüsi. Analüüsis võrreldakse erinevate mudelite täpsust ja kiirust. Samuti uuritakse välja, kuidas ning milliste meetoditega on võimalik optimeerida tarkvara nii, et võimalikult efektiivselt utiliseerida riistvara ning selle tulemusena ka tarkvara tööd sujuvamaks ning kiiremaks muuta.

Märksõnad: masinnägemine, tehisintellekt, sardsüsteem, manussüsteem, optimeerimine, konteinerdamine, virtuaalkeskkond, närvivõrk, masinõpe, riistvara kiirendus.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 40 leheküljel, 8 peatükki, 24 joonist, 4 tabelit.

Foreword

The topic of this thesis was initiated together with the supervisors, Heigo Mõlder and Karl Janson. The system described in the thesis and built during its course is used in the TalTech and Baltic Workboats joint project "Development of pre-configured autonomous open-water sea-trial technology for vessels".

I would like to thank my supervisor Heigo Mõlder for his help, for the thesis topic, and for his supervision; my co-supervisor Karl Janson for assisting with the technical solution; and my consultant Uljana Reinsalu for her help on the neural-network topics.

List of abbreviations and terms

AIS Automatic Identification System

ARM Advanced RISC Machines – a reduced-instruction-set computer architecture developed by Arm Holdings

CNN convolutional neural network

COCO Common Objects in Context – an annotated image dataset

CPU Central Processing Unit

CSI Camera Serial Interface

CUDA Compute Unified Device Architecture

DL Deep Learning

FPS Frames Per Second

FSD Full Self Driving – Tesla's full-autonomy system for its vehicles

GB GigaByte

GFLOPs billions of Floating-point Operations per second

GPU Graphics Processing Unit

Gbps Gigabits per second

HAVS Hand-Arm Vibration Syndrome

HMMA Half-precision Matrix Multiply and Accumulate – multiplication and summation of half-precision floating-point matrices

IMMA Integer Matrix Multiply and Accumulate – multiplication and summation of integer matrices

L4T Linux for Tegra – a Linux distribution for Nvidia Tegra embedded devices

MB megabyte

NVDEC Nvidia Video Decoder

NVDLA Nvidia Deep Learning Accelerator

ONNX Open Neural Network Exchange – an open format for exchanging neural-network models

PoE Power Over Ethernet

PVA Programmable Vision Accelerator

SBC single-board computer

SoM System on Module (or SoC, system on chip)

TFLOPs trillions of Floating-point Operations per second

TOPs trillions of Operations per second

USB Universal Serial Bus

YOLO You Only Look Once – a philosophy in which the neural network processes an image only once

eMMC embedded MultiMedia Card

mAP mean Average Precision

ms millisecond

px pixel

Table of contents

1 Introduction 19
2 Sea trials 22
3 Computer vision 24
3.1 Computer vision operating principle 24
3.2 Convolutional neural network 25
4 Hardware selection 29
4.1 Control unit / onboard computer 29
4.2 Tensor cores and CUDA 32
4.3 Cameras 32
4.4 Control tower 35
5 Software 38
5.1 Computer-vision neural network model 39
6 Computer-vision application 42
6.1 Working environment 42
6.2 Main program 44
6.3 Computer-vision model and software optimisation 48
7 Results 51
7.1 Conducting the experiments 51
7.2 Analysis of results 52
8 Summary 57
References 59
Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis 63

List of figures

Figure 1.1. Baltic Workboats' Navy 18 WP patrol vessels [12] 20

Figure 2.1. Sea trial designed for measuring the vessel's turning radius [13] 23

Figure 3.1. The image, represented as a matrix (left), is operated on with the filter (centre), producing a dot product as a feature map (right) 27

Figure 3.2. SoftMax-applied output layer in the neural network (purple) and probabilities for each class (right) [17] 28

Figure 4.1. The chosen computer: the Jetson AGX Xavier (right, without the heat-sink module) [26] 31

Figure 4.2. A Tensor Core multiplying and summing three-dimensional $4\times4\times4$ matrices [27] 32

Figure 4.3. Module with the Arducam Fisheye camera [34] 35

Figure 4.4. Simplified operating diagram of the control tower 36

Figure 4.5. The control tower's placement on the patrol vessel [35] 37

Figure 5.1. Architecture of the YOLOv5 models [54] 40

Figure 6.1. Simplified diagram of how containerised applications operate [56] 43

Figure 6.2. Versions of the software installed in the development-machine container, chosen to match the production-machine container state as closely as possible 43

Figure 6.3. Software versions in the production-system container 44

Figure 6.4. Simplified block diagram of the main program 45

Figure 6.5. Example of running the main program from the command line 45

Figure 6.6. Simplified block diagram of the computer-vision program's operation .. 46

Figure 6.7. Initialisation of architecture-specific parameters 47

Figure 6.8. Detections generated and processed by the neural network, displayed on a frame (bounding boxes and the detected classes inside them together with the probabilities) 48
Figure 6.9. Running YOLOv5l, with a single input at 640×640-pixel resolution, in unoptimised form does not utilise the whole GPU, as seen in the monitoring tool jtop [62] 49
Figure 6.10. Running the YOLOv5l model, with a single input at 640×640-pixel resolution, in optimised form utilises the whole GPU 50

Figure 7.1. Model performance with a single 640×640-pixel frame input 53
Figure 7.2. Model performance with a single 1280×1280-pixel frame input..... 54
Figure 7.3. Model performance with three 640×640-pixel frame inputs in parallel. 55
Figure 7.4. Model performance with three 1280×1280-pixel frame inputs in parallel 56

List of tables

Table 4.1. Parameters of the Jetson-series computers [24] 30
Table 4.2. Parameters of the Jetson AGX Xavier and the Xavier NX [24] 31
Table 4.3. Cameras matching the parameters 34
Table 5.1. Overview of the parameters of the most popular models 39

1 Introduction

Logistics plays an extremely important role in the optimal functioning of humanity as a whole, encompassing the transport of people and goods as well as the provision of services. It is therefore vital that this system operate practically flawlessly and without disruption, but disruptions are unfortunately unavoidable – the Ever Given container ship grounding incident in the Suez Canal [1] is one example. The cause is generally the human factor (inattention, inadequate situational assessment) and/or a technical failure, which can result in an accident.

Preparing vessels and seafarers for navigation is extremely resource-intensive, both financially and in terms of time. The behaviour of the sea is highly unpredictable in open waters, and the risk of drowning is high even under the best preparation; in 2019 drowning deaths ranked third in worldwide cause of death [2], and contributing factors at sea include incompetent preparation (of both the crew and the vessel), overcrowding aboard, excessive trust in the autopilot, and inattention or underestimation of situations.

Long periods at sea are also mentally taxing: there is no direct contact with loved ones, and most of the time is spent in social isolation, except where the crew consists of at least two members. There is also a greater risk of conditions such as hand-arm vibration syndrome (HAVS), cardiovascular disease, pneumonia (in severe cases progressing to lung cancer), general overstrain (caused by muscle tension, excess emotional stress, loneliness that is often suppressed via excessive alcohol consumption and/or smoking, fatigue, and/or insufficient physical activity), and chronic depression [3].

There are several solutions to avoid such problems, of which autonomous control systems using radar and computer vision are currently the most relevant and effective. In the automotive industry, several companies are pursuing full autonomy: Tesla (AutoPilot, FSD) [4], Lucid (DreamDrive) [5], Waymo [6], Nvidia (Nvidia Drive) [7], and Comma.ai (the cheapest semi-autonomous system available) [8].

In the maritime industry, fewer companies have pioneered autonomous systems: Kongsberg [9] and Rolls-Royce [10].

An autonomously controlled vessel offers far greater operational efficiency than a crewed one for several reasons: vessel mass is smaller (there is no need for a ventilation system, crew quarters, galley, bunks, sanitation system, or captain's cabin), so less material is required to build it, which in turn improves manoeuvrability. Because the crew is not directly on board, the need for docking is also reduced (only for maintenance/repair, cargo loading, and/or refuelling) and the energy cost of operating the vessel decreases. The vessel can also be sent to operate in harsher conditions without posing a direct risk to human health (and, in the best case, can save time by taking a more direct but more hazardous route).

Tallinn University of Technology and AS Baltic Workboats are partners in a project whose aim is to carry out vessel sea trials autonomously, i.e. without human involvement (see Figure 1.1) [11]. There are also plans for further development in the same field, to apply autonomous systems beyond sea trials and give vessels the ability to perform tasks and navigate at sea independently.

Figure 1.1. Baltic Workboats' Navy 18 WP patrol vessels [12]

The aim of this bachelor's thesis is to develop a computer vision system that assists with conducting vessel sea trials. This requires selecting suitable hardware, software, and a machine learning model, then creating the computer vision software and optimising the model.

The following questions are addressed in pursuit of this aim:

- Which hardware should be used for the computer vision system?
- Which machine learning model can best detect objects at sea?
- How can objects be detected on an autonomous vessel using computer vision?
- How can the machine learning model be optimised for resource-constrained hardware?

The thesis is organised as follows. The Sea trials chapter introduces the methodology of vessel sea trials and the solution for automating them. Chapter 1 discusses computer vision – its nature and operating principle – and explains, among the models used in computer vision, the convolutional neural network in particular. Chapter 2 covers the hardware used to build the computer vision system, hardware-level accelerators, and gives a simplified overview of the system. Chapter 3 surveys the different software stacks for building computer vision applications and determines which computer vision model is used in the application. Chapter 4 explains the nature and operating principle of the computer vision application, and describes its development process and optimisation. Chapter 5 carries out experiments with the computer vision models and analyses how much model optimisation affects frame-processing throughput in the application. Chapter 6 summarises the contents and the fulfilment of the thesis's objectives.

2 Sea trials

During shipbuilding, the vessel's parameters need to be determined to confirm that they meet the previously specified requirements. These parameters are also recorded in the vessel's handbook. To determine them, sea trials have been devised in which the vessel covers a designated route while manual measurements are taken; these measurements are then used to assess the vessel's capabilities and to inform the development of future vessels.

Sea trials are necessary for measurement accuracy and trial reproducibility. At the time of writing, measurements are taken with precise equipment, but the vessels are operated by different people, and the results are recorded manually – which introduces inaccuracies and does not guarantee consistent reproducibility of the trials.

Several problems arise with manually conducted sea trials:

1. Differences in measurement results, which depend on the vessel operator's reactions and psychological state.
2. Because the vessel is generally operated by a single person during the trials, that person must monitor several activities in parallel while collecting data, which in turn entails partial data loss over certain periods.
3. Entry of measurement data is not automated; preparing the protocol is labour-intensive and requires the cooperation of several people.
4. The vessel has no awareness of its surrounding environment (no capability to handle situations and avoid accidents on its own), so this task falls to the captain, whose inattention may create critical situations or cause accidents.

It is therefore reasonable to automate the sea trials. To do so, the vessel would need to be equipped with autonomy-enabling features that allow it to perform the trials on its own and to log data during them.

An autonomous system issues specific navigation commands to the vessel and records data

over a designated period. Precise results emerge from the collected material. For example, a vessel moving in waves has a constantly varying speed, so a measured speed is directly tied to the moment of recording. By analysing the data from a chosen recording period, the speed characteristics of the vessel during the trial can be determined. Pre-programmed sea trials help avoid operator-to-operator variability – for example, the rate at which the vessel turns and the time taken to do so depend on the operator's intuition and other personal traits, which in turn hinders an objective assessment of the vessel's maximum capability [13] (see Figure 2.1).

