CHALMERS
UNIVERSITY OF TECHNOLOGY

Real-time characteristics of marine object detection under low light conditions

Marine object detection using YOLO with near infrared camera

Master's thesis in Master Program Embedded Electronic system design

Emil Emanuelsson, Lin Wang

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:Spring

Real-time characteristics of marine object detection under low light conditions

Marine object detection using YOLO with near infrared camera

Emil Emanuelsson, Lin Wang

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
Gothenburg, Sweden 2020

Real-time characteristics of marine object detection under low light conditions
Marine object detection using YOLO with near infrared camera
Emil Emanuelsson, Lin Wang

Supervisor: Ola Benderius, Department of Mechanics and Maritime Sciences
Examiner: Ola Benderius, Department of Mechanics and Maritime Sciences

Master's Thesis 2020:81
Department of Mechanics and Maritime Sciences Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Digitaltryck
Gothenburg, Sweden 2020

Real-time characteristics of marine object detection under low light conditions
Marine object detection using YOLO with near infrared camera
Emil Emanuelsson, Lin wang
Department of Mechanics and Maritime Sciences
Chalmers University of Technology

Abstract

This work discusses how a near infrared camera can be used to detect objects in a marine environment. The goal is to identify marine objects in real-time under low light conditions using object detection algorithm Yolo v3. Some different image processing methods were analyzed, such as saliency, edge detection and convolutional neural networks (CNN). Then the implementation of a scalable collision avoidance was presented. The system uses Intel Realsense camera, OpenDLV software framework, and the Linux operating system. For this work, an OpenDLV interface was implemented for the camera, and OpenDLV perception microservice was used to run its built-in Darknet-based Yolo v3 implementation.

Then the real-time characteristics of the system and the performance were evaluated. It was proven that the system does not have real-time characteristics because of the underlying OS and sensor communication protocol. The system achieved 0.71 mean average precision (mAP) on boats with the test images. It was concluded that the system still need more complete training and testing. Finally, a suggestion on how to implement a similar system with real-time capabilities was given. This includes changing camera, OS and some part of the software that was used.

Keywords: object detection, CNN, neural network, NIR, camera, RTS, Yolo v3.

Acknowledgements

We want to thank Volvo Penta for giving us the opportunity to do this thesis at their company and for all the guidance they have given us. We also want to thank Ola Benderius, our supervisor and examiner from Chalmers University of Technology who has helped us with both technical guidance and lots of input for the report.

Emil Emanuelsson and Lin Wang, Gothenburg, June 2020

Contents

List of Figures

List of Tables

1

Introduction

Night vision technology exists to extend human activity beyond natural light for surveillance, monitoring, or low-light detection [5]. In this work, the focus was low-light detection using image processing, examples of other ways of achieving low-light detection are radar detection and multi-sensor fusion.

Collision avoidance is widely implemented in modern road vehicles but the marine environment adds different challenges. For example, compared to a road vehicle, a marine vehicle is harder to control, both in lateral and longitudinal directions. Secondly, traffic organization is not as strict in marine environments as in road networks. Furthermore, perception algorithms might be harder to be implemented due to water reflections and water fluctuations, especially surface motions. Collision avoidance can be divided into two kinds, independent and collaborative. The collaborative way of it usually involves communications between vehicles. For example, by broadcasting the current coordinates and predict the future position can be an effective way to avoid collisions. However, there are certain limitations, as it only works with the vehicles that have the corresponding system, and it does not work with the collision between vehicles and other objects. Another way is more independent, it uses sensors to build an image of the objects around the vehicle, then identifies the objects in the image in order to avoid the collision between the vehicle with any object.

For a collision avoidance system to be considered formally safe, it is vital that it is designed both in terms of hardware and software, as a real-time system. This can be done by putting formal constraints on system reaction times on all system layers. If the system does not comply to such formal real-time constraints, it might, in some rare cases, fail to properly react to collision scenarios by missing to process sensor data, or by failing to send actuation signals to the vehicle. Even though lack of action is the most likely outcome of non-compliance to real-time conditions, one might even imagine cases such as delayed action or behavior oscillation, that might in turn lead to a increased scenario severity. Apart from the system performance, formal real-time compliance is also a vital component for formalized system verification [6].

This work built a real-time collision avoidance system for marine application under none-ideal lighting environment.

1.1 Scope and limitations

The goal of the work was to design and implement a prototype of a collision avoidance system with both real-time and night vision capabilities, intended for marine applications. In addition, the system should be able to detect collision threats both above and under water. The focus of the prototype was the detection of collision scenarios, and therefore the work will not consider object tracking or vehicle actuation.

1.2 Problem questions

The following research questions were investigated as part of this work:

RQ1 How can NIR or IR cameras be integrated in an embedded Linux-based computer system in an instrumented marine vehicle platform?

RQ2 Using this system, can an algorithm be designed to detect and identify other marine vehicles up to 20 m distance from the camera?

RQ3 Given the general software and hardware requirements above (especially connected to low light conditions), processing data from the camera image sensor with a fixed exposure time (rolling or global shutter) to a stream of identified objects (bounding boxes), how well can it be proven to fulfil formal real-time characteristics suitable for a collision avoidance system?

1.3 Thesis outline

The next section provides the current technical background related to this work, including sensor technology, machine learning, computer vision, image processing and marine applications. Then the third section reviews the basic theory used, It includes camera technology, infrared technology, neural network. After the theory, the fourth section explains the methods used and how design decisions were made. The section is divided into seven parts: camera calibration, system integration, algorithm selection, software platform, data collection, model training and test methodology. Then the fifth section presents the results of this work. It elaborates the overview of the system, also shows the results in both accuracy and real-time aspect. In the sixth section, the limitations and potential improvements of this work are discussed. Finally in the seventh section, the work is summarized and the answers to the research questions are evaluated.

2
Background

This section provides the background for this work, first a brief introduction to sensor technology, then some relevant machine learning, computer vision, image processing techniques, and neural networks are introduced.

2.1 Sensor technology

In Latin, the word *sensor* has its root in *sentire*, meaning *to perceive* [7]. So from the middle English era, sensor started meaning a device that respond to a stimuli received by a sensitive element as input and generate a corresponding output. But now sensor can also mean the sensitive element itself. For most systems, sensors service as the eyes for the system, collecting the sensor-perceived environment data and feed it to the system for future processing. Different sensors are now fusion together in more complex systems to get more detailed information.

Together with the rapid development of operational systems over recent years, sensors are widely used in industry for such systems. The process of generating output from input can be seen as a conversion of energy from one to another form [7]. With different input energy type, sensors can be divided. For example, gravitational sensors use gravitational attraction, mechanical sensors takes motion, mechanical forces or displacement as occurrence, electromagnetic sensors like infrared sensors or UV sensors, are sensitive to electrical charge, current or electromagnetic wave energy.

Safety related systems built base on sensor technology are commonly found in the whole transportation industry. Due to the perceiving ability of sensors, engineers use them to be the *eyes* of the system, providing detailed environment information, so that the features like collision avoidance can be achieved.

2.1.1 Collision avoidance

There exist systems for vehicle collision avoidance [8][9]. For marine vehicles, these are based on radio signal or communication between them. This require all vehicles to have the same equipment which would require either laws or a industry standard to make ideal use of this kind of systems. These systems only avoid collision with other vehicles not any shallows or docks. This work differentiates from such systems by focusing on independent collision avoidance based on sensors. At the same time, it differentiates from similar systems of road-vehicle by being implemented on ma-

rine vehicles.

