TTK4550 Engineering Cybernetics, Specialization Project

Detection and pose estimation by means of synthetic camera data

Author : Marthe Lauvsnes

Supervisor : Edmund Brekke
Co-Supervisors : Annette Stahl and Rudolf Mester

Fall 2019

NTNU
Norwegian University of
Science and Technology

Preface

This report was written as part of the course TTK4550 Engineering Cybernetics, Specialization Project. The project was completed during the fall 2019 and the intention is to do further work related to detection and pose estimation in the master thesis during the spring 2020.

I would like to thank my supervisors Edmund Brekke, Annette Stahl and Rudolf Mester for their advice and contribution to this project.

Abstract

A method for calculating the heading of a surface vehicle relative to a camera is presented, and applied to 2D images taken from a simulated 3D environment. The SLIC superpixel algorithm combined with a convolutional neural network and principal component analysis appeared to be the most robust approach, while the Harris corner detector and Hough line transform appeared less robust. The superpixel algortihm was used to oversegment the image into a desired number of segments. A YOLO classifier convolutional neural network was used to classify whether a superpixel was part of a boat, resulting in a binary image of those superpixels classified as boat parts. Principal component analysis was then used to obtain a line through the resulting binary object, on which triangulation was performed to finally obtain the desired yaw-angle.

Table of Contents

List of Tables

List of Figures

Abbreviations

ANN	=	Artificial Neural Network
CNN	=	Convolutional Neural Network
CPU	=	Central Processing Unit
CV	=	Computer Vision
GPU	=	Graphics Processing Unit
PCA	=	Principal Component Analysis
SLIC	=	Simple Linear Iterative Clustering
YOLO	=	You Only Look Once

Chapter 1

Introduction

1.1 Motivation

Making use of a 2D image from a camera sensor to predict the heading of another surface vehicle by using an artificial neural network is a novel application of computer vision (CV) pose estimation which we have not been able to find previously discussed in the literature. This approach can augment the situational awareness and be useful for vessels in general, but particularly for autonomous vehicles since they are often already equipped with cameras, and the weight and cost of other sensors can be a big consideration.

Knowing the course of other vessels in maritime transport is important for enabling safe and optimal traffic interaction. Given that another vessel is observed and is moving with a small crab angle (Fossen, 2013), the heading of that moving vessel can then be used as a good estimate for its course. Knowing the course of other vessels can then be used to aid in optimal path planning, for instance for collision avoidance or interception (i.e. replenishment at sea).

Conceptually, the heading of a target vessel can be calculated by the CV system by first visually finding the bearing (horizontal angle) of the target vessel relative to the observer vessel. Absolute bearing of the target vessel (relative to true north), which is equivalent to the heading of the target, can then be calculated if the absolute bearing of the observer vessel is also known.

Traditionally, mariners have estimated the heading of other vessels by visual observation, sometimes aided with devices such as magnifying binoculars. Marine VHF radio is another valuable tool which has enabled communication of actual or intended course between vessels. VHF was first installed on vessels starting around the 1900s, and became widespread starting in the 1920s (Korcz, 2018). Marine radars emerged on commercial vessels from around the 1950s, and today radar is perhaps the most important method for estimating heading of other vessels, often supplemented with information from an auto-

matic identification system (AIS). However, these methods are sometimes not suitable or can fail for various reasons, and adding computer vision for detecting heading can increase robustness and lead to less human error.

One recent example from March 2019 that suggests how useful estimates of heading could have been is when a Tesla crashed with a semitrailer (Lambert, 2019). The semitrailer turned left into another lane and neither the autopilot nor the driver was able to respond in time. Information about the semitrailer's heading could, if not already implemented, have been extremely valuable in this case. The reason is that the control system for the Tesla then could have detected that the vehicles were on a collision path, and therefore could have tried to avoid or minimize the consequences.

Some advantages of generating images in a simulated (synthetic) 3D environment, as done in this project using a game engine, is the ability to check the accuracy of the predictions and easier sampling of data.