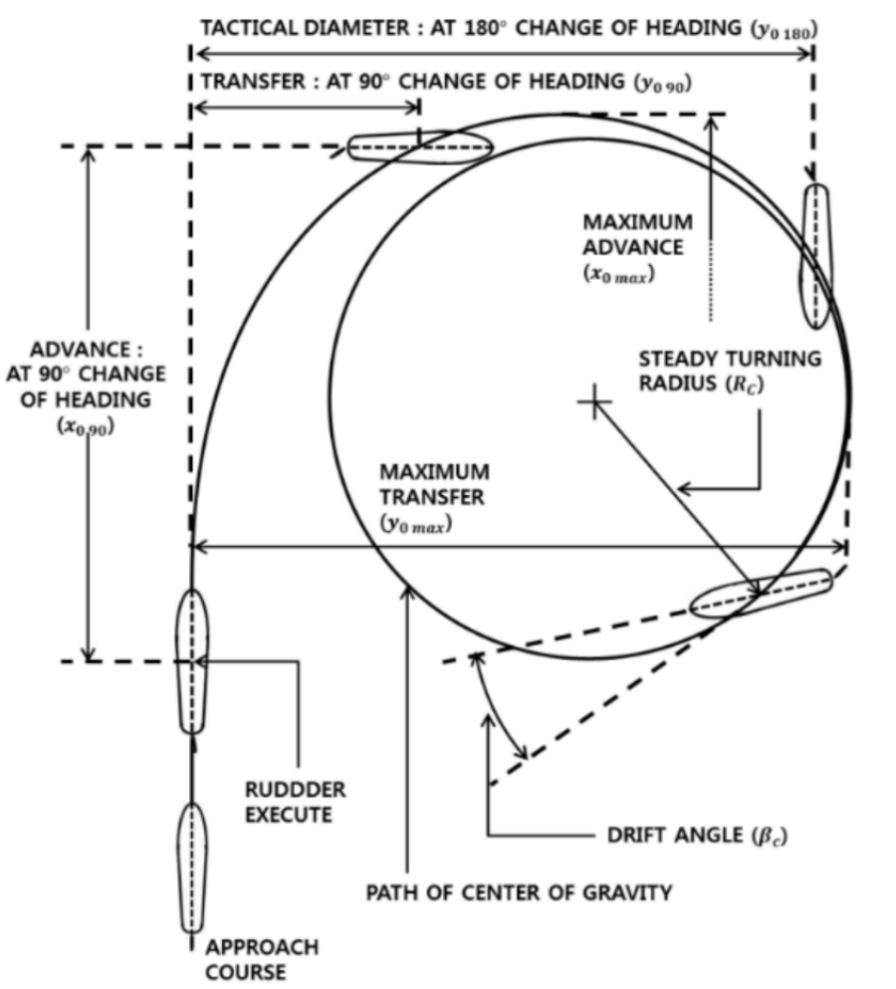

Figure 2.1. Sea trial designed for measuring the vessel's turning radius [13]

The information collected during automated sea trials gives vessel designers and engineers a good overview, which in turn is a great help when designing more capable vessels and developing existing watercraft.

3 Computer vision

Computer vision is a field of artificial intelligence concerned with extracting information that is meaningful and useful to humans from digitised visual material (photos, videos) and then making decisions based on that information, using computers. Where artificial intelligence gives computers the ability to think, computer vision gives them the ability to see, observe, and understand their surroundings.

Computer vision functions similarly to human vision, but humans have a lifetime head start at this skill – they begin learning to distinguish objects from birth. Over a lifetime a person becomes able to distinguish many objects, judge their distance, recognise whether they are stationary or moving, and detect defects in images or objects.

In computer vision, computers are "trained" to carry out similar operations, but much faster, using cameras, massive datasets, and algorithms in place of the human eye's retina, optic nerves, and the brain's visual interpretation. Such a system also has advantages over human vision, chiefly specialisation. Once we teach the system what a vessel and a buoy look like, the resulting detection model is able to analyse many hundreds of objects and situations per second and react with far greater accuracy, reliability, and speed than a human performing the same task.

Computer vision is in use in several fields: energy management (consumption forecasts), manufacturing (product-defect inspection), and the automotive industry (autonomous vehicles and driver-assistance systems), and the demand for computer vision specialists is on the rise [14].

3.1 Computer vision operating principle

To train and create a computer vision model, a massive dataset is required; it is analysed many thousands of times until the model finds regularities, patterns, and – in the case of objects – characteristic shapes in the data, which finally allow it to identify the sought

object independently in new, previously unseen images.

Machine learning is used to perform this task; it includes artificial neural networks and, within those, deep learning and convolutional neural networks in particular.

Machine learning uses algorithmic models that enable a computer to understand the content of visual data. Given a sufficient amount of data, the computer can teach itself, on the basis of that data, what an image contains and how images differ from one another. Algorithms enable the machine to learn on its own, which is significantly more efficient than a human pre-programming the computer to recognise images.

A convolutional artificial neural network helps the computer vision model understand an image by examining the input as several smaller, labelled regions. Convolutional operations are performed on the labelled regions, on the basis of which the neural network makes predictions about what it sees. The neural network repeats these operations and iteratively checks the accuracy of its predictions until the results are accurate enough and true. Once such an operation is successfully completed, the model is able to see and recognise images in a format understandable to a human [14].

3.2 Convolutional neural network

The convolutional neural network (*Convolutional Neural Network, ConvNet, CNN*) is a deep-learning algorithm that can find characteristic features of a given input image and distinguish them from one another. Compared with other image-recognition algorithms, the amount of preprocessing required for a CNN is significantly smaller. A CNN is also capable of creating filters on its own, with enough training, in order to distinguish image features (learning the peculiarities of images), unlike manually tuned algorithms, which cannot do this as accurately [15].

This neural network is distinguished from others by the layers that form its foundation:

- Convolutional layer
- Pooling layer
- Fully connected layer

The convolutional layer is the CNN's first layer, the fully connected layer is the last, and

pooling and additional layers sit between them. With each successive layer the complexity of the network grows and the system's understanding of the input image becomes more complete. The upper layers grasp simpler features such as colours and edges. In deeper layers the system develops a more generalised understanding of the elements or images in the picture, and on the basis of the information obtained – provided there is enough of it – decides what the photo depicts.

The **convolutional** layer (*convolutional layer*) is the most important part of a CNN; the largest amount of computation takes place here. This layer requires input data, a filter, and a feature map. Assume the input is a colour image in the form of a three-dimensional matrix (green, red, blue – three channels; height, width, depth – three dimensions). With that, the input data are defined. We also need a feature-detection function – the kernel or **filter** – which moves stepwise across the data, checks whether a particular feature is present in the image, and performs the corresponding computations. The operation just described is convolution.

The **filter** is a two-dimensional matrix filled with constant weights that represents a portion of the image. Filters come in various sizes, but traditionally they are three-by-three matrices. The filter is applied to a portion of the image, producing the dot product of the image region matrix and the filter matrix, which is passed on to the output matrix. The filter matrix then shifts forward by a predetermined number of pixels until this operation has been performed across the entire image, resulting in a map of convolutional features (see Figure 3.1).

Looking at the figure, we see that in the output matrix a single argument corresponds to several input pixels – i.e., for this operation both the convolutional layer and the pooling layer discussed next belong to the partially connected layers of the neural network. After each convolutional operation the feature map is processed in a ReLU [16] activation function layer, and the resulting outputs are then sent into the **pooling** layer.

The **pooling** layer downsamples the input, thereby reducing the number of input parameters, increasing the efficiency of the neural network, and reducing the risk of overfitting. Similar to the operations of the convolutional layer, processing here also covers the whole image; the only difference from convolution is that the filter in this layer has no weights. In this layer it is possible to perform max pooling and average pooling. In the first case, as the

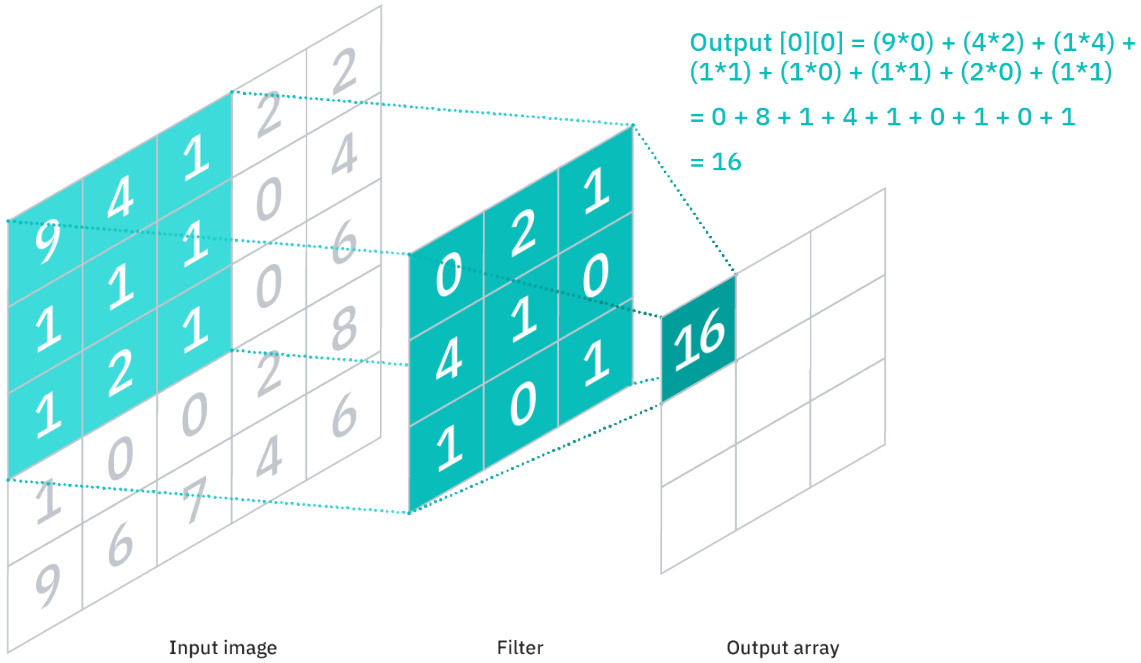

Figure 3.1. The image, represented as a matrix (left), is operated on with the filter (centre), producing a dot product as a feature map (right).

filter moves across the image, the maximum-value pixel is chosen and sent to the output matrix; in the second case, the mean value of the pixels under the filter as it moves across the image is computed and passed to the output matrix. Max pooling is used more than average pooling.

The **fully connected** layer classifies the information gathered and filtered from the previous layers. This layer uses the softmax [17] activation function to classify the inputs correctly, producing a probability estimate for each class such that the estimates for all classes sum to one (see Figure 3.2) [18].

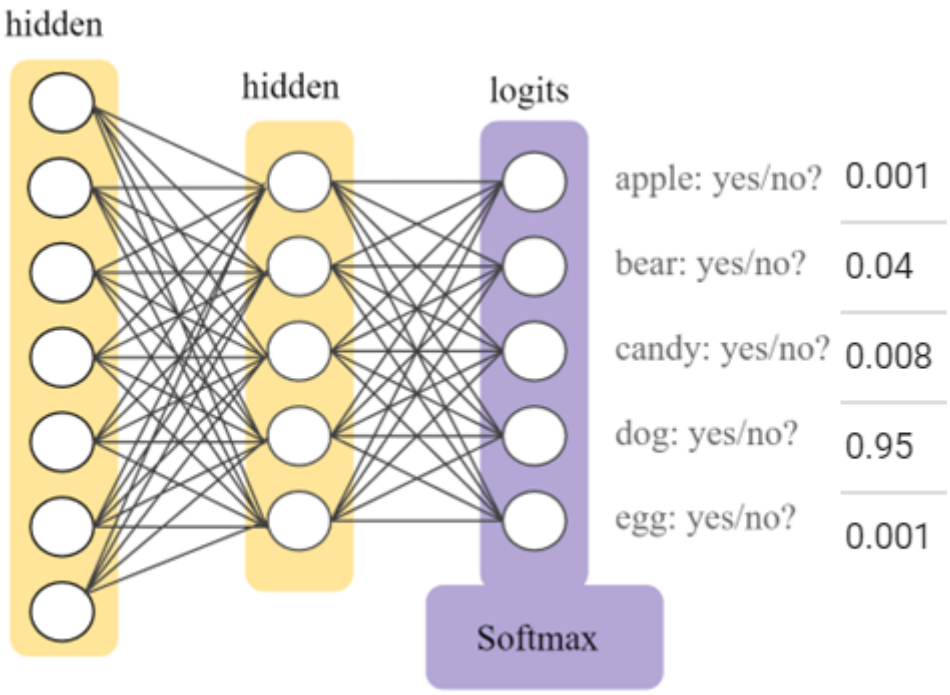

Figure 3.2. SoftMax-applied output layer in the neural network (purple) and probabilities for each class (right) [17]

4 Hardware selection

Hardware selection was guided by the given requirements so that the control-tower prototype would operate as flawlessly and reliably as possible. It was also taken into account that the control tower should be as easy as possible to reconfigure for different vessel types. The requirements set for the system were:

- Three cameras as inputs, with a resolution of at least 1000×1000 pixels.
- Vessel-detection rate of at least 15 frames per second.

A similar system is intended to be applied to different vessel types in the future, so the components were chosen from among widely used and readily available devices.