Collision avoidance systems often consist of many sensors which result in very large images. Object detection in such large images require large amount of computations that is very expensive to perform in the desired time frame. To decrease the computational power needed, Uzkent proposed an algorithm that chooses a low resolution version of the image if the image is dominated by large object and high resolution if it is dominated by small objects [10]. Collision avoidance systems need to be very accurate to protect users, people around the vehicle and the vehicle itself. As Pan Wei writes, there is different strengths and weaknesses for different kinds of sensors. Cameras have poor vision in mist and rain but a LiDAR works as usual in such circumstances. A LiDAR however have no color vision and can be unreliable for long distances [11]. For this reason it is preferred to create collision avoidance systems with different types of sensors.

These goals can be achieved through different algorithms while using a camera image. Some examples being edge detection, saliency and convolutional neural networks.

2.2 Machine learning

The process of a computer learning a specific task is called machine learning [12]. It is a series of artificial intelligence. An old definition for machine learning, according to Arthur Samuel, is "the field of study that gives computers the ability to learn without being explicitly programmed." [13]. Similar to how human learns, the computer uses the features of the learning task to reproduce the result in the future. The systems might improve with experience and time. For example, children learn that the open-top, thin wall container that is usually used to hold liquids for pouring or drinking is a cup. The feature for the cup then is the open-top trait, or that most of the time it is used to drink water from. After some times of seeing the cup, and being told it is a cup, the children are then learn to classify a cup in the future when they see one.

As for machines, if the task is to have the machine do the same with the children—learning to classify cups. Then the computer is first given lots of examples, in another word, the material to learn from. An ideal set of machine learning data should contain the input variables, the features of the cups in this case, and the supervision like a class or label, that tells the computer *this is a cup!*. The computer then start *learning*, by extracting the same features from different examples. Like children extract the characteristic of the cups as top-open container with thin wall. Different from the kinds, the computer needs a classifier function to maps the set of features into labels in the future, in which way the computer *learns* how a cup looks like.

The previous example shows how supervised learning is done. The computer receives pre-labeled input examples and converge to a classifier function. However, there is another area of machine learning with out supervision. Non-supervised learning is

associated to the process of building up models without pre-labeled data but to analyze the similarities among input data [14]. So there will be no classifier function but functions that analyze how points are organized in input data, then the results will be checked manually, therefore, large data does not suit for non-supervised learning algorithm.

Machine learning has many uses. Software use it for learning user's behavior, mailboxes use it for spam detection. Smart phones achieve voice recognition using machine learning, as well as stock trading. It is also widely used in robotics, healthcare, e-commerce, gaming analysis, Internet of things, and so on.

2.3 Computer vision

One big area of machine learning comes with computer vision. It is the artificial intelligence that is trained to understand and make use of the visual data. As human brain transform the two dimensional image projected on our retina into a meaningful three-dimensional world. It is easy for us to separate the object from the background of the scene, make judgement of someone's mood base on the person's face. The purpose of computer vision is to replicate similar action with computers. It is widely used in technologies like activity recognition and object classification.

A common goal of computer vision is to detect objects and map the environment of the device [15]. Computer vision can be applied in many different fields such as automotive, surveillance and medical industry. Object detection can be implemented thought several different methods. Relevant methods of collision avoidance for marine environment are introduced later in this chapter. There exist a number of object detection methods for IR images. Some typical methods uses machine learning, background subtraction, optical flow, frame difference or saliency [16].

2.3.1 Image processing

The first step in most computer vision applications is image processing. As shown in Fig. 2.1a–2.1d, images can be pre-processed so that it is more suitable for the future analysis.

Mapping pixel values from one image to another is the standard image processing operator [17]. When each pixel is manipulated independently it is called point operators. Fourier transform and image pyramid consider the neighbour pixel values when a new pixel comes in. These kind of neighborhood operators are especially useful when the input images have a variety of resolutions. Geometric transformations as global operators are used to analyze images with deformation and rotations.

2.3.2 Edge detection

D. Sangeetha and P. Deepa [18] suggested a Canny edge detection algorithm developed on an FPGA. They stated that Canny edge detection algorithm has poor

(a) The original image

(b) Image with increased exposure

(c) Image with increased contrast

(d) Blurred image

Common image processing operations. image(b),(c),(d) are post-processed image from image(a)

real-time characteristics, and developed their algorithm on an FPGA to combat this defect. If Canny edge algorithm should be used in a real-time critical system, and that the system needs high computational power [19]. the Canny edge algorithm have performance limitations from the Gaussian filter witch is traditionally used in the Canny edge algorithm. This filter is considered to be not sufficient to decrease all types of noise. Other limitations are the high and low thresholds, they are assigned manually which limits the adaptability of the algorithm [20][21]. This result in that conventional canny edge algorithms did not perform as well on IR images from the sea, as IR images from the sea tend to have lower SNR [21]. Because of this, Liu et al. developed an improved canny edge algorithm for IR cameras at sea. They used contrast limited adaptive histogram equalization (CLAHE) filter to improve the image quality, and instead of Gauss filter, they used morphological filtering. They also implemented the improved OTSU method to assign the threshold values. This algorithm improved the suppression of the noise from the sea and thus resulted in less false edges.

2.3.3 Saliency

Saliency algorithms is widely used for object detection for it's accuracy [16]. L. Huo et al. suggested an algorithm for object detection in an RBG image based on

saliency and sparse representation [22]. Their algorithm used superpixels to identify the background and thus also identifies the target. It was shown that the algorithm needs 0.18 second per image on an Intel Core i3-550 3.2 GHz and 4 GB RAM coded in Matlab. Zhang et al proposed an saliency algorithm for infrared moving objects [16]. Their algorithm used local saliency based on the correlation between gray features and motion features. The algorithm has higher precision than other infrared object detection algorithms. It was tested on a PC with 2.8 GHz Intel CPU and 8 GB RAM and a 15 fps image stream.

2.3.4 Convolutional neural network

Convolutional neural network has been around since 1980 and was introduced by K. Fukushima [23][24]. CNN has since been used to recognize various objects in images, such as handwriting on checks. CNN was not used in large scale object detection until 2012 when A. Krizhevsky won the Imagenet objecet classification competition with an algorithm based on CNN [25]. The algorithm is called AlexNet and achieved 15% top-5-error which was 10% better top-5-error than the runner up [26]. Since then, CNN has been the dominating algorithm used for object detection, and been improved massively. Recently the networks has become much deeper and more complex after the implementation of residual units [3]. The top algorithms are now down to 2% to 4% error rate for object classification [26]. The current main challenges for object detection are the bounding box precision, often measured in mean average precision (mAP), and the calculation load [4][26][27].

2.3.5 Night vision

Night vision technology that aids human to see in night condition can be achieved with different approaches. For those that involves imaging systems, the system usually includes three parts: optical objective, image intensifier tube and optical ocular [28]. In this work, these three parts are expected to be integrated in the cameras that are used.

3
Theory

This chapter gives a more in-depth description of the technology and algorithms used in this work. First, night vision technology is introduced implemented by camera technologies. The second part of the chapter provides a brief description of the object detection method used during this work. First an overview of convolutional neural networks (CNN) then a description of the specific algorithm used, Yolo v3.

3.1 Camera

Camera is a common component used in night vision. Due to the limited humanly visible illumination in the environment and it is possible to illuminate the area with IR without disrupting the environment as much IR cameras are usually chosen. With different settings of the camera, the output frame may vary a lot. This section talks about related camera parameters and IR cameras.