4.1 Control unit / onboard computer

The control unit is a device that processes information from the cameras attached to it and forwards the results to the autopilot. The onboard computer sits in the vessel's control tower, surrounded by three cameras. Analytical algorithms are applied to the information from the cameras (in this context, real-time video); their output – detected objects and their relative positions in the frame – is passed to the autopilot, providing additional information about the surrounding environment, on the basis of which the autopilot decides how the vessel should behave.

There are several single-board computers suitable for this task (*Single-Board Computer*, SBC [19], *System on Module* (SoM)), the most popular being the Nvidia Jetson series [20], which is specialised for AI, computer-vision, and machine-learning applications, and the Raspberry Pi [21]. The Raspberry Pi is ruled out here because the control-tower system involves working with four cameras in parallel (the Raspberry Pi has only one camera input). The task requires substantial compute, but the Raspberry Pi lacks both the necessary throughput and the requisite hardware (the device's CPU does include an integrated graphics processor, but it is too weak for the task at hand). For parallel

work, a discrete graphics processor (*Graphics Processing Unit* (GPU) [22]) together with specialised CUDA cores (*CUDA Cores* [23]) is particularly beneficial – one of the Nvidia Jetson computers' main advantages.

The Jetson family contains four devices: Nano, TX2, Xavier NX, and AGX Xavier, the last of which is the most capable (see Table 4.1). The Nano is the most basic version, suitable for the simplest computer-vision applications; the TX2 is somewhat more capable in CPU and GPU performance; and the Xavier NX and AGX Xavier are aimed at industrial use cases, with six- and eight-core CPUs and Volta-architecture GPUs that also include matrix-operation accelerators (*Tensor Cores*) [24] (see Figure 4.1).

Table 4.1. Parameters of the Jetson-series computers [24]

Parameters	**Jetson Nano**	**Jetson TX2 Series (NX, TX2i)**	**Jetson Xavier Series (NX, AGX)**
AI Performance	472 GFLOPs	1.33 TFLOPs	21/30 TOPs
GPU	128-core NVIDIA Maxwell™	256-core NVIDIA Pascal™	384/512-core NVIDIA Volta™ with 48/64 Tensor Cores
CPU	Quad-Core Arm Cortex-A57	Dual-Core NVIDIA Denver and Quad-Core Arm Cortex-A57	6/8-core NVIDIA Carmel Arm
DL Accelerator	_____	_____	2x NVDLA v1
Vision Accelerator	_____	_____	2x PVA v1
Memory	4 GB 64-bit LPDDR4	4/8 GB 128-bit LPDDR4	8/16/32/64 GB 128-bit LPDDR4x
Storage	16 GB	16/32 GB	16/35/64 GB
CSI Camera	Up to 4 cameras (up to 18 Gbps)	Up to 5/6 cameras (up to 30 Gbps)	Up to 6 cameras (up to 62 Gbps)

Given that information from three cameras must be analysed in parallel and with high accuracy, it makes the most sense to use a Xavier-series system. Compared with the other series, the Xavier-series computers offer their performance through a larger number of GPU cores, a newer architecture (Volta [25]) that includes matrix accelerators (*Tensor Cores*, which neural networks need for matrix computations), and Deep Learning Accelerators, which enable more efficient processing with multiple camera inputs.

Within this series both the NX and AGX Jetsons are available, and the best candidates have been chosen (see Table 4.2). The most suitable among these is the Jetson AGX Xavier, because the device must be able to run a large neural network and handle several operations at once (reading information from the cameras, processing it in the neural network, and presenting the processed information in the output video), which requires substantial computational resources.

Table 4.2. Parameters of the Jetson AGX Xavier and the Xavier NX [24]

Parameters	**Jetson AGX Xavier**	**Jetson Xavier NX 16GB**
AI Performance	32 TOPs	21 TOPs
GPU	512-core NVIDIA Volta™ GPU with 64 Tensor Cores	384-core NVIDIA Volta™ GPU with 48 Tensor Cores
CPU	8-core NVIDIA Carmel Arm®v8.2 64-bit CPU 8MB L2 + 4MB L3	6-core NVIDIA Carmel Arm®v8.2 64-bit CPU 6MB L2 + 4MB L3
DL Accelerator	2x NVDLA v1	2x NVDLA v1
Vision Accelerator	2x PVA v1	2x PVA v1
Memory	32 GB 256-bit LPDDR4x 136,5 GB/s	16 GB 128-bit LPDDR4x 59,7 GB/s
Storage	32 GB eMMC 5.1	16 GB eMMC 5.1
CSI Camera	Up to 6 cameras (up to 62 Gbps)	Up to 6 cameras (up to 30 Gbps)

Figure 4.1. The chosen computer: the Jetson AGX Xavier (right, without the heat-sink module) [26]

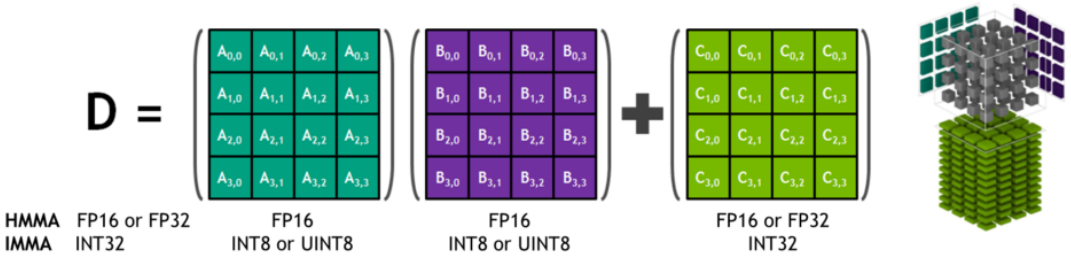

Figure 4.2. A Tensor Core multiplying and summing three-dimensional 4×4×4 matrices [27]

4.2 Tensor cores and CUDA

Tensor Cores are programmable matrix-operation accelerators that perform their operations in parallel with the CUDA [23] cores. These cores implement a newer kind of mixed-precision floating-point and integer operations – HMMA (*Half-Precision Matrix Multiply and Accumulate*) and IMMA (*Integer Matrix Multiply and Accumulate*) – which accelerate linear algebra, signal processing, and deep-learning inference [27] (see Figure 4.2).

CUDA is a parallel-computing platform and programming model that increases the number of computations multi-fold by using the GPU in place of the CPU [23].

The purpose of CUDA is to accelerate parallel computation, and a CUDA core is analogous to a CPU core. The only difference is in their construction: a CPU core is able to solve complex computations sequentially, while a CUDA core handles simpler computations in the GPU. The GPU's advantage over the CPU is that thousands of CUDA cores are placed on a single chip, which allows complex computations to be divided among many cores and therefore performed faster than a CPU with a smaller number of more capable cores. This technology is integrated and widely used in the machine-learning field by Nvidia.

4.3 Cameras

When selecting cameras, several parameters must be considered to ensure that the system meets the requirements and operates successfully. Because in this task the computer-vision system is housed in the control tower, which is mounted on the vessel's mast, the cameras (or the housing in which they are placed) must be capable of operating across wide temperature ranges and changing weather conditions. The camera software should also be chosen to favour the newest and clearly documented material, and its compatibility with the host

computer must be considered. This ensures better camera configuration and makes the deployed computer-vision software easier to manage.

The following parameters were considered when selecting cameras:

- **Resolution** (in pixels, px) – directly tied to object-detection accuracy and detection speed. The lower the resolution of the input given to the detection algorithm, the less accurate/less certain the neural network's prediction. In this case the detection process is faster, since the algorithm deals with a smaller amount of data and fewer computations, which ultimately improves inference speed.

  At higher resolution, detection accuracy improves but inference speed decreases, because the amount of input data grows. A high-resolution photograph contains more information, which in turn allows the neural network to give more confident predictions.

- **Frame rate** (*Frames per second*, FPS) – a parameter that describes the camera's information throughput, i.e. the ratio between the number of frames transmitted and the time. A high frame rate ensures a visually smoother data stream and easier tracking, and also reduces data loss. At a low frame rate, tracking an object and predicting its trajectory would be significantly more difficult, because the object's motion is visually intermittent and the accuracy of the neural network's predictions is harmed accordingly.
- **Colour sensor** (monochromatic or polychromatic sensor) – i.e., single-colour or multi-colour sensor. A monochromatic sensor suits environments where the colour contrast between the object and the environment is large, ensuring effective object detection. Since this is a monochromatic image, a single value is given per pixel, falling in a fixed range (typically 0. . . 255 or 0.0. . . 1.0).

  A polychromatic sensor, on the other hand, suits environments where the contrast between background and object is not very large, or where object detection requires more information. With this kind of image transmission, three values are given as a matrix per pixel, the RGB (Red, Green, Blue) colours each in the range 0. . . 255 or 0.0. . . 1.0 (for example [255, 0, 255]). This

transmission mode is three times as data-heavy as transmitting a monochromatic image, since three values are given per pixel.

Devices that would meet the parameters mentioned above are listed below (see Table 4.3):

Table 4.3. Cameras matching the parameters

Camera name	**Resolution (px)**	**Frame rate (FPS)**	**Operating temp. (°C)**		**Interface**
SurveilsQUAD – Sony IMX290 System [28]	1920x1080	120	-30	85	CSI
OpenCV AI Kit: OAK—D [29]	1280x800 (stereo) 4056x3040 (centre)	120 (stereo) 60 (centre)	N/A	N/A	USB-C/PoE
OpenCV AI Kit: OAK—1-PoE [30]	4056x3040	60	N/A	N/A	USB-C/PoE
Atlas IP67 7.1 MP Model [31]	3208x2200	74	-20	55	PoE
Atlas IP67 2.8 MP Model [32]	1936x1464	173	-20	55	PoE
Arducam 12MP IMX477 [33]	4056x3040	60	N/A	N/A	USB-C
Arducam Fisheye Camera [34]	2592×1944	30	N/A	N/A	CSI and Ethernet

*N/A – data not available

At the time of writing, the SurveilsQUAD cameras were initially used, since they had been prepared in advance for the computer system, and the relevant software modifications for the Xavier had already been made by project parties at the Department of Computer Systems. The project under development at the time of writing was in its development phase, so the budget was limited; we therefore made do, at first, with cameras whose baseline capabilities sufficed for the experiments. During the control-tower design it became apparent that the placement of the chosen cameras was limited by their short cables, and the manufacturer did not supply longer cables for the units.

Therefore the Arducam Mini camera was chosen (see Figure 4.3), connected to a Raspberry Pi and from there, via an Ethernet cable, to an Ethernet switch and ultimately to the Jetson Xavier. This also solved the cable-length issue, and it became possible to install the latest software on the Xavier, which the older cameras' outdated drivers did not support.

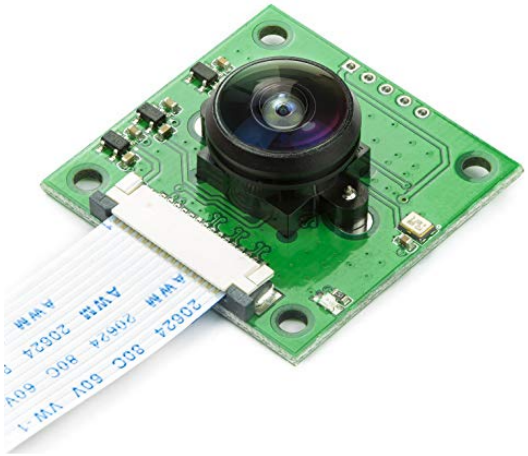

Figure 4.3. Module with the Arducam Fisheye camera [34]

4.4 Control tower

The control tower sits at the bow of the vessel (see Figure 4.5) and its task is to take over control of the vessel on the captain's command and handle it autonomously with as little captain intervention as possible. To do this, the tower has a brain in the form of the Jetson AGX, which gathers information from all sensors and cameras, processes it, and then forwards a control decision to the autopilot, which in turn sends signals to the vessel's engines and rudder. The Jetson receives information about the surrounding environment from cameras, X-band radar, AIS, and a weather-station wind sensor. Data exchange within the control-tower system takes place via an Ethernet switch and a 4G router, which enables data to be sent and received from the user. Developers can also push software updates to the vessel's systems. AIS (Automatic Identification System) helps determine the vessel's position relative to other watercraft using satellites, the radar detects nearby objects as point clouds, and the cameras and the computer vision running on them identify vessels and other watercraft among the detected objects. AIS, radar, and the cameras with computer vision complement one another, helping the vessel detect and orient itself in its surroundings. The system is powered by a direct-current power source (see Figure 4.4).