3.1.1 Exposure

In photography, total amount of light incident on a sensitive material is defined as exposure [29]. Exposure is the product of the quantity of illumination on the imaging sensor and the duration of the exposure, as defined by Eq. 3.1.

$$H = E \cdot t \tag{3.1}$$

Eq. 3.1 is called the reciprocity equation. Where H is the exposure, E is the illumination and t is exposure time. For night vision technology, due to the limitation of natural light, a light projector can be useful to increase the illumination of the environment, increase exposure time is also beneficial for gaining more information. However, the longer the exposure time, the more time between frames it would be, so finding the optimal combination of illumination and exposure time is crucial for night vision technology, though the optimal combination depends on both the lighting condition and specific scene.

3.1.2 Global and rolling shutter

Image sensor, usually camera, can capture frames with different methods. Global shutter and rolling shutter are two of them. Exposure time, also called shutter speed is an important factor when choosing the frame capture method.

When all pixels are exposed and sampled at the same time, the image sensor is using global shutter [30]. Since all pixels are exposed at the same time there will not be any spatial distortion. However, depending on the exposure time and movement of the object there can still be motion blur. After the exposure, all light across the image sensor is blocked at the same time, while the data is to be sampled and written to the memory, then the sensor will be reset for the exposure of the next image.

Rolling shutter means that the image sensor is samples one pixel row at a time, see Fig. 3.1. The benefit with rolling shutter is that it is possible to have longer exposure time than global shutter with the same fps. The downside is that it can create motion distortion when there is a moving object in the frame, see Fig. 3.2. As in this work the camera is used for machine learning where spatial distortion is unacceptable and the frame is expected to be taken during the same time duration, global shutter is the preferred method.

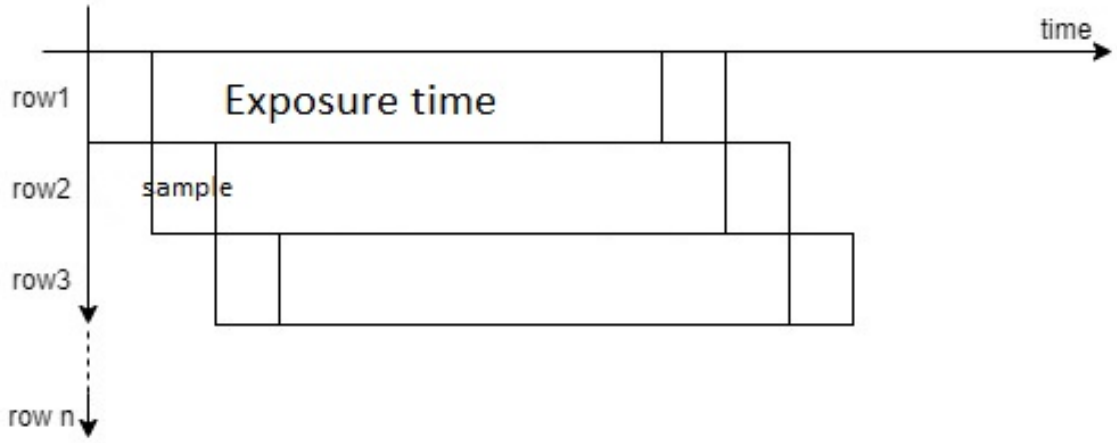


Figure 3.1: Rolling shutter scheme: The time between the image sensor read out the first line and the next line is the exposure time [1].

Figure 3.2: Spatial distortion of a windmill caused by rolling shutter method [2]

3.2 Infrared Technology

3.2.1 NIR FIR camera

NIR and FIR cameras are commonly used in night vision technology. They are distinguished by the difference on the wavelength of the electromagnetic wave. Figure 3.3 demonstrates the definition of electromagnetic wave.

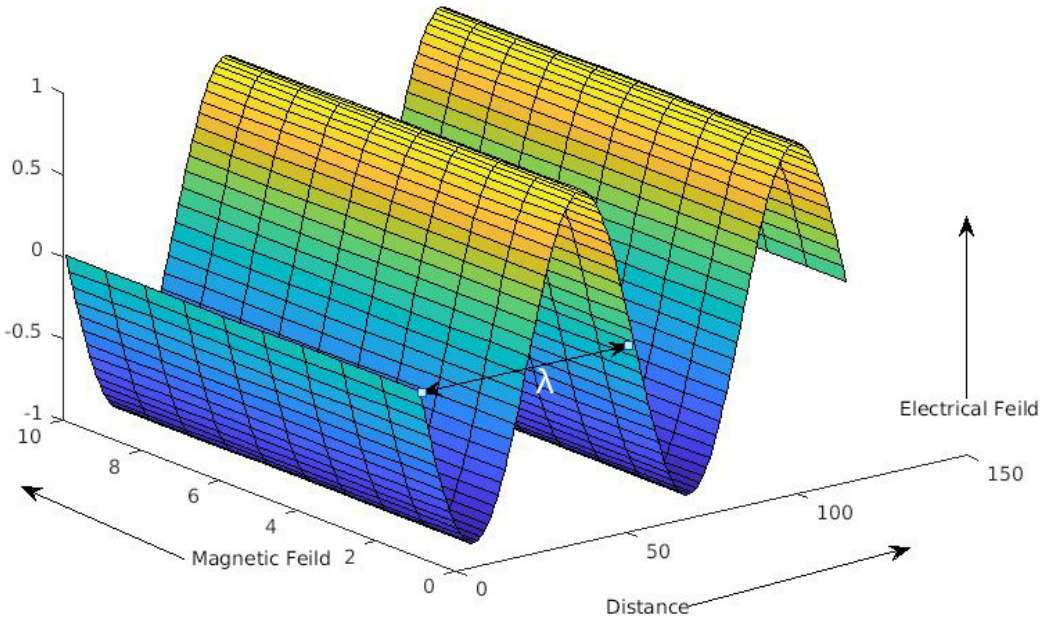


Figure 3.3: Electromagnetic wave: when vibration happens at right angles to the direction of waves traveling, the waves can be defined as electromagnetic waves, λ is the wavelength of the radiation, it is defined as one cycle of the distance of the wave in the travel directioncamera1.

Electromagnetic waves with wavelength between 700 and 3000nm are defined as NIR [30]. NIR image contains more information than what human eyes can perceive, which is a big advantage when it comes to night vision. Also, when adding NIR illumination to the environment targeted by a camera the human eye perception would not be effected. Together with accurate real-time characteristic, NIR cameras are widely used in object detection.

Figure 3.4: IR image of a tea jar in the dark with exposure time of 0.165 seconds, no extra illumination support.

FIR, on the other hand, indicates electromagnetic waves with wavelength between

3 and 40um [30]. FIR cameras are usually used for thermal image creation, and detection for pedestrians and animals in automotive industry.

3.3 Convolutional neural network

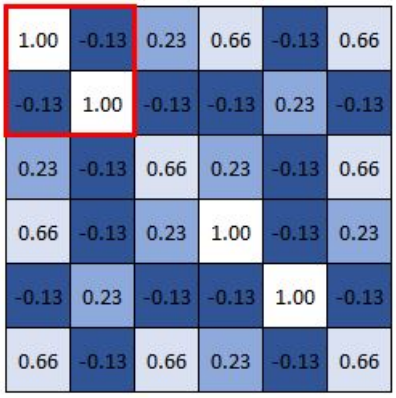
Convolutional neural networks, CNN, are a commonly used method for object detection. CNN based systems has won the ImageNet large-scale visual recognition challenge (ILSVRC) several times in recent years [31][26]. It is a neural network type that mainly uses the convolutional operator in network layers, rather than being fully connected, to extract the features of test samples. After the feature extraction, fully connected layers calculates the probability of an feature in the image. The current state-of-the-art architectures, including residual layers, make it possible to train deeper networks for higher performance.

3.3.1 Convolutional layers

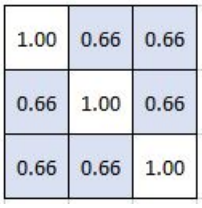
The goal of a CNN is to match a part of the test sample to a feature. To do this, it extracts the features of the training set. Convolutional layers are the main calculating layer of a CNN for features extraction. Convolutional layers consist of a stack of three dimensional matrix filters with learnable parameters [32]. The matrix size is most commonly 3x3x3 and 1x1x3, though 5x5x3 and 7x7x3 are also common. The third dimension depends on how many channels are used. Most commonly three channels, for example RGB images. The filters are slid over all of the images, usually using one pixel steps. Thus the output of each filter is a two dimensional image with slightly reduced height and width of the input image. If a 3x3 filter is used, the output will be two rows and columns smaller if no padding is used. The step size can be larger than one pixel, the step size affects the size of the output. The output of the layer is an image with input width and height, the depth depends on how many filters are used in the layer. Padding is often used as a tool to enable the filters to be applied on the edges of the image, thus not losing any information on the edges. Padding is implemented as dummy pixels around the image, the amount of padding depends on the size of the filters used, for example 1 bit for a 3x3 filter.

3.3.2 Pooling

After the convolutional layers, one image becomes a stack of filtered images, to shrink the image stack, pooling is needed. By setting a window size, usually 3x3, and a stride, usually 2. Then move the window across the filtered images, taking the corresponding value to represent the window.(Max pooling is to take the maximum value of each window, shown in Fig. 3.5). Then every window becomes one pixel, thus the image stack is reduced.



(a) Filtered Image after convolutional layers



(b) Shrink image

Figure 3.5: Max pooling. The red triangle in panel (a) is the window, the size in this case, is 2*2. Max pooling tasks the max value in each window. Panel (b) shows the image after max pooling. It can be easily seen that the pattern of the image is still kept thought the size is reduced.

3.3.3 Activation function

Every neuron created by the convolutional layers is going to be activated from the input. Common activation functions include sigmoid and rectified linear unit (ReLU). For sigmoid, when the input is close to 1, the output is 1, when the input is -1, the output is 0, when the input is 0, the output is around 0.5. For ReLU, all negative input will have an output as 0, while positive inputs stay as they are.

3.3.4 Fully connected layers

The main difference between convolutional layers and fully connected (FC) layers is that a neuron in FC is connected to all neurons in the previous layers while in convolutional layers they are only connected to local neurons, size of the filter. Because of this FC has many more parameters to learn than the convolutional layers. An CNN usually includes a few FC layers in the end of the architecture to compute the probability of classes.

3.3.5 Residual units

When linear CNN starts to become very deep, more than 20 layers, the performance starts to degrade. This means that a network with 40 layers with similar architecture as a network with 15 layers can have more errors than the shallower network [3]. Then residual units can be used to combat the degradation. Residual units use forward feed from previous layers, see Fig. 3.6. The residual unit can be described as $y = F(x, W_l) + x$. Where x and y are the input and output, W_l is the weights associated with the convolutional layers. The residual units add the forwarded layer element wise, thus the layers need to have the same dimension. The layers can be

feed forward any number of layers, ResNet uses stacks of 2 residual unit with two convolutional layers [3].

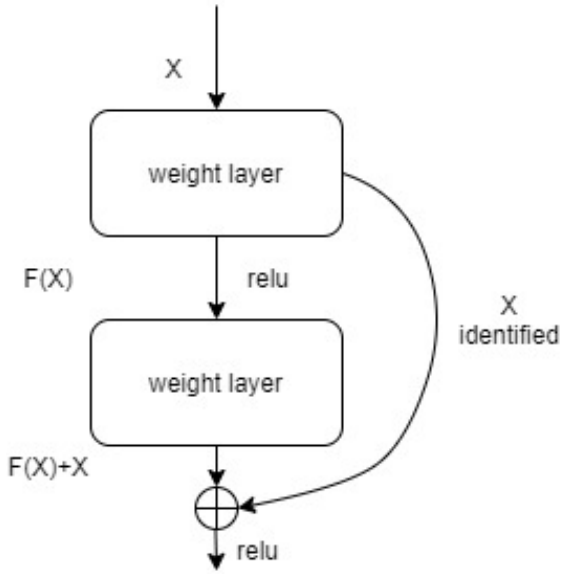


Figure 3.6: Residual unit example [3]

4
Method

In this chapter, the methods used in this work is described. First, the complete system is demonstrated, then how the camera is calibrated is presented, followed by each part of the system and how they were integrated. Finally, training and testing methodologies are explained.

4.1 System overview

The complete system is shown in figure 4.1. The camera sends the image through USB to the PC where the Linux kernel and a modified video for linux (V4L) kernel receives the image. The camera sends an image in Y8 format to the PC. After the image is received, it can be accessed by the Intel librealsense SDK. With the SDK, an OpenDLV camera interface receives the signal. The interface converts the image to I420 and ARGB format, and stores it in two shared memories. An OpenDLV perception microservice then fetches the image from the ARGB shared memory. The OpenDLV perception microservice uses Yolo on the fetched image to identify any marine vehicles in the image and highlights them. Finally the processed image is displayed to the user.

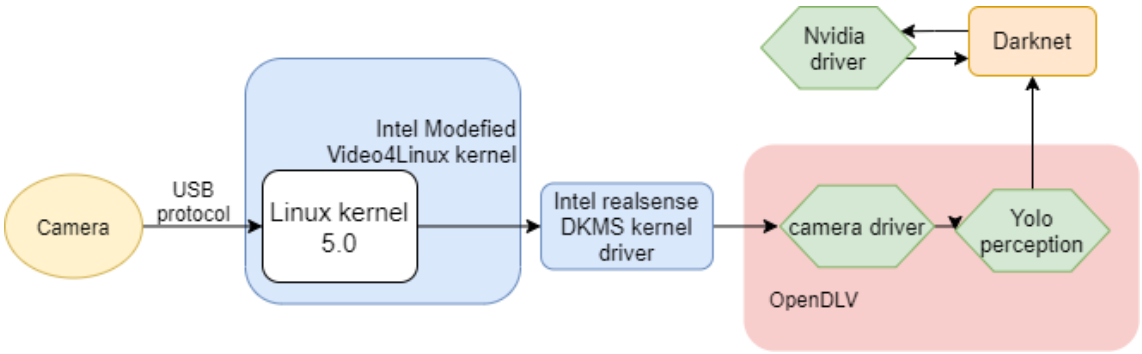


Figure 4.1: System overview

4.2 Camera calibration

The camera settings that need to be tested are the exposure time and the gain. The exposure time is desired to be as short as possible. This is because the shorter exposure time used, the higher FPS is available. With long exposure time, relatively strong light sources in the image appear larger than they actually are. As most

maritime vessels have some light sources on them, the features of the vessel might be covered by the light source when using to long exposure. The gain is used to amplify the image and thus enhance the features in the image. The gain has no desired limits, for the camera used in this work, the minimum gain is 16 and maximum 248. The exposure times used in the tests are 15ms, 30 ms and 60 ms, this enables FPS of 30 and 15. More detail of the tests can be seen in Table 4.1.