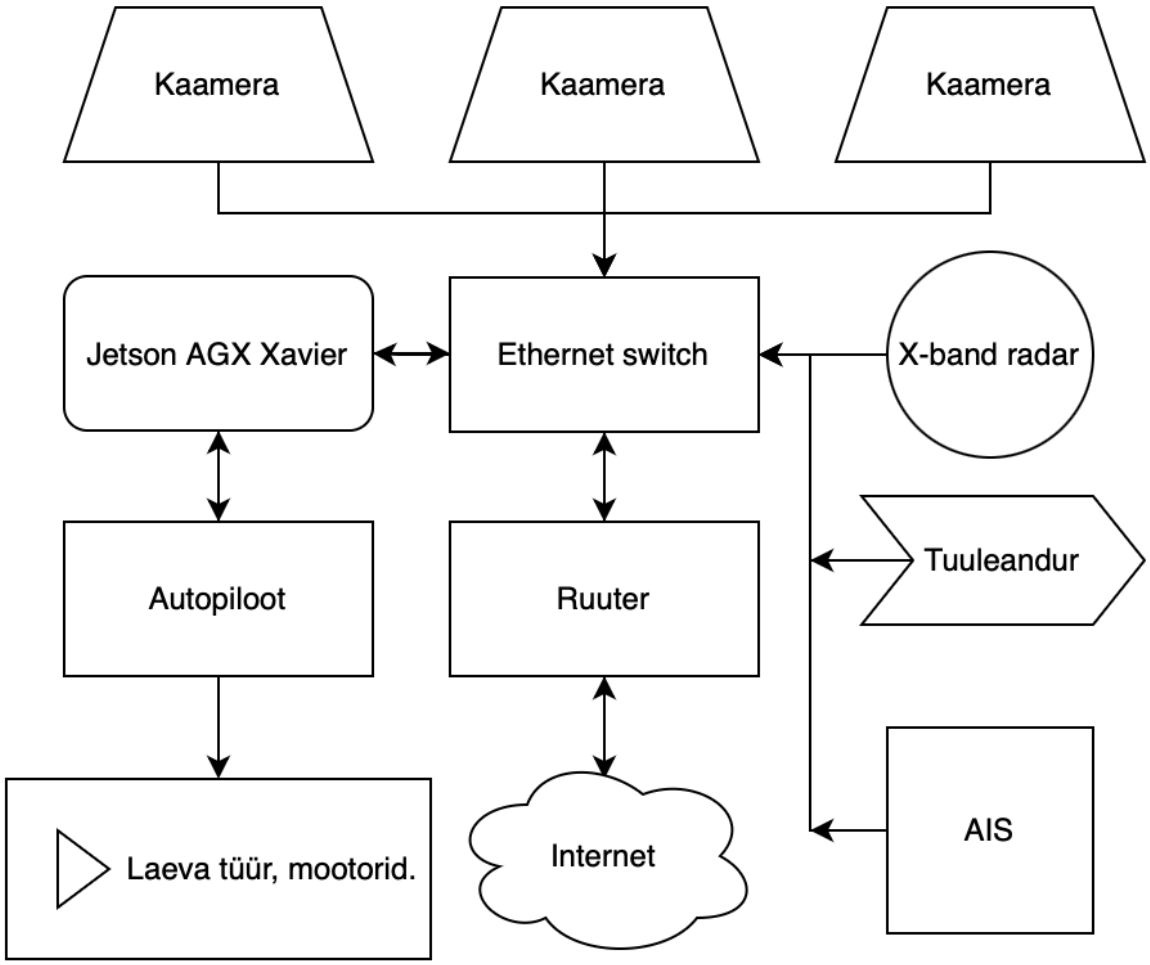

Figure 4.4. Simplified operating diagram of the control tower

Figure 4.5. The control tower's placement on the patrol vessel [35]

5 Software

There are different software solutions for building computer-vision systems. The software for this work encompasses several interdependent parts: image processing and transmission, data processing, and neural networks.

For **image processing**, several modules are available (OpenCV [36], Scikit-Image [37], SciPy [38], Pillow [39]), of which OpenCV and Scikit-Image were built specifically for machine learning. To solve this task, the OpenCV module was used instead of Scikit-Image, because the former also supports reading video via Gstreamer [40], which allows use of the Nvidia video decoder [41] integrated into the Jetson. This saves the CPU when reading the image stream and does so significantly more efficiently, leaving more headroom on the system's CPU for image processing and the computer-vision model's work.

For **data processing** the Pandas [42] module is widely used, providing simple and intuitive commands to analyse and manipulate data. This module performs operations on the CPU, and for extreme data volumes data processing can become time-consuming. In addition to Pandas, several alternative modules have been built that are similarly or marginally better optimised, the closest in operating principle and writing style being cuDF [43]. cuDF is essentially most similar to Pandas, but uses the GPU instead of the CPU for data processing, which – especially when handling large data volumes – gives a significant time advantage over Pandas, because processing takes place with much greater parallelism than on the CPU. At the time of writing, the Pandas module is in use, because the thesis author has broader experience working with this module. There are later plans to switch to development with cuDF for the purpose of software optimisation.

For **neural networks** there are several platforms and frameworks: TensorFlow [44], Keras [45], PyTorch [46], Scikit-Learn [47], of which the most popular and widely used are TensorFlow and PyTorch. This work used PyTorch, because compared with TensorFlow, PyTorch's syntax is more Pythonic, which makes it easier to integrate with Python objects

and makes error messages easier to understand. The thesis author also has prior experience developing with PyTorch.

5.1 Computer-vision neural network model

There are several models available for object detection, differing from one another in accuracy, input resolution, size, and speed. Choosing a model means accounting for the system's throughput: more capable systems such as servers can run massive, accurate models in parallel with minimal impact on input-processing speed. Smaller systems such as laptops and embedded systems, by contrast, are better suited to running smaller, faster models.

Table 5.1. Overview of the parameters of the most popular models

Model	**Max input (px)**	**Speed (ms)**	**Accuracy (mAP)**
FasterRCNN [48]	1000x600	198	73.2 (PASCAL VOC 2007)
YOLOv5 [49]	1280x1280	26.2 (V100 GPU)	72.7 (MSCOCO)
MobileNet SSD V2 [50]	320x320	200 (Google Pixel 1)	22.2 (MSCOCO)
EfficientDet [51]	1536x1536	153 (V100 GPU)	55.1 (MSCOCO)

For this system, the model had to be chosen with the best accuracy-to-speed ratio in mind. The chosen model also had to be capable of processing information from several video inputs at once (see Table 5.1).

The YOLOv5 architecture network was selected to solve the task; it was built by Ultralytics [52] as an evolution of YOLOv4 [53] and is faster and more accurate than its predecessors. It is also a compact model that is easy to run on embedded systems. The model operates on the single-pass frame-processing philosophy (*You Only Look Once*) – the image passes through the network only once (unlike other models), and within that single pass object detection and classification happen together, which is why the model's detection operation is also faster (see Figure 5.1).

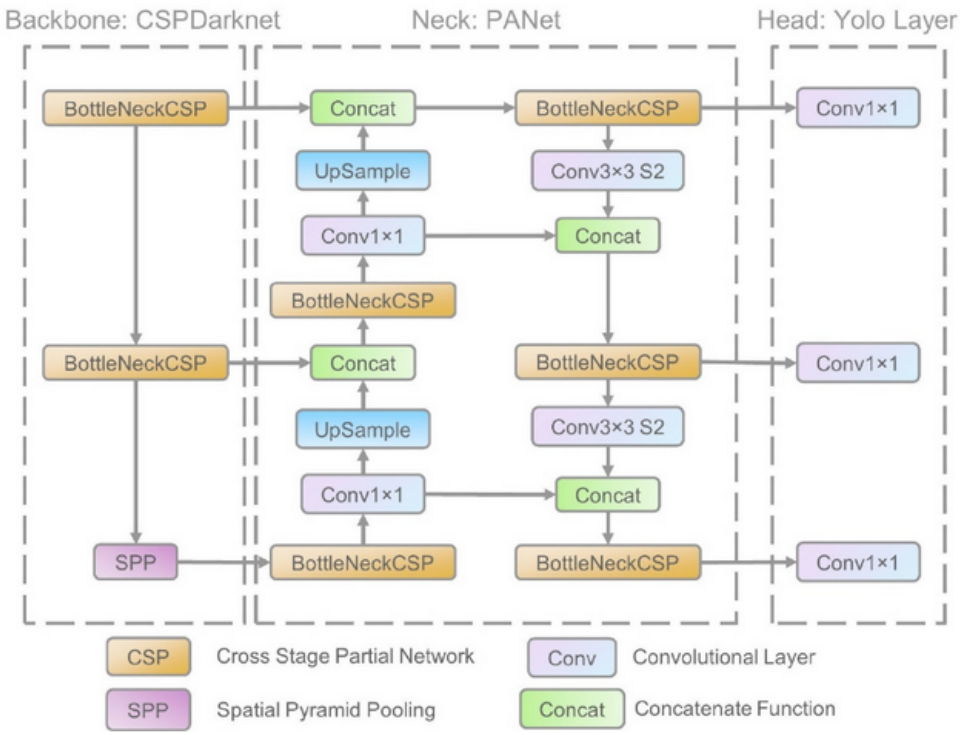

Figure 5.1. Architecture of the YOLOv5 models [54]

The model consists of three parts: backbone, neck, and head.

- The **backbone** is the base on which the model is built; here, CSPNet (*Cross-Stage Partial Network*) was chosen. It is a kind of neural-network model split into several stages (*Cross-Stage*), in which each layer extracts information from the given image – such as edges and shapes characteristic of the object in the image. Part of the information acquired in earlier layers is passed on to the next layers for processing (*Partial*), which allows the network to progressively understand what the image depicts better and more accurately, using processed information from the previous layers. The backbone consists mainly of convolutional layers and a downsampling layer, whose output passes on to the model's neck. A model with this kind of architecture helps reduce the number of computations required.
- The **neck** of the neural network is responsible for assembling and organising the information received from the backbone into a feature map, using filters. This model applies a spatial attention pooling method that helps the network focus on important parts of the image, and a Spatial Pyramid Pooling layer, which allows objects to be

detected at different resolutions.

- The **head** generates the probabilities and coordinates of detections for the objects found in the frame. This model also uses anchor boxes, which are pre-configured with given aspect ratios that help detect objects of different sizes and shapes. The model's head contains a number of convolutional layers that predict detection coordinates and probabilities for each object [54].

6 Computer-vision application

The aim of this work was to build the computer-vision application in such a way that it would be able to run on as wide a range of control units as possible. To do this, the application had to be packaged so that the controller's existing configuration was not altered and no additional software was installed. At the same time, the application had to be able to choose its internal settings based on the hardware, so that the software's operation would be as close to the hardware as possible – i.e. optimised – thus ensuring universality. The application's inputs had to be easily interpreted by the user and well documented, to avoid bad inputs. If the software's operation got stuck, displaying a correct and informative error message was important. A simplified version of the device's software solution has been released by the author and is available – and continuously updated – on GitHub [55].

6.1 Working environment

Software development and the computer-vision processes' work take place in an isolated environment, allowing the software to be tested on systems with different hardware configurations. This development approach also guarantees, from a security and stability standpoint, that the host system's software and state remain untouched. Containerisation lets a single device run several applications, all of which can have different configurations and software, while granting the user-defined direct access to the host's hardware (see Figure 6.1).