Test number	Exposure time (ms)	Gain
1	15	16
2	15	50
3	15	80
4	15	110
5	15	140
6	15	170
7	15	210
8	15	248
9	30	16
10	30	50
11	30	80
12	30	110
13	30	140
14	30	170
15	30	210
16	30	248
17	60	16
18	60	50
19	60	80
20	60	110
21	60	140
22	60	170
23	60	210
24	60	248

Table 4.1: Used values for gain and exposure time for tests

4.3 OpenDLV

OpenDLV is a Chalmers developed microservice software framework for robot control. It is created with the header only library libcluon, which enables shared memory and communication between different microservices. This feature enables easy scaling of the system and add functions to the system by adding a new or a copy of a microservice. For this work, that enabled the system to be tested with several cameras and adapted to different marine vehicles types and models. The microservices in OpenDLV run inside Linux Namespace, typically using Docker, which enables easy portability as the microservices can be run on any system with the same kernel

without installing dependencies. Docker also makes the system language independent.

The system designed for this project are built on top of OpenDLV framework. The units inside works as separate microservices.

4.4 OpenDLV integration of the camera

The camera was integrated into a Linux based system where the images are processed by the object detection algorithm. To integrate the camera, an interface for OpenDLV was written in C++. A Docker image was then built from the OpenDLV camera interface to be able to scale the system easily to include several cameras.

The interface first fetches the relevant options supplied by the user when starting the interface. These include options such as FPS, which camera to start, resolution, shared memory name, etc. Then the shared memories and camera are initiated. The camera is controlled by functions supplied in the *librealsense* library. The camera is set to only supply one infrared stream out of two available. After initiation the interface enters a loop that handles the data from the camera. In the start of the loop the most recent image is fetched. The images from the camera use the Y8 pixel format, this is a format that neither OpenDLV or Yolo support thus it is converted to I420 and optionally to ARGB. The converted image is then stored in a shared memory where an OpenDLV container can access, for this system, a Yolo container should access the image.

The conversion from Y8 to I420 format is done by copying the Y bits from each pixel in Y8 format to corresponding I420 pixel. I420 uses U and V bits that Y8 does not use, these has to be assigned 128 to each byte of U or V bits. I420 uses one byte each for U and V values for two pixels. Thus, the amount of U and V bytes is width times height divided by two. The I420 format is used as base to convert to ARGB pixel format. The C++ library *libyuv* supplies a function for conversion from I420 to ARGB.

4.5 Yolo

When developing CNN system,s the main focus has been to make them more accurate. This has made the systems larger and more complex, many state-of-the-art architectures use 50–65 million parameters [33]. But in recent years the interest of real-time embedded object detection systems has increased. Some existing architectures with potential for live object detection are Faster R-CNN, Tiny Yolo v2, Yolo v3 and Yolo v3 nano [34].

Faster R-CNN uses two different stages. One stage where the regions of interest, ROI is identified and one where the objects are classified [35]. Yolo has a more linear architecture than faster R-CNN. This enables Yolo's run-time to be more

predictable.

Algorithm Name	Size	mAP (VOC 2007)	Ops (VOC 2007)	mAP (COCO)	FPop/s (COCO)	FPS
Tiny Yolo v2 [34][35]	60.5 MB	57.1%	6.97B	-	-	
Tiny Yolo v3 [34][35]	33.4 MB	58.4%	5.52B	33.1%	5.56B	220
Yolo nano [34]	4.0 MB	69.1%	4.57B	-	-	-
Yolo v3 416x416	-	-	-	55.3	65.86B	35
Faster R-CNN [35]	-	69.9%	-	41.5	-	0.5

Table 4.2: Comparison of object detection algorithms.

Yolo combines the two stages of object detection into one neural network instead of having separate network for the two tasks [36]. The system divides the input image into a SxS grid. Each grid cell predicts bounding boxes and confidence score of objects in the cell. The confidence score is defined as the probability of an object in the box times the IOU with the ground truth. If there is an object in the cell the desired score should be equal to the intersection over union (IOU) between the ground truth and the predicted box. If there is an object that spans several cells the cell that is in the center of the object is responsible to detect it. The bounding boxes are defined by 5 values, x, y, w, h and confidence score. The x and y values are the coordinates (x,y) of the center of the bounding box. The w and h values are width and height of the bounding box and the confidence is the IOU between the ground truth and the bounding box. The grid cell also predicts one class probability set, no matter how many bounding boxes there are in the cell. Finally, to classify the object, the class probability and bounding box confidence are multiplied to give a class-specific score for each bounding box [36].

Yolo v3 uses a feature extractor with 53 convolutional layers called Darknet-53. It is a hybrid of the network used in Yolo v2, Darknet-19, and residual networks [4].

4.6 Data collection and training

Two data sets were used, one recorded in Gothenburg for this project and the other is the NIR subset from the Singapore Maritime Dataset [37]. The Gothenburg dataset was recorded with the camera used in this work from the shore of Götaälv and Saltholmen harbour with varying light conditions. The city light in the background during the night is a potential limitation for the result of this work.

When training the weight file, a model for Darknet-53 that is pre-trained on imagenet was used as base model. Since the purpose of this work was to identify marine vehicles for night vision, with pre-trained classes, this work could benefit from the training for marine vehicles before. It made the training process shorter and more

	type	filters	size	output
	Convolutional	32	3x3	256x256
	Convolutional	64	3x3/2	128x128
	Convolutional	32	1x1	
1x	Convolutional	64	3x3	
	Residual			128x128
	Convolutional	64	3x3/2	64x64
	Convolutional	64	1x1	
2x	Convolutional	128	3x3	
	Residual			64x64
	Convolutional	256	3x3/2	32x32
	Convolutional	128	1x1	
8x	Convolutional	256	3x3	
	Residual			32x32
	Convolutional	512	3x3/2	16x16
	Convolutional	256	1x1	
8x	Convolutional	512	3x3	
	Residual			16x16
	Convolutional	1024	3x3/2	8x8
	Convolutional	512	1x1	
4x	Convolutional	1024	3x3	
	Residual			8x8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 4.3: Architecture of Darknet-53 [4]

effective.

Yolo achieves machine learning using supervised learning. As mentioned in Sect. 2.2, supervised learning requires labeled images with bounding boxes. More specifically, for training Yolo, each images need a corresponding text file with the information about the relevant objects in the image. The text file includes all objects' class index and box's relative coordinates in separate lines for each image as shown in Fig. 4.2. A software was used to generate the text file for each picture.

Before the training, Yolo v3 configuration file was also modified in order to define the iteration times and classes. Files that specify the training data and classes list

```
[8 0.057813 0.872917 0.081250 0.045833]
[<object-class> <x\_center> <y\_center> <width> <height>]
```

Figure 4.2: Example of text file used for training Yolo.

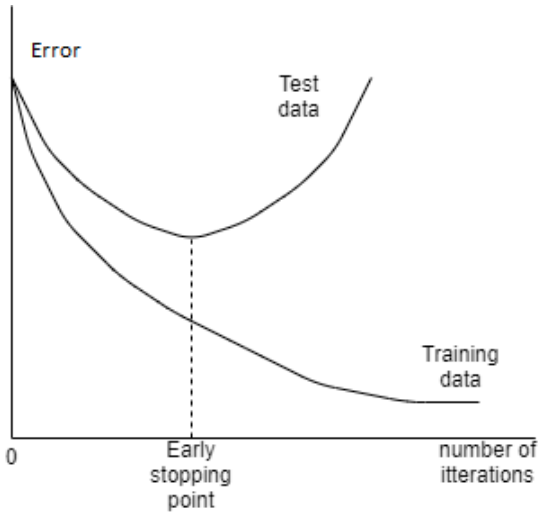


Figure 4.3: Error over number of iterations with given data set

were also created.

While training, several versions of weights were saved in different stages of the training. The different weights were then tested on the test data to find the best one as the weights with most training iterations was and are most likely over-fitted. If the training would avoid the over-fitting, it would risk training the model less than ideal. Over-fitting refers to the situation when the model is too closely fit to the training data that the model fails to classify the object that it has been trained for [12]. The model can then only detect objects on the training image, but not general objects in the same class. The relationship between performance of the model on test data and training data to number of iterations are compared in figure 4.3.