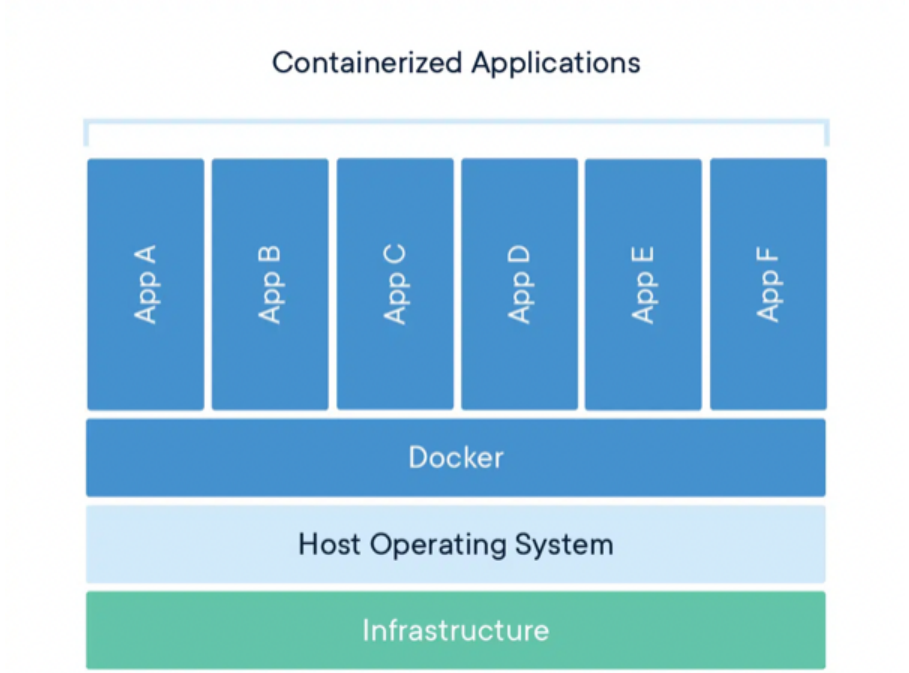

Figure 6.1. Simplified diagram of how containerised applications operate [56]

For system development and testing an environment was set up on a separate development machine using the virtualisation platform Docker [57]. The base was the PyTorch container from the Nvidia NGC catalogue, intended for x86_64-architecture devices [58]. An environment analogous to the Jetson's software specifics was built into it, and architecture-specific functions were also added in code that took effect according to the peculiarities of the development system and the production system (see Figures 6.2 and 6.3).

```
root@b562b8aef883:/workspace/torching# python3
Python 3.8.12 | packaged by conda-forge | (default, Oct 12 2021, 21:59:51)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch, cv2
>>> cv2.__version__
'4.2.0'
>>> torch.__version__
'1.11.0a0+bfe5ad2'
>>>
```

Figure 6.2. Versions of the software installed in the development-machine container, chosen to match the production-machine container state as closely as possible

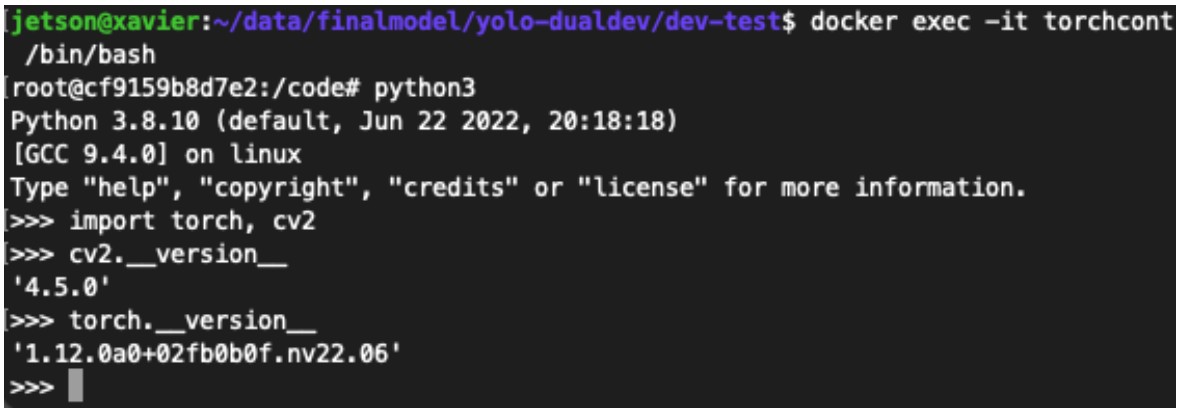

Figure 6.3. Software versions in the production-system container

A container was likewise created for the production system; its base was a container from the NGC catalogue built on L4T (*Linux for Tegra*), specifically designed for aarch64-architecture Jetson devices [59]. It was configured with the additional libraries needed for computer vision and tied in with the on-device neural-network model optimisation tool, TensorRT [60].

6.2 Main program

The main work happens in the inference script (see the simplified Figure 6.4 and the technical Figure 6.6), to which the user provides as input the neural-network model and its required parameters, the input stream (a video file or a camera), and, if needed, the desired mode (video recording, or a mode without video output intended for debugging and speed checks) (see Figure 6.5).

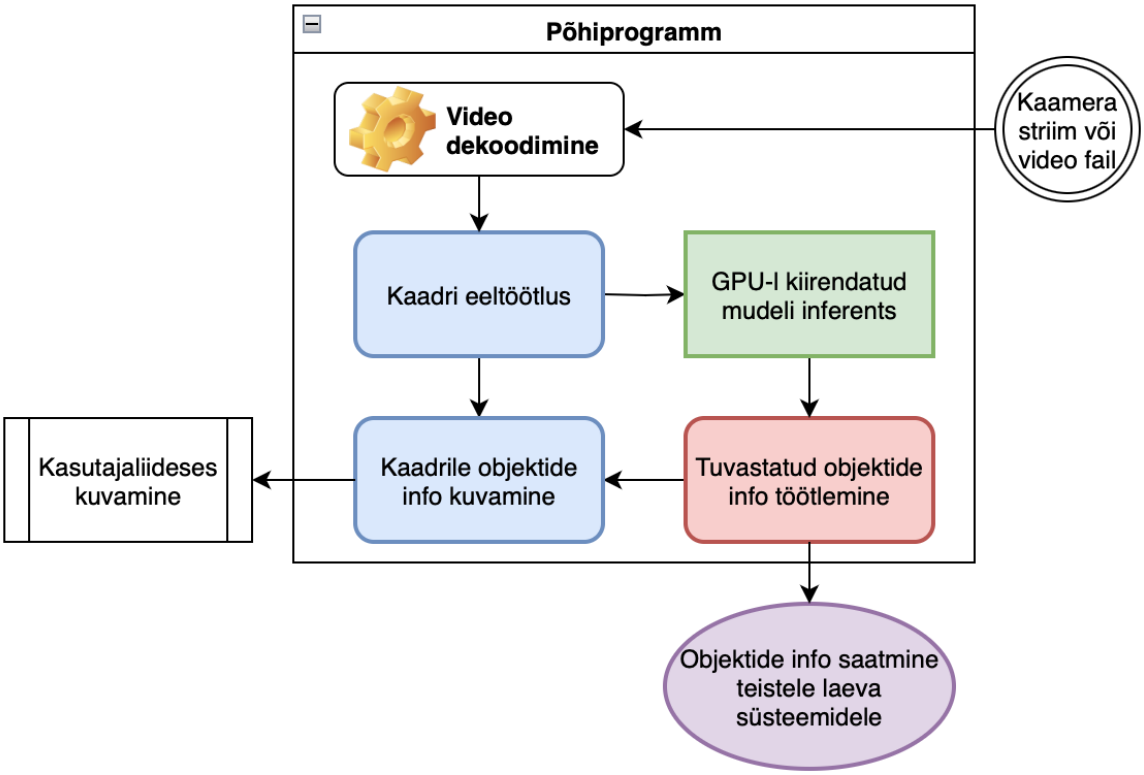

Figure 6.4. Simplified block diagram of the main program

```
jetson@xavier:~$ python3 inference.py --rt-model models/yolov5m6_640x640_batch_1.engine
 --input-video video.mp4 --perf
```

Figure 6.5. Example of running the main program from the command line

A script called `inference.py` is run with Python; it is given as input a TensorRT-optimised model named `yolov5m6`, the input stream is a video file `video.mp4`, and performance mode is also enabled, which runs the process without the web-interface video output to the user – making it possible to evaluate system latency and the direct effect on Xavier resource usage.

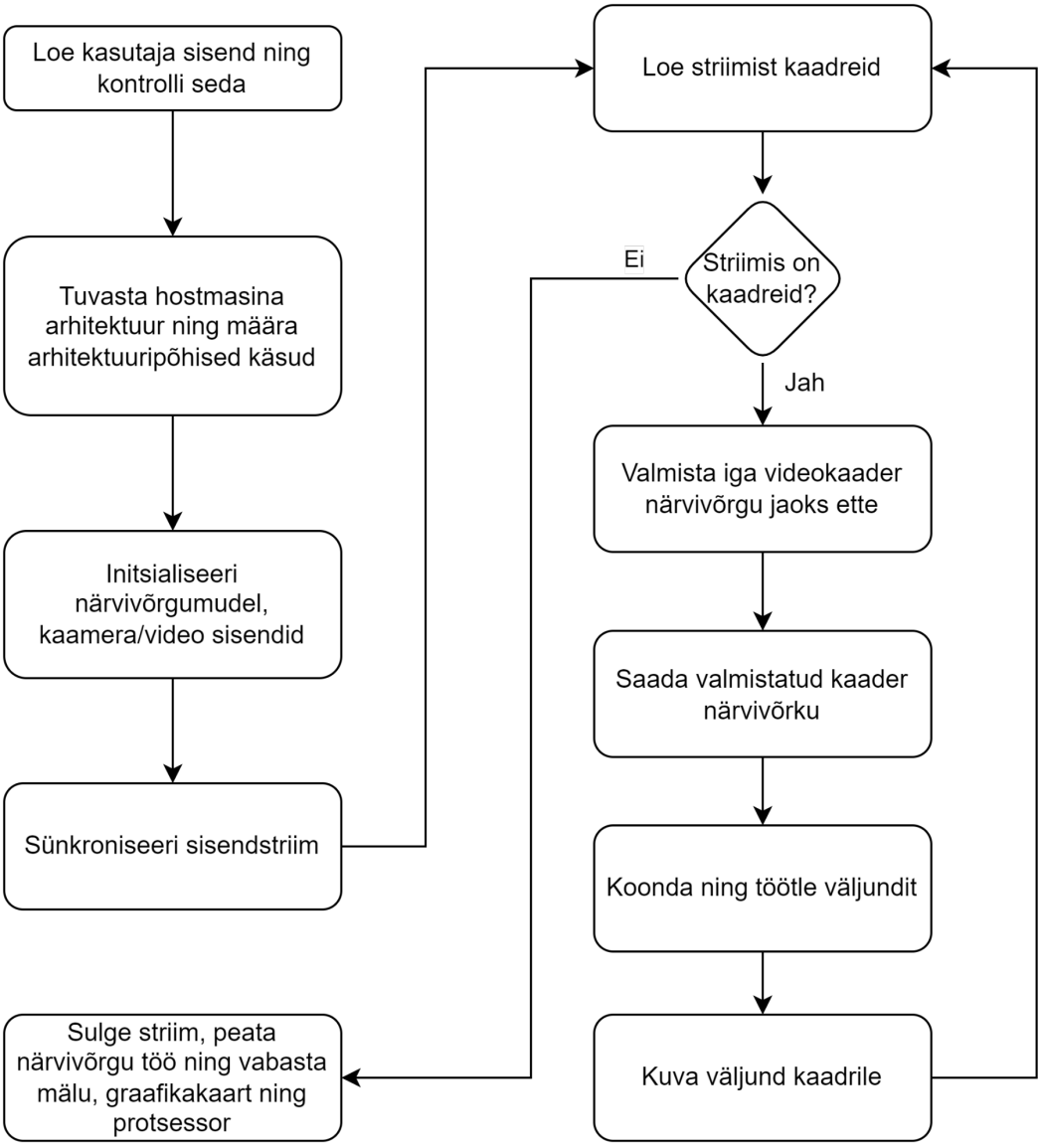

Figure 6.6. Simplified block diagram of the computer-vision program's operation

To detect the machine architecture, a Python module is used; based on its output, the required settings are applied for the environment that ran the script. The development environment is a desktop computer with an x86_64 architecture; during setup the corresponding video decoder and encoder are initialised, the special-use port is set, and a scaling parameter for the information coming from the cameras is configured (see Figure 6.7).

```
if platform.machine() == "x86_64":  # For PC
    decoder = "avdec_h264"
    video_converter = "videoconvert"
    app_port = 3000
    resize = True
elif platform.machine() == "aarch64":   # For Jetson
    decoder = "nvv4l2decoder"
    video_converter = "nvvidconv"
    app_port = 3030
    resize = False
```

Figure 6.7. Initialisation of architecture-specific parameters

The neural-network model is loaded using the user-defined "model" parameter, which checks the existence of the model and the file format – in this case, `.engine`-type models optimised with TensorRT [59]. After the model-format check, the model parameters' metadata is read in; based on it, GPU memory and core resources are allocated.

To read information from the cameras, the Nvidia Video Decoder (NVDEC) [61], a hardware accelerator located on the Nvidia GPU, is used; this allows the CPU and GPU to be fully dedicated to managing the neural network and processing data.

Frames received from the cameras are scaled to an input format suitable for the neural network, after which the model extracts information from the frames and formulates statistical outputs. The output information is then sent to post-processing, where the results are formatted and the locations of detected objects on the frame are displayed along with their confidence probabilities (see Figure 6.8). A new frame is then read in, and the program's work repeats until no more frames come in – i.e. either the camera stops working or the user terminates the program.