4.7 Test methodology

Due to limitations, the test was not done in the live system, but with pre-recorded videos taken by the OpenDLV microservice. The video images were not used for training and is recorded with the same light condition. Yolo v3 with re-trained weights are used to test if the model can recognize desired object in certain environment.

5

Results

In this section the results of the tests and a software evaluation are presented. First the results from camera settings test are shown. Following the camera settings, the training results, and finally the process times of the system are presented.

5.1 Camera settings

The results from the camera setting test can be seen in figure 5.1a. The target was an stationary ferry approximately at about 50m distance. As seen in the figures the camera image got more noisy with higher gain and exposure time. The images in figure 5.2c and 5.2i are deemed to be the best, as the details of the targeted marine vehicle are easier to extract compared to the other ones. However, no image is ideal, the most detailed images are too sensitive to strong lights and still doesn't supply strongly defined details. Thus it was decided to use the cameras auto exposure function for this work.

(a) Exposure time 15 ms, gain 16

(b) Exposure time 30 ms, gain 16

(c) Exposure time 60 ms, gain 16

(d) Exposure time 15 ms, gain 50

(e) Exposure time 30 ms, gain 50

(f) Exposure time 60 ms, gain 50

(a) Exposure time 15 ms, gain 80

(b) Exposure time 30 ms, gain 80

(c) Exposure time 60 ms, gain 80

(d) Exposure time 15 ms, gain 110

(e) Exposure time 30 ms, gain 110

(f) Exposure time 60 ms, gain 110

(g) Exposure time 15 ms, gain 150

(h) Exposure time 30 ms, gain 150

(i) Exposure time 60 ms, gain 150

(j) Exposure time 15 ms, gain 180

(k) Exposure time 15 ms, gain 180

(l) Exposure time 15 ms, gain 180

(m) Exposure time 15 ms, gain 248

(n) Exposure time 15 ms, gain 248

(o) Exposure time 15 ms, gain 248

5.2 Training results

The first training used only 50 images from Göta älv in a very dark envirionment. The resulting mAP was 0.243 and learning rate of 0.001. Figure5.3 shows the detection before and after the training. The pre-trained weights does not detect anything on the image. Whereas on the predication with post-trained weights, two pink bounding boxes are put around marine vehicles on the image with none-ideal light condition with confidence score of 81% and 78%. It shows the trained model obtained the ability to detect marine vehicles in none-ideal light environment.

(a) Prediction with pre-trained weights file

(b) Prediction with post-trained weights file

Figure 5.3: Prediction before and after training

Second, more complete training was done with 500 training images and 70 test images. This training was performed with 18000 iterations. The best test result was found after 10000 iterations where the mAP was 0.71 on the test data. Examples of

predictions can be seen in figure 5.4. When testing with the training data the mAP was 0.95 after 10000 iterations and 0.97 after 18000 iterations.

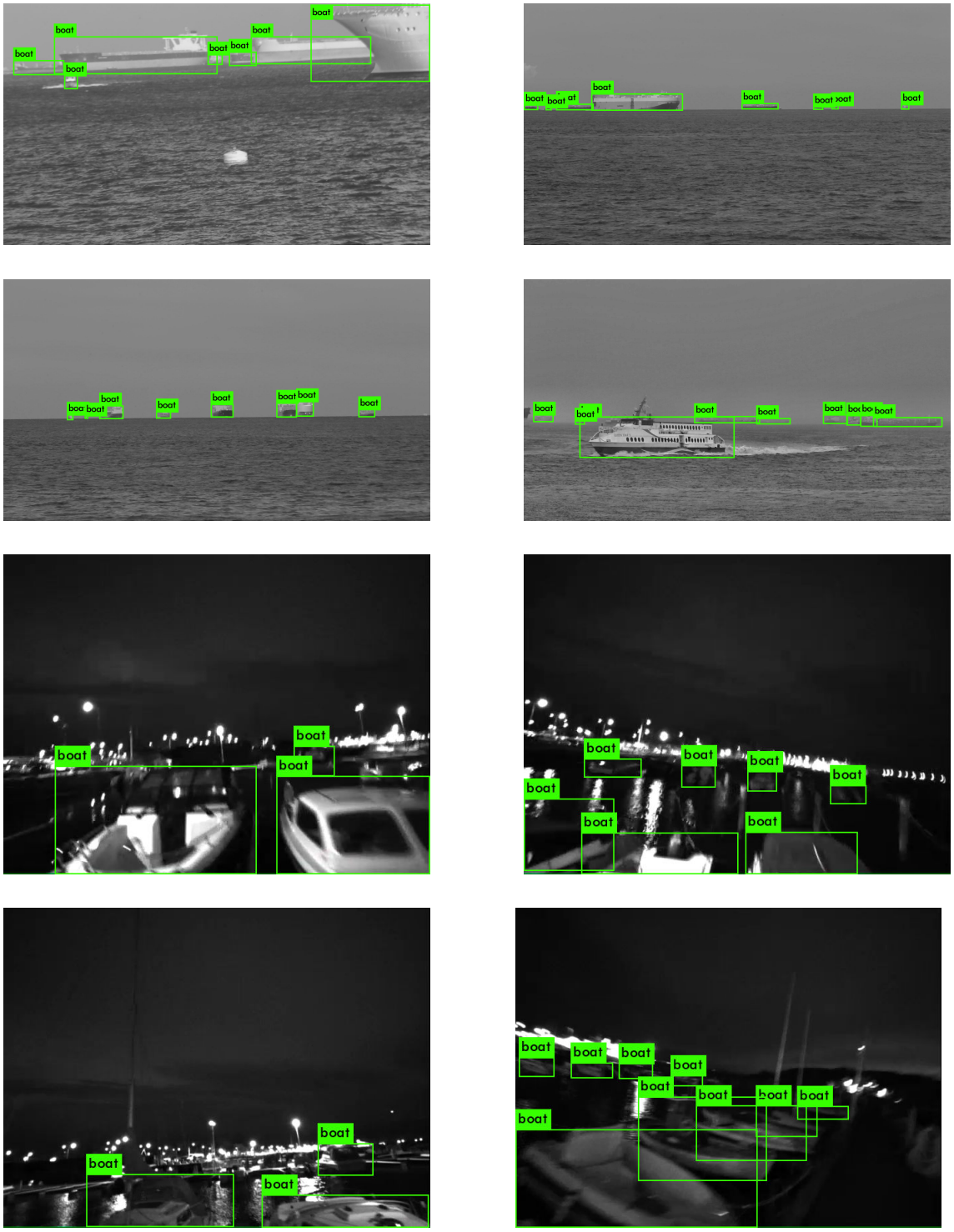


Figure 5.4: Prediction results after the training

5.3 Process time

The real time characteristics of the system was tested by measuring the execution time of the different processes separately. The OpenDLV camera interface was measured by taking a timestamp after an image was received and one after the conversions to I420 and ARGB pixel format was performed. Then the timestamps was printed to a text file. The text file was then processed by a Python script to calculate the process times and extract the longest process time. The results can be seen in Table 5.1 below

Process	Max Time	Mean time
OpenDLV camera interface	883 us	410 us
OpenDLV YOLO	55232 us	47092 us
Rendering images	6137 us	2846 us

Table 5.1: Process times of the various components of the system.