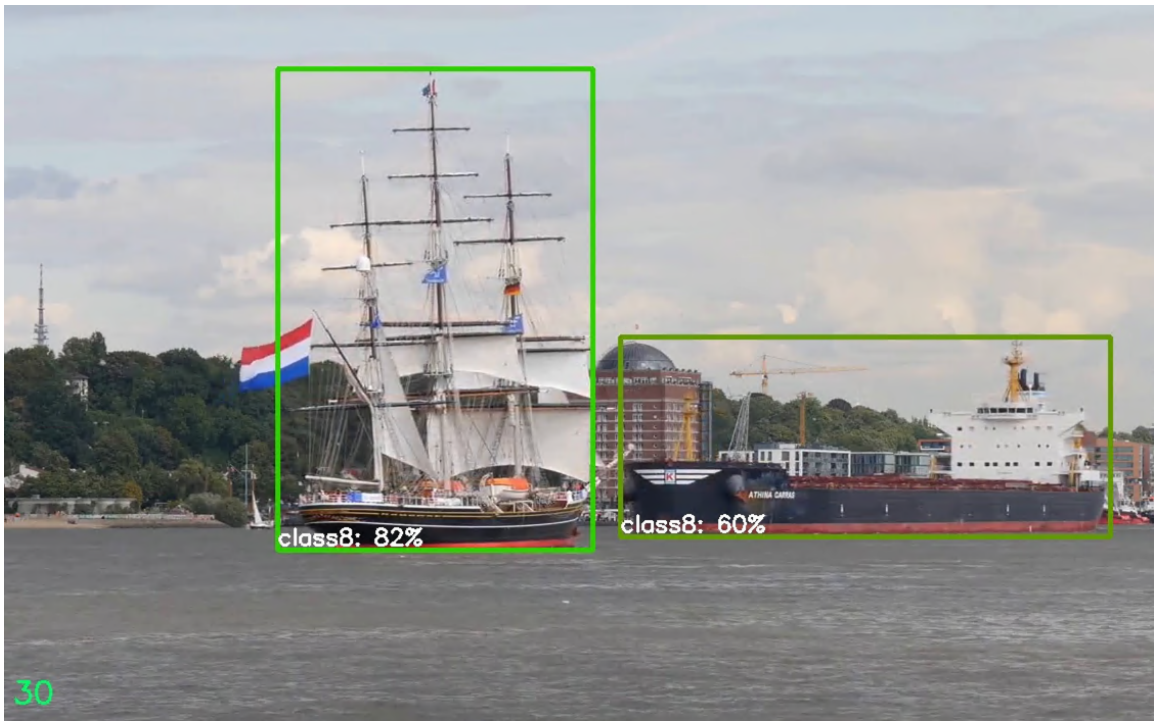

Figure 6.8. Detections generated and processed by the neural network, displayed on a frame (bounding boxes and the detected classes inside them together with the probabilities)

6.3 Computer-vision model and software optimisation

In the early phase of the computer-vision application, several problems arose both with reading inputs from the cameras and with running the computer-vision model on the hardware.

The first problem with the application was reading the stream from the cameras with multiple inputs, which was demanding for the CPU (frame throughput was low) and therefore the control unit's temperature was high. As a result, there was not enough resource left over for the other devices that communicated with the Jetson (the radar's operation was crippled). Computer vision was also not operational, because the CPU was fully occupied with reading the frames from the cameras and rescaling them.

To solve this, the hardware video decoder NVDEC [41] integrated into the Jetson was put into use, via OpenCV, by providing it with an accelerated GStreamer command that included the input and the video-decoder call.

A second problem was that larger computer-vision models could not be run successfully,

because the models were loaded into CPU memory and a marginal portion of the computational work, alongside frame processing in the neural network, fell on the GPU (see Figure 6.9).

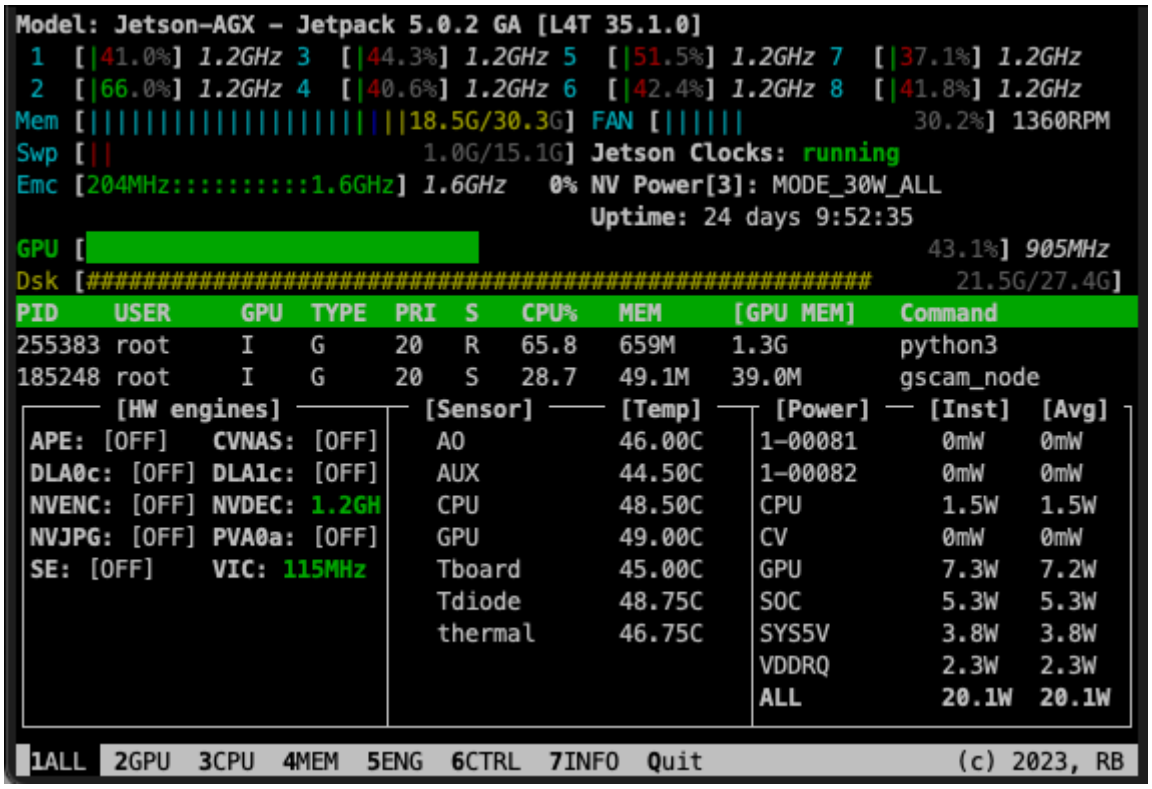

Figure 6.9. Running YOLOv5l, with a single input at 640×640-pixel resolution, in unoptimised form does not utilise the whole GPU, as seen in the monitoring tool jtop [62]

The solution to this problem was Nvidia's TensorRT tool, which can take neural networks in several formats as input and accelerate them according to the hardware available. When using this tool, neural networks were prepared in ONNX [63] format, the most universal and easy-to-use way of interpreting machine-learning models.

TensorRT's operating principle is model quantisation – reducing the precision of the model's parameter computations (for example, from 32-bit floating-point to 16-bit floating-point). This allows the use of high-performance vector operations on hardware specialised for them, with marginal impact on the model's accuracy. Second, model pruning takes place – the removal of unnecessary parameters that have minimal impact on the model's inference results, reducing the size of the model and thereby improving its speed (see section 5.2).

As a result of the optimisations (see Figure 6.10):

- GPU utilisation rose (from 43.1% to 99.8%);
- average power consumption decreased by 26.8% (from 20.1 W to 14.7 W);
- operating temperatures dropped by 3.3% (from 46° to 44.5°);
- memory usage dropped by 1.6% (from 18.5 GB to 18.2 GB).

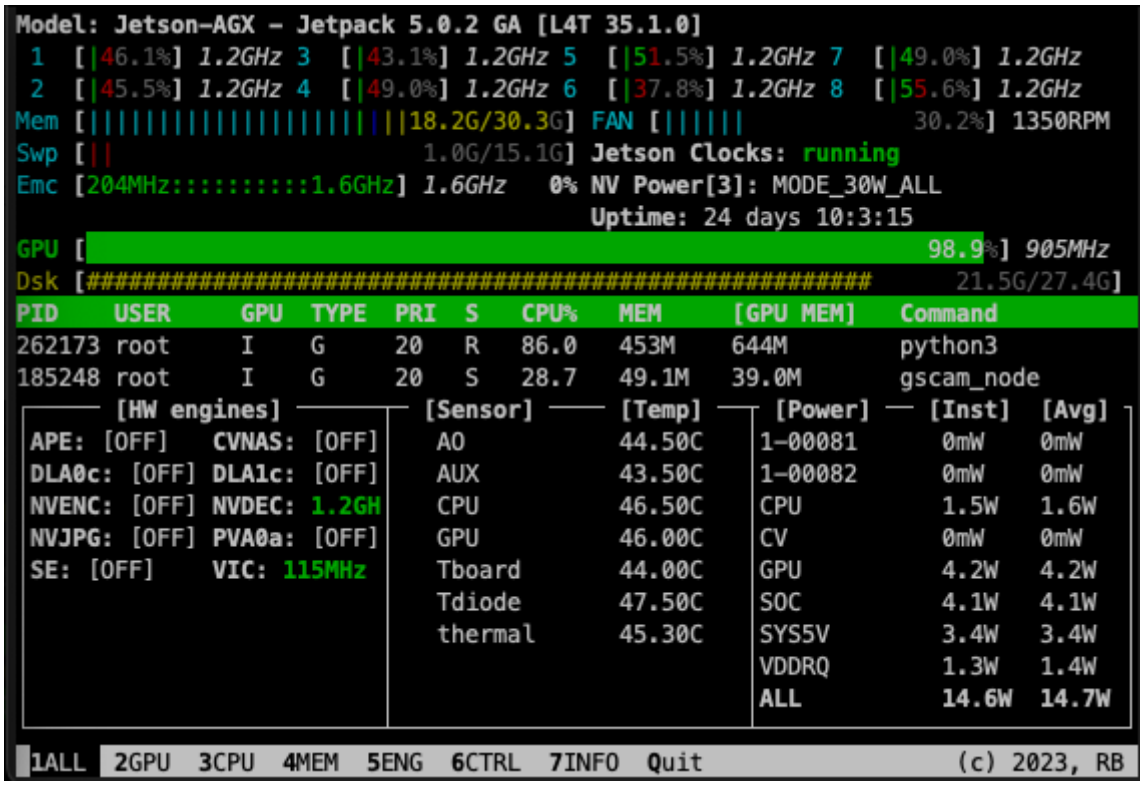

Figure 6.10. Running the YOLOv5l model, with a single input at 640×640-pixel resolution, in optimised form utilises the whole GPU

7 Results

Selecting the most suitable model was based on inference speed and the model's accuracy in evaluating objects. All experiments were carried out on a Jetson AGX Xavier 32 GB unit. The model-evaluation experiments were carried out under ideal conditions, which excluded all possible factors that could introduce deviations in the test results of the evaluated model.

These experiments analyse only the neural-network model's processing capability, isolated from the rest of the system (cameras, preprocessing of the frames coming from them, and presentation/transmission of the detected-object information from the frames to other systems), in order to obtain the most optimal and bias-free result regarding model performance and hardware utilisation.

7.1 Conducting the experiments

The experimental environment conforms to the following conditions:

1. The frames used for inference are specifically selected, standardised, and prepared before the test begins – i.e., scaled to the size required for the neural-network input and preprocessed. This excludes any additional work or load on the GPU and CPU that would negatively affect the neural network's operation, and allows all the tested models to be evaluated equally on a common dataset.
2. Before each test run, the tested models have undergone a hardware warm-up phase – i.e., have been fully read into the GPU memory of the machine and have run 50 calibration inference iterations using the previously mentioned prepared test frames. This creates an environment in which all models are equally prepared and ready to operate at their maximum capability.
3. The number of test runs is 1000, during which the inference speed and detection accuracy on the prepared frames are measured.
4. Detection accuracy is evaluated using the following formula:

$$\alpha = \frac{Korrektsed\ tuvastused}{Kõik\ tuvastused},\ kus\ 0 \leq \alpha \leq 1$$

The closer the result is to one, the more accurate the model and the more capable it is of detecting objects.