The camera feed was set to 15 fps. When worst case scenario occurs, fully serial computation and slowest processing times, the system can achieve 16 fps. The process times were measured using Intel(R) Core(TM) i7-9800X @ 3.80 and Nvidia Quadro RTX 4000. The resulting schedule can be seen in figure 5.5

Figure 5.5: Schedule of a worst case cycle of the system. Green = Camera interface, Black = Yolo and red = Rendering.

6
Discussion and improvements

In this chapter, the evaluation for the system's real-time characteristics are elaborated. Some potential improvements of the work are also presented, followed by possible future work.

6.1 Real-time analysis

With the result presented in section 5.2, the process times can be considered to be fast enough for most marine implementations. When traveling in 50 km/h or 27 knots, which is a high speed for most marine vehicles, the vehicle travels almost 1 m between each frame. This distance should not make any difference if an object appears between two frames. This however, does not prove the that the system fulfill real-time requirements. While some parts of the system does, not all parts do such as the USB connection between the camera and PC and some are unknown. See Table 6.1 for all components.

Even through the Linux kernel and the USB protocol can execute very similar in most executions, they are not formally real-time systems as they are not theoretically predictable. The Linux kernel can use a real-time patch but the patch is only soft real-time which means that it only optimizes to reach as many deadlines as possible, not every deadline. The same logic can be applied to the parts in Table 6.1 that are tagged as *Unknown*. The difference with these are that they are not open source and the supplier has not specified any real-time characteristics thus it is unknown if they are predictable. The OpenDLV camera interface is a real-time software if the camera firmware and operating system are real-time systems as the OpenDLV interface waits for a new image every time. If the time it takes to receive an image

System part	Real-time (RT) characteristics
Linux kernel	Soft RT
Camera firmware	Unknown
USB connection	Not RT [38]
OpenDLV camera interface	dependable RT
OpenDLV perception (Darknet)	Unknown
Intel camera driver	Unknown
Nvidia driver	Unknown

Table 6.1: Real-time characteristics of the system parts

can be predicted, the interface then can be considered to be a real-time software. The OpenDLV perception microservice however is not a real-time software as it uses darknet to run Yolo v3 which has unknown characteristics, thus it is assumed not to be formally real-time proven. At the same time, OpenDLV perception microservice uses a loop to report the objects found, which means the execution time depends on how many objects are found, so it is not real-time.

Depending on how the system result will be used, the real-time characteristics of the system might has stricter requirements. If the system is used as a assistance tool for a captain, soft real time can be good enough. Soft real-time system has value because the decisions of the captain are not fully reliant on the result from it. In another cases, if the system is used in an autonomous system the decisions are more reliant on the result. Late results from the system has the possibility to cause incorrect decisions of an autonomous vehicle thus the requirements of the system is hard real-time.

6.1.1 Yolo v3 real-time analysis

Yolo v3 is a residual CNN with all the standard layer types and special Yolo layers. All standard layers are fixed for an application and are mainly dependent on the input image size. The special Yolo layers are also predefined but also depends on number of classes and predefined bounding boxes used. Thus, the amount of computations in a Yolo v3 network are predictable and Yolo v3 has real-time characteristics as a network architecture. The last layer of a CNN network is a fully connected (FC) layer, the FC layer computes the result of the CNN. Normally, only the positive results are interesting and reported to the user or next module in the system, but then the system becomes unpredictable as the amount of reported data depends on the input image. To make the result report predictable it is possible to report all of the result from the last FC, both positive and negative results, or a limit of the number of reported objects can be implemented.

6.1.2 Convert the system to a real-time system

From the real-time analysis it is concluded that the system is not a real-time system but it is possible to make a real-time system with the same concept by changing or modifying some components. First, the camera need to exhibit real-time characteristics. Possibly, the camera used in the project might on some level show such characteristics, but this is not clearly specified by the supplier. However, since the communication protocol (i.e the USB link) is not real-time capable, a camera change is required in any case. The communication protocol used must be fast enough to transfer an image for every deadline and have time for the other processes. Se Table 6.2 for examples of how much data need to be transferred. When choosing camera, it is important that the exposure time can be controlled, because it is essential for the real-time characteristics. It is enough that the exposure time have a upper limit but otherwise reactive to the light conditions which should be better for performance. To achieve the suggested real-time schedule in Fig. 6.1 A two-

Image format	kB/deadline
Y8 640x480	307,2
Y8 1024x768	786,4
I420 640x480	614,4
I420 1024x768	1572,9
ARGB 640x480	1228
ARGB 1024x768	3147,7

Table 6.2: Amount of data needed to be transferred from the camera to PC each deadline.

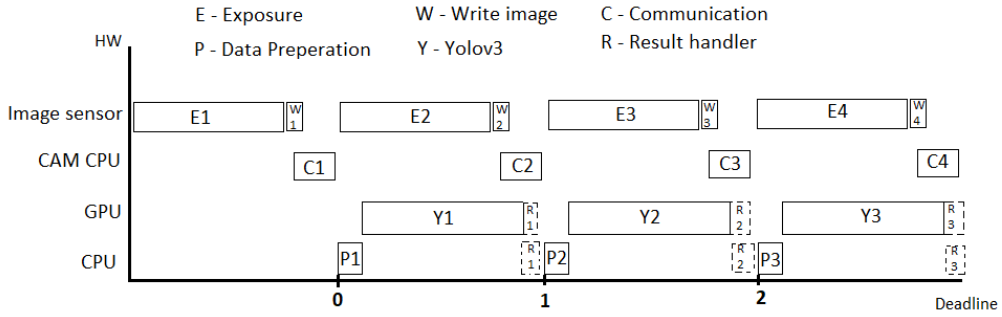


Figure 6.1: Suggested real-time schedule

way memory is needed to enable the camera CPU to start reading the data to the communication link in parallel to the image sensors writing the image.
A GPU can be used in a real-time system but they are usually not designed with real-time in mind [39] [40]. Thus, there are missing or incomplete real-time documentations. If NVIDIA GPU is used such as in this work, one need to be aware of the pitfalls identified by M. Yang et. al. [40] for real-time programming of CUDA.

Next the Linux kernel can use a real-time option but then is only soft real-time which might be good enough depending on application. If not, the Linux kernel will have to be changed to a real-time one. A real/time operating system that is suitable is Blackberry's QNX OS. Because it has the ISO 26262 ASIL D certification, the international safety standard for automotive. Lastly the OpenDLV perception will have to be changed to not call Darknet to do the detection, and the result reportage have to be changed. The report loop can be limited to a fixed amount or the method has to be changed. An option for reporting the result is to report the results for all neurons in the in the FC layers.

It is also possible to implement the algorithm on an FPGA instead of CPU and GPU.

As an FPGA design would remove obstacles as operating system and other third-party software, then only the used algorithms need to have real-time characteristics which they are already proven to have. An added benefit is that the algorithm could potentially be executed with more parallelism than in a GPU. The downside of using FPGA is higher development cost and time, it is also more difficult to make changes to the system. For these reasons, it is recommended to implement an FPGA design when the algorithm is fully developed and is not expected to have any major changes.

6.2 Potential improvements

This project used a NIR camera, an option would be to use FIR camera. The benefits of choosing a FIR camera are that there would be no need for any complimentary light source. It is unknown if the features from a FIR camera would be as useful for the Yolo algorithm. But the features should be more consistently captured compared to a NIR camera without complementary light source. The largest drawback for FIR camera is the field of view. Many FIR cameras has around 10° while NIR camera has around 90°. FIR cameras are also generally more expensive than NIR cameras. The field of view is the main reason why FIR cameras was not used in this project as it was desired that the system should be scalable to include 360° view around a boat. FIR is still an interesting possibility for improvement of the system if the cost of large field of view can be decreased.