5. The model's speed is evaluated by the time taken for inference in milliseconds (shown in the charts as the number of frames processed in one second).

Jetson AGX Xavier software configuration:

- JetPack 5.0.2
- Ubuntu 20.04.5
- kernel 5.10.104-tegra
- L4T 35.1.0
- Python 3.8.10
- torch 1.12.0
- torchvision 0.13.0
- cv2 4.5.0
- TensorRT 8.4.1.5
- CUDA 11.4

7.2 Analysis of results

The experiments were run in two configurations: with one frame and with three frames processed in parallel, since in the production environment the neural network processes information from three frames. The *mean Average Precision* parameter shown in the charts (in yellow) expresses the model's mean total accuracy across all classes; it is derived from experiments carried out on classes selected from the COCO 2017 validation dataset, with an equal number of test samples chosen from all classes [64]. The *accuracy* parameter (in red) was obtained on a separately created dataset containing only vessels. Results are shown in ascending order of model size (number of parameters) – the smallest model on the left, the largest on the right – to provide an intuitive overview of all parameters.

The accuracy fluctuations seen in the charts for models with larger inputs (see Figures 7.2 and 7.4) are due to the fact that the scaling layers of the "6"-suffixed models were designed

for larger input resolutions, while the layers of the other models were designed for smaller input resolutions [65]. The latter therefore exhibit lower accuracy at this resolution.

The speed difference between optimised and unoptimised models is at least twofold; the largest speed difference is 4.28×, for the yolov5l6 pair (see Figure 7.3). The accuracy difference between optimised and unoptimised models is minimal ($1.7 \cdot 10^{-3}$), i.e. optimisation did not significantly affect accuracy. As a result, the accuracy values initially shown on the chart overlapped, so their averaged values are displayed instead.

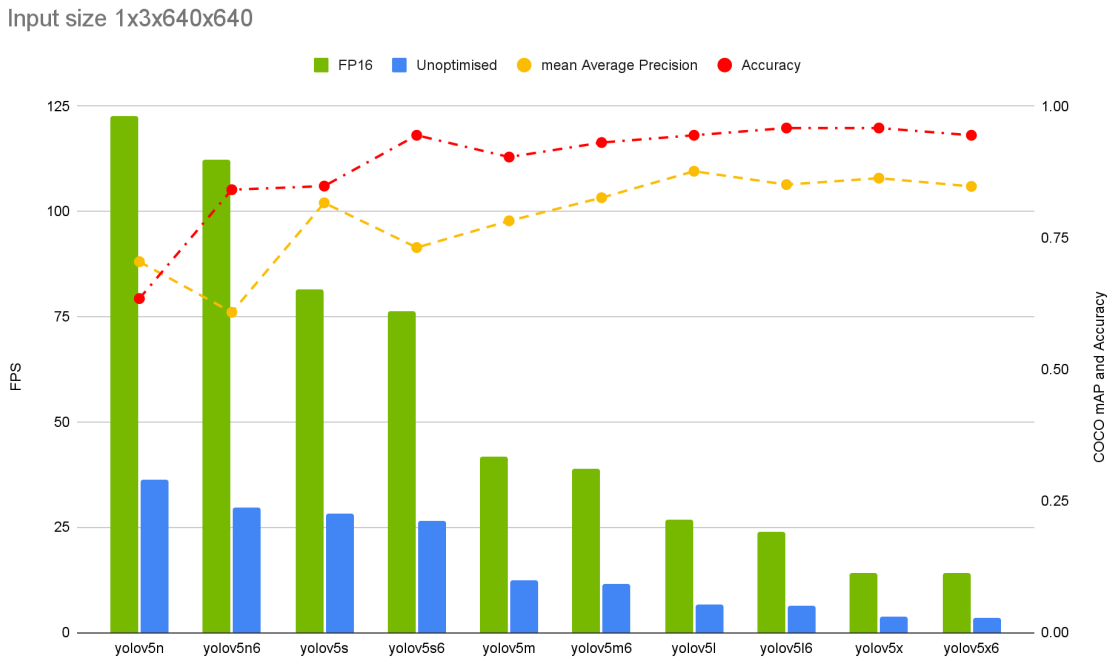

Figure 7.1. Model performance with a single 640×640-pixel frame input

The speed difference between the fastest and slowest optimised model is 108 frames per second, with the best speed-up being fourfold. The mean optimised speed difference is 3.5×. Model accuracy grows up to the YOLOv5l model and there is no significant accuracy gain beyond it; from that model onward the speed-to-accuracy ratio decreases.

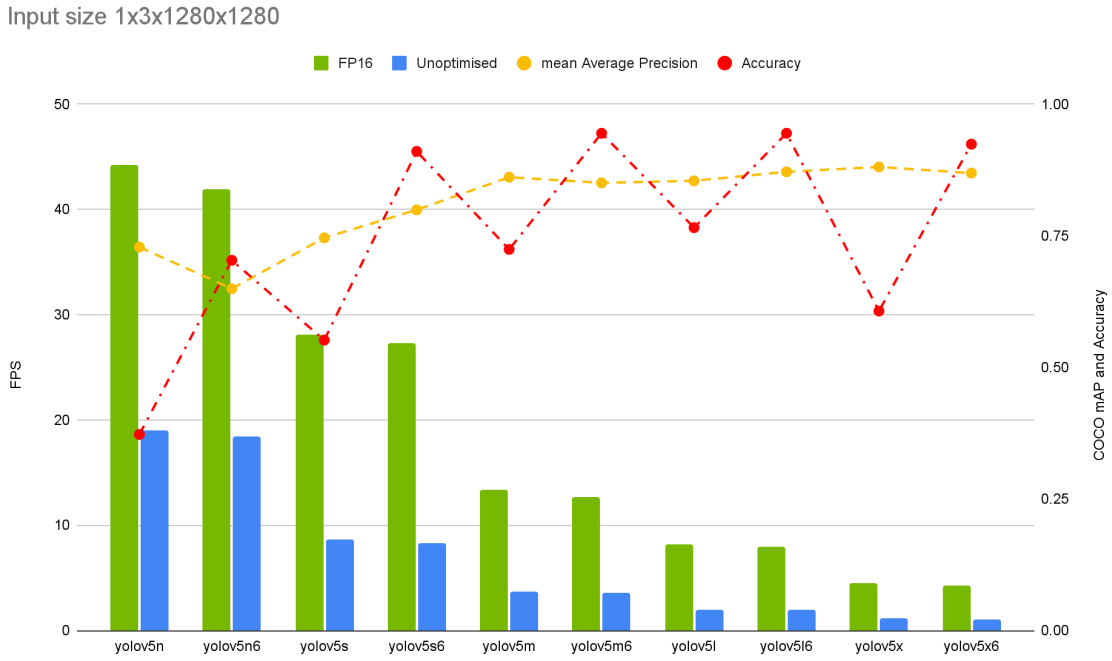

Figure 7.2. Model performance with a single 1280×1280-pixel frame input

The speed difference between the fastest and slowest optimised model is 40 frames per second, with the best speed-up of 4.1×. The mean optimised speed difference is 3.5×. Model accuracy grows up to the YOLOv5m model and there is no significant gain beyond it; from that model onward the speed-to-accuracy ratio decreases.

The mean speed difference between the single-input 640×640 and 1280×1280 models is 36 frames per second.

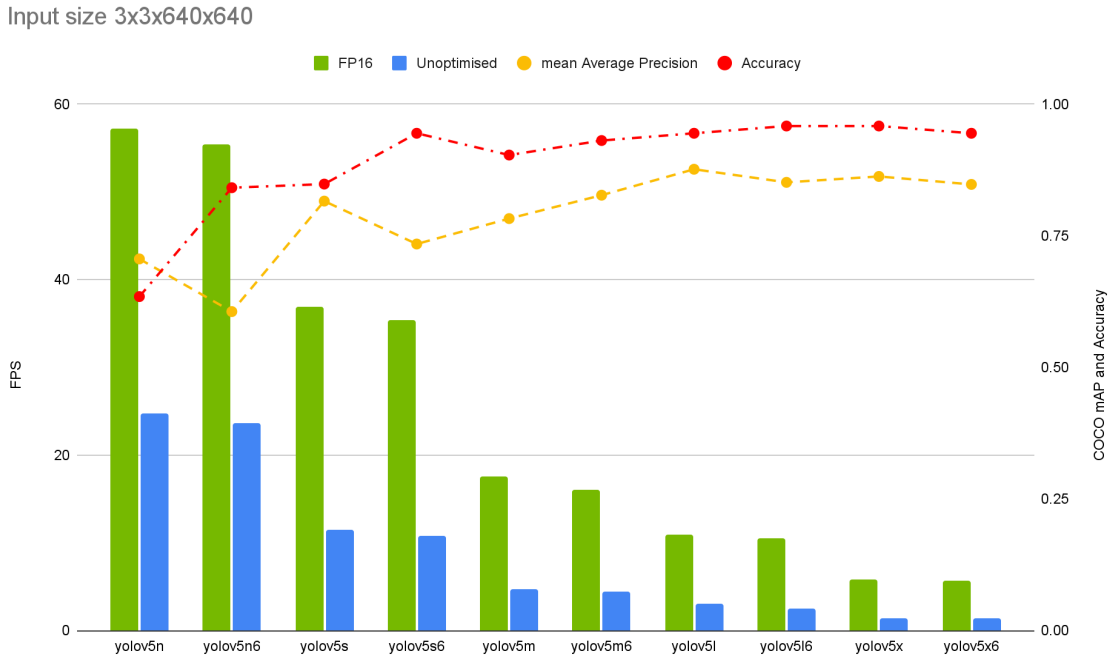

Figure 7.3. Model performance with three 640×640-pixel frame inputs in parallel

The speed difference between the fastest and slowest optimised model is 52 frames per second, with the best speed-up of 4.3×. The mean optimised speed difference is 3.4×. Model accuracy grows up to the YOLOv5l model and there is no significant gain beyond it; from that model onward the speed-to-accuracy ratio decreases.

The three-input 640×640 models are on average 1.4× faster than the single-input models at the same resolution.

The speed difference between the fastest and slowest optimised model is 16 frames per second, with the best speed-up of 4.2×. The mean optimised model speed difference is 3.5×. Model accuracy grows up to the YOLOv5m model and there is no significant gain beyond it; from that model onward the speed-to-accuracy ratio decreases. In this case the models are 1.1× faster than the single-input models at the same resolution.

The mean speed difference between the three-input 640×640 and 1280×1280 models is 18 frames per second.

The three-input models are on average 1.2× faster than the single-input models.

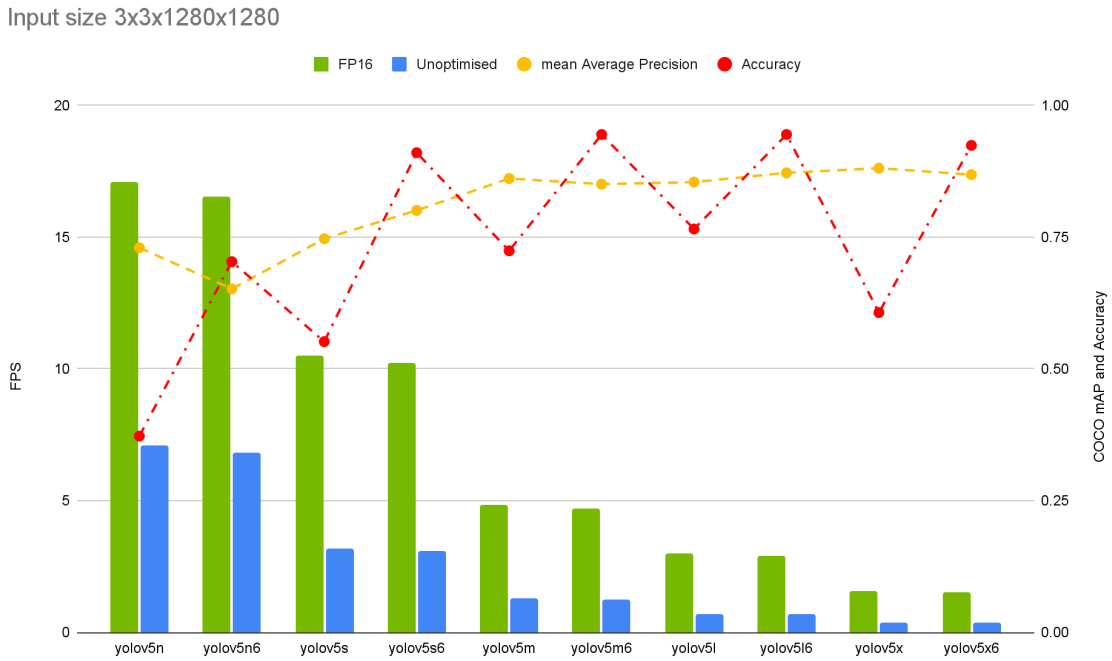

Figure 7.4. Model performance with three 1280×1280-pixel frame inputs in parallel

8 Summary

For the development of the computer-vision system, suitable components were identified that, in the end result, worked well together, ensuring the system's successful, conformant operation in both hardware and software.

In hardware terms, the chosen control unit was the Nvidia Jetson AGX Xavier, which can successfully run larger and more accurate neural-network models with multiple inputs. The hardware of this controller also allowed high-throughput model acceleration, which substantially contributed to the rate at which the computer-vision application processes frames received from the camera. The Arducam Fisheye module chosen as the camera also contributed to more accurate object detection by delivering information at the required resolution to the control unit.

On the software side, containerising the entire application played a key role. It made development and running in the production environment easier, since the application could be run on different computer systems without modifications to the system software itself.

The most important component of the computer-vision software was the chosen object-detection method, i.e. the neural-network models built for computer vision. From among them, YOLOv5 by Ultralytics was selected. It was the best compared with the other models available, both for its unified object-detection-and-classification architecture and for the best ratio of speed to accuracy. It was also one of the few models available that could be successfully accelerated in hardware.

The most important results from the computer-vision model experiments:

- Computer-vision model accuracy grows up to the YOLOv5m / YOLOv5l models.
- The mean accuracy change between optimised and unoptimised models was small ($1.7 \cdot 10^{-3}$).
- The mean model speed-up after optimisation was $3.5\times$.

- The best optimised model's speed gain compared with its unoptimised version was 4.3×.

When optimising the computer-vision application, smooth and accelerated application performance was achieved; the optimisation also significantly reduced the control unit's energy consumption (by 26.8%) and as a result the operating temperatures were lower. Consequently, data processing and the neural network's operation also became significantly faster, and all of the control unit's hardware accelerators were put to good use.

References

[1] *Blockage of the Suez Canal, March 2021*. URL: https://porteconomicsmanagement.org/pemp/contents/part6/port-resilience/suez-canal-blockage-2021/.

[2] *Drowning, WHO*. URL: https://www.who.int/news-room/fact-sheets/detail/drowning (visited on 10/28/2021).

[3] *Marine Insight, 7 Dangerous Diseases/Disorders Seafarers Should Be Aware Of*. URL: https://www.marineinsight.com/marine-safety/7-dangerous-diseasesdisorders-seafarers-should-be-aware-of/ (visited on 10/28/2021).

[4] *Tesla, Future of Driving*. URL: https://www.tesla.com/autopilot (visited on 01/10/2023).

[5] *Lucid Announces Integration of Its Proprietary DreamDrive Pro Advanced Driver-Assistance System with NVIDIA DRIVE Hyperion Software-Defined Platform*. URL: https://ir.lucidmotors.com/news-releases/news-release-details/lucid-announces-integration-its-proprietary-dreamdrive-pro (visited on 01/10/2023).

[6] *Waymo, The World's Most Experienced Driver*. URL: https://waymo.com (visited on 01/10/2023).

[7] *Nvidia Drive, End-to-End Solutions for Autonomous Vehicles*. URL: https://developer.nvidia.com/drive (visited on 01/10/2023).

[8] *Comma.ai, Openpilot: An open source driver assistance system*. URL: https://github.com/commaai/openpilot (visited on 01/10/2023).

[9] *Kongsberg, Autonomous Shipping*. URL: https://www.kongsberg.com/maritime/support/themes/autonomous-shipping/ (visited on 10/20/2021).

[10] *Rolls Royce, Autonomous Ships*. URL: https://www.rolls-royce.com/~/media/Files/R/Rolls-Royce/documents/%20customers/marine/ship-intel/rr-ship-intel-aawa-8pg.pdf (visited on 10/20/2021).

[11] *Täisskaalas laevade autonoomsed käigukatsed avavees*. URL: https://www.scc.ee/taisskaalas-laevade-autonoomsed-kaigukatsed-avavees/ (visited on 02/14/2022).

[12] *Navy 18 WP – Baltic WorkBoats*. URL: https://bwb.ee/vessel/navy-18-wp/ (visited on 03/27/2022).

[13] *Laevade eelseadistatud autonoomsete avaveekatsete tehnoloogia arendamine*. URL: https://www.etis.ee/Portal/Projects/Display/fde46bb2-4579-42ad-bbe9-5bc7c1bef77d (visited on 03/31/2022).

[14] *What is computer vision?* URL: https://www.ibm.com/topics/computer-vision#:~:text=Computer%20vision%20is%20a%20field,recommendations%20based%20on%20that%20information. (visited on 04/04/2022).

[15] *A Comprehensive Guide to Convolutional Neural Networks*. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53 (visited on 05/02/2022).

[16] *Rectified Linear Unit, DeepAI*. URL: https://deepai.org/machine-learning-glossary-and-terms/relu (visited on 05/13/2022).

[17] *Multi-Class Neural Networks: Softmax*. URL: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax (visited on 05/14/2022).

[18] *Convolutional Neural Networks*. URL: https://www.ibm.com/cloud/learn/convolutional-neural-networks (visited on 05/15/2022).

[19] *Technopedia, Single-Board Computer*. URL: https://www.techopedia.com/definition/9266/single-board-computer-sbc (visited on 12/06/2021).

[20] *Nvidia Embedded AI Systems*. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/ (visited on 12/06/2021).

[21] *Raspberry Pi, Model 4B datasheet*. URL: https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf (visited on 12/06/2021).

[22] *Graphics Processing Unit*. URL: https://graphics.fandom.com/wiki/Graphics_processing_unit (visited on 12/06/2021).

[23] *CUDA FAQ*. URL: https://developer.nvidia.com/cuda-faq (visited on 02/11/2022).

[24] *Jetson Modules Technical Specifications*. URL: https://developer.nvidia.com/embedded/jetson-modules (visited on 01/22/2022).

[25] *Nvidia Volta Architecture*. URL: https://www.olcf.ornl.gov/wp-content/uploads/2018/12/summit_workshop_Volta-Architecture.pdf (visited on 03/12/2022).

[26] *Jetson AGX Xavier*. URL: https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/ (visited on 02/09/2022).

[27] *NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics*. URL: https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/ (visited on 02/10/2022).

[28] *SurveilsQUAD - Sony IMX290 Synchronized Multi-Camera System*. URL: https://www.e-consystems.com/nvidia-cameras/jetson-agx-xavier-cameras/sony-starvis-imx290-synchronized-multi-camera.asp#key-features (visited on 02/07/2022).

[29] *OpenCV AI Kit: OAK—D-PoE– OpenCV.AI*. URL: https://store.opencv.ai/products/oak-d-poe (visited on 02/07/2022).

[30] *OAK-1-PoE — DepthAI Hardware Documentation 1.0.0 documentation*. URL: https://docs.luxonis.com/projects/hardware/en/latest/pages/SJ2096POE.html (visited on 02/07/2022).

[31] *Atlas IP67 7.1 MP Model*. URL: https://thinklucid.com/product/atlas-ip67-7-1-mp-imx420/ (visited on 02/06/2022).

[32] *Atlas IP67 2.8 MP Model*. URL: https://thinklucid.com/product/atlas-ip67-2-8-mp-imx421/ (visited on 02/06/2022).

[33] *Arducam 12MP IMX477*. URL: https://www.uctronics.com/arducam-12mp-imx477-camera-and-2-1mp-ov9282-monochrome-camera-with-dm1090ffc.html (visited on 02/08/2022).

[34] *Arducam Fisheye Camera for Raspberry Pi – 5MP OV5647*. URL: https://www.arducam.com/product/arducam-raspberry-pi-camera-5mp-fisheye-m12-picam-b0055/ (visited on 03/29/2022).

[35] *Mindchip Artificial Captain*. URL: https://mindchip.ee/artificial-captain/ (visited on 01/10/2023).

[36] *OpenCV*. URL: https://opencv.org/ (visited on 04/22/2022).

[37] *Scikit-Image, Image processing in Python*. URL: https://scikit-image.org/ (visited on 04/22/2022).

[38] *SciPy - Fundamental algorithms for scientific computing in Python*. URL: https://scipy.org/ (visited on 04/22/2022).

[39] *Pillow – Python Imaging Library (Fork)*. URL: https://pillow.readthedocs.io/en/stable/ (visited on 04/22/2022).

[40] *GStreamer, Open source multimedia framework*. URL: https://gstreamer.freedesktop.org/ (visited on 04/26/2022).

[41] *Nvidia Video Decoder*. URL: https://developer.nvidia.com/nvidia-video-codec-sdk (visited on 04/26/2022).

[42] *Pandas - Python Data Analysis Library*. URL: https://pandas.pydata.org/ (visited on 04/29/2022).

[43] *cuDF, A Python GPU DataFrame library*. URL: https://docs.rapids.ai/api/cudf/stable/ (visited on 04/29/2022).

[44] *Create production-grade machine learning models with TensorFlow*. URL: https://www.tensorflow.org (visited on 01/10/2023).

[45] *Keras, Deep Learning for humans*. URL: https://keras.io (visited on 01/10/2023).

[46] *PyTorch, Tensors and Dynamic neural networks in Python with strong GPU acceleration*. URL: https://pytorch.org (visited on 01/10/2023).

[47] *scikit-learn, Machine Learning in Python*. URL: https://scikit-learn.org/stable/ (visited on 01/10/2023).

[48] *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. URL: https://arxiv.org/pdf/1506.01497.pdf (visited on 01/10/2023).

[49] *Ultralytics, YOLOv5*. URL: https://github.com/ultralytics/yolov5 (visited on 01/10/2023).

[50] *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. URL: https://arxiv.org/pdf/1801.04381.pdf (visited on 01/10/2023).

[51] *EfficientDet: Scalable and Efficient Object Detection*. URL: https://arxiv.org/pdf/1911.09070.pdf (visited on 01/10/2023).

[52] *YOLOv5: The friendliest AI architecture youĺl ever use*. URL: https://ultralytics.com/yolov5 (visited on 01/10/2023).

[53] *YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection*. URL: https://github.com/AlexeyAB/darknet (visited on 01/10/2023).

[54] *A Forest Fire Detection System Based on Ensemble Learning*. URL: https://www.mdpi.com/1999-4907/12/2/217 (visited on 01/10/2023).

[55] *Yolo-dualdev, Develop and deploy TensorRT optimized YOLO models on Nvidia*. URL: https://github.com/The-Magicians-Code/yolo-dualdev (visited on 05/17/2023).

[56] *Docker, Use containers to Build, Share and Run your applications*. URL: https://www.docker.com/resources/what-container/ (visited on 01/10/2023).

[57] *Docker: Accelerated, Containerized Application Development*. URL: https://www.docker.com (visited on 01/10/2023).

[58] *The PyTorch NGC Container*. URL: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/pytorch (visited on 01/10/2023).

[59] *Machine Learning Container for Jetson and JetPack*. URL: https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-ml (visited on 01/10/2023).

[60] *Nvidia TensorRT*. URL: https://developer.nvidia.com/tensorrt (visited on 01/10/2023).

[61] *Accelerated GStreamer User Guide*. URL: https://developer.download.nvidia.com/embedded/L4T/r32_Release_v1.0/Docs/Accelerated_GStreamer_User_Guide.pdf?t=eyJscyI6ImdzZW8iLCJsc2QiOiJodHRwczovL3d3dy5nb29nbGUuY29tLyJ9 (visited on 01/10/2023).

[62] *Jetson-Stats*. URL: https://github.com/rbonghi/jetson_stats (visited on 01/10/2023).

[63] *Open Neural Network Exchange*. URL: https://onnx.ai (visited on 01/10/2023).

[64] *COCO, Common Objects in Context*. URL: https://cocodataset.org (visited on 01/10/2023).

[65] *YOLOv5, Pretrained checkpoints*. URL: https://github.com/ultralytics/yolov5#pretrained-checkpoints (visited on 01/10/2023).

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Tanel Treuberg

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "Improving situational awareness of autonomous vessels using computer vision", supervised by Heigo Mõlder and Karl Janson
    1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
    1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright
2. I am aware that the author also retains the rights specified in clause 1 of the nonexclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

2023

[1]The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive licence shall not be valid for the period.