Another improvement is to change the image format used by Yolo v3 to a grey scale format. This would decrease the amount of channels of the layers in Yolo v3 from three to one and thus decrease the amount of computations.

The final training used 500 images. It did not result in a weight file suitable for general usage. To create a better training it is recommended to gather more and more varied data than used in this work.

6.3 Future work

If this system should be used or be further developed it is recommend to include more object classes, as there are more than boats that should be avoided for a marine vehicle. Examples of such classes could be buoys, rocks, and swimmers. Another improvement can be to determine the distance to an object found.

In the systems current form it could be used as helping tool for a captain by displaying the result image. To use the system in a safe way it is recommended to implement the system in a real-time environment as discussed in section 6.1.2. When implemented as a real-time system it could also be used together with a more active system, such as path planning or steering assistance.

7
Conclusion

In this work, it was demonstrated that a NIR camera could be integrated in an embedded Linux-based computer system. Using this system, an algorithm to detect other marine vehicles, it was not limited to the 20 m requirement. The system could not be proven to fulfill real-time characteristics but it was discussed how the system needs to be modified to fulfill such requirements.

The possibility of integrating a NIR camera into a Linux system for object detection has been tested by using a camera interface for the chosen camera. The camera interface was built into OpenDLV, a layered software framework that is intended for autonomous vehicles. By combining the camera interface with the framework and connecting the microservices' data flow with OpenDLV standard message set.

The object detection algorithm used in this work was Yolo v3. The camera and Yolo v3 was then evaluated on how suitable they are for marine application with non-ideal light. The camera device is with an NIR camera and an embedded IR projector enabling it to have a better night vision image quality. Yolo v3 as a real-time object detection algorithm has the potential to be trained with low-light marine vehicle data, then later be used for detection. Even though the model still need more training to be properly evaluated, it was concluded that the system has potential for object detection in both non-ideal light and marine environment.

The system's real-time characteristics were then evaluated. The system is currently not a real-time system but it is possible to use Yolo v3 in a real-time system. However, it was found that the system need to change operating system and communication protocol to meet formal real-time requirements.

Bibliography

[1] Yilin Zhou, Mehdi Daakir, Ewelina Rupnik, and Marc Pierrot-Deseilligny. A two-step approach for the correction of rolling shutter distortion in uav photogrammetry. *ISPRS Journal of Photogrammetry and Remote Sensing*, 160:51 – 66, 2020.

[2] Creative Commons CC BY-SA 2.0. Jinx! Wicked blades.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[4] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018.

[5] Lianfa Bai, Jing Han, and Jiang Yue. *Night Vision Processing and Understanding. [electronic resource].* Springer Singapore, 2019.

[6] Hermann Kopetz. *Real-Time Systems. [electronic resource] : Design Principles for Distributed Embedded Applications.* Real-Time Systems Series. Springer US, 2011.

[7] Kourosh Kalantar-zadeh. *Sensors. [electronic resource] : An Introductory Course.* Springer US, 2013.

[8] Honda motor co. ltd.; patent issued for small boat collision avoidance apparatus (uspto 10,049,582), Aug 27 2018. Name - Honda Motor Co Ltd; Copyright - Copyright 2018, NewsRx LLC; Last updated - 2018-08-23.

[9] Sheng-Long Kao and Ki-Yin Chang. Study on fuzzy gis for navigation safety of fishing boats. *Journal of Marine Engineering & Technology*, 16(2):84–93, 2017.

[10] Burak Uzkent, Christopher Yeh, and Stefano Ermon. Efficient object detection in large images using deep reinforcement learning, 2019.

[11] Pan Wei, Lucas Cagle, Tasmia Reza, John Ball, and James Gafford. Lidar and camera detection fusion in a real time industrial multi-sensor collision avoidance system, 2018.

[12] RODRIGO F MELLO and Moacir Antonelli Ponti. *Machine Learning. [electronic resource] : A Practical Approach on the Statistical Learning Theory.* Springer International Publishing, 2018.

[13] A.L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, (1-2):206, 2000.

[14] M. Emre Celebi and Kemal Aydin. *Unsupervised Learning Algorithms.* Springer, 2016.

[15] Jack Lasky. Computer vision. *Salem Press Encyclopedia of Science*, 2019.

[16] Baohua Zhang, Doudou Jiao, Haiquan Pei, Yu Gu, and Yanxian Liu. Infrared moving object detection based on local saliency and sparse representation. *Infrared Physics Technology*, 86:187 – 193, 2017.

[17] Richard Szeliski. *Computer Vision. [electronic resource] : Algorithms and Applications.* Texts in Computer Science. Springer London, 2011.

[18] D. Sangeetha and P. Deepa. Fpga implementation of cost-effective robust canny edge detection algorithm. *Journal of Real-Time Image Processing*, 16(4):957–970, Aug 2019.

[19] Radhika Chandwadkar. Comparison of edge detection techniques. 08 2013.

[20] *Image Identification of a Moving Object Based on an Improved Canny Edge Detection Algorithm*, volume Volume 4A: Dynamics, Vibration, and Control of *ASME International Mechanical Engineering Congress and Exposition*, 11 2018. V04AT06A058.

[21] Lisang Liu, Fenqiang Liang, Jishi Zheng, Dongwei He, and Jing Huang. Ship infrared image edge detection based on an improved adaptive canny algorithm. *International Journal of Distributed Sensor Networks*, 14(3):1550147718764639, 2018.

[22] Lina Huo, Shuyuan Yang, Licheng Jiao, Shuang Wang, and Jiao Shi. Local graph regularized coding for salient object detection. *Infrared Physics Technology*, 77:124 – 131, 2016.

[23] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36(4):193–202, apr 1980.

[24] Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. Multi-column deep neural networks for image classification, 2012.

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[26] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[27] Dilip K. Prasad, Huixu Dong, Deepu Rajan, and Chai Quek. Are object detection assessment criteria ready for maritime computer vision?, 2018.

[28] K. Chrzanowski. Review of night vision technology. *Opto-Electronics Review*, 21(2):153, 2013.

[29] Elizabeth Allen and Sophie Triantaphillidou. *The Manual of Photography and Digital Imaging.* Taylor Francis Group, 2010.

[30] Massimo Bertozzi, Luca Bombini, Alberto Broggi, Paolo Grisleri, and Pier Paolo Porta. *Camera-Based Automotive Systems*, pages 319–338. Springer US, Boston, MA, 2010.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[32] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition.

[33] Fei Wang, Mengqing Jiang, Chen Qian, Shuo Yang, Cheng Li, Honggang Zhang, Xiaogang Wang, and Xiaoou Tang. Residual attention network for image classification, 2017.

[34] Alexander Wong, Mahmoud Famuori, Mohammad Javad Shafiee, Francis Li, Brendan Chwyl, and Jonathan Chung. Yolo nano: a highly compact you only look once convolutional neural network for object detection, 2019.

[35] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015.

[36] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015.

[37] D. K. Prasad, D. Rajan, L. Rachmawati, E. Rajabaly, and C. Quek. Video processing from electro-optical sensors for object detection and tracking in maritime environment: A survey, 2016.

[38] Jan Axelson. *USB Complete : The Developer's Guide*. Lakeview Research, 2015.

[39] D. vom Bruch. Real-time data processing with gpus in high energy physics. *Journal of Instrumentation*, 15(06):C06010–C06010, Jun 2020.

[40] M. Yang, N. Otterness, T. Amert, J. Bakita, J.H. Anderson, and F.D. Smith. Avoiding pitfalls when using nvidia gpus for real-time tasks in autonomous systems. In *Leibniz International Proceedings in Informatics, LIPIcs*, volume 106, University of North Carolina, 2018.