1.2 Literature review

Another advantage of using synthetic data found in the literature is the ability of using the resulting training weights of the neural network on real world sampled images in later works, which would be an example of transfer learning. For an overview on transfer learning, see (Pa and Yang, 2010). Transfer learning was a central topic in the master theses of (Hølland, 2019) and (Grini, 2019).

According to the author of the You Only Look Once (YOLO) method (Redmon, 2018), the YOLO v3 has produced very competitive scores when compared to a selection of other well-known object detection methods through tests on the well known Common Objects in Context (COCO) data and label set (Lin et al., 2014). The master thesis of (Grini, 2019) found that in his project with his images of vessels, the YOLO v3 produced better results over the Single Shot MultiBox Detector (SSD).

1.3 Problem description

The goal of this specialization project is to estimate the yaw angle of a surface vehicle based on a 2D (digital camera) image. The task was simplified by only considering one specific surface vehicle, nice weather conditions and no background other than ocean and sky. Also, 2D images were obtained by using synthetic camera data from the Autoferry simulator developed by (Hem et al., 2019).

We will only consider images in a single frame; thus we will not look at target tracking from frame to frame, similar to what has been done in the theses written by (Helgesen,

2019) and (Grini, 2019). The method discussed in this thesis could be extended to continued tracking across several frames. Extended object tracking (EOT) has been discussed in the theses of (Ruud, 2018) and (Lopez, 2019), which can be used as a foundation for such future work.

YOLO v3 (Redmon and Farhadi, 2018) was chosen as a deep learning method for object detection based on the literature review.

1.4 Report outline

- In chapter 2 theoretical background is explained with theory on different feature extraction methods, camera geometry, artificial neural networks and an axis convention for surface vehicles.
- In chapter 3, the Unity game engine is presented and the simulated Autoferry environment will be used to obtain 2D images.
- In chapter 4, three different methods are discussed for estimating a line that can later be used to calculate target heading. Partial implementations are presented for two of the three methods, while a full implementation is carried out for the third method.
- In chapter 5, results of all three methods are compared. Afterwards, final results of the chosen method are discussed.
- In chapter 6 a conclusion and suggestions for further work are presented.

Chapter 2

Theory

The underlying theory for the three alternative approaches will be introduced in this section. For an overview on how these are connected, see figure 2.1. An axis convention for ships will also be introduced.

2.1 Feature Extraction

Feature extraction is a technique that can be used on images to find characteristics and attributes in the image. If these traits are used to describe the image, the dimensionality will be reduced, which can greatly speed up the processing of the image. The features are intended to be non-redundant and to provide some useful information about the image. Examples of methods for feature extraction are presented in this section. The features extracted are corners, lines and superpixels.

2.1.1 Harris Corner Detector

The Harris corner detector is used for, as the name indicates, detecting corners in an image. Corners are identified by a large change of intensity in all directions. The Harris corner detector uses this principle and it is described mathematically in the following way:

$$\Sigma_{x,y}\, w(x,y)\, [I(x+u, y+v) - I(x,y)]^2 \tag{2.1}$$

Where $w(x,y)$ is the window function at the point (x, y), $I(x,y)$ represents the intensity at the point (x, y) and u and v are displacements in the x and y direction respectively.
Equation (2.1) needs to be maximized in order to find corners. In particular we need to look at the following part of equation (2.1) as it is the term that determines the maximum:

$$[I(x+u, y+v) - I(x,y)]^2 \tag{2.2}$$

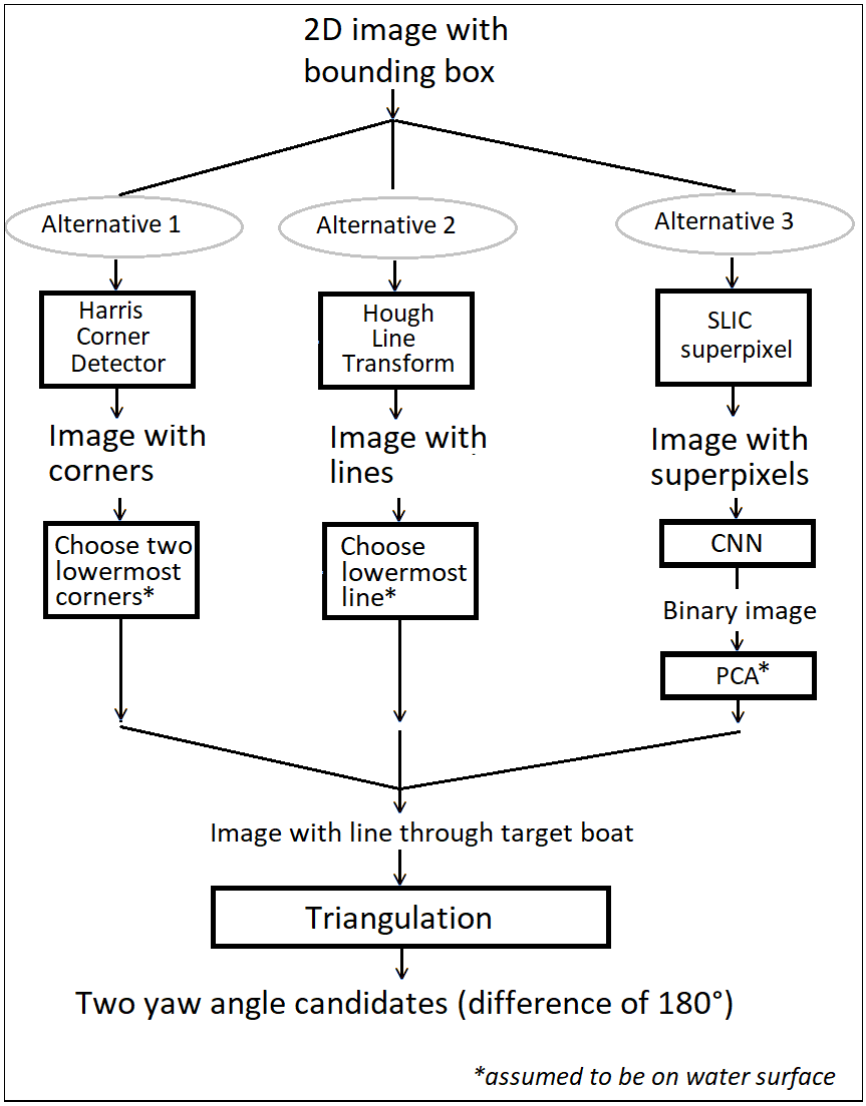


Figure 2.1: Overview of the three different approaches considered.

By performing Taylor expansion, cancelling terms and then expanding the remaining expression, the result is:

$$\begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.3}$$

where

$$M = \Sigma_{x,y} w(x,y) \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix} \tag{2.4}$$

A value R is then calculated to determine if the window contains a corner.

$$R = det(M) - k(trace(M))^2 \tag{2.5}$$

where k is an empirical constant in the interval [0.04, 0.06] (Szeliski, 2005). The value R is large for a corner, a threshold can therefore be set to decide how easily points will be detected as corners .

2.1.2 Hough Line Transform

The Hough line transform can be used to detect lines in an image. This section briefly describes how this method works.

The Hough line transform uses polar coordinates (r, θ) to represent a line. For a given point (x, y) in the image, the result will be a sinusoidal curve given by the formula $r = x\cos\theta + y\sin\theta$. The greater amount of curves that intersect in a specific point (r, θ), the more likely it is that (r, θ) represents a line in the image. In figure 2.2 five sinusoidal curves intersects in the point when $r = -0.027$ and $\theta = 2.36$, these values for r and θ represents a line in the xy-plane. Two versions are the standard Hough transform and the probabilistic Hough line transform (OpenCV, n.d.b). The standard Hough transform uses all edge points in the image when drawing sinusoidal curves, while the probabilistic Hough line transform only uses a subset of randomly selected edge points. A parameter for this algorithm is the minimum number of points required to form a line. Another parameter is the maximum gap between two points on the same line. These parameters are thresholds and determine how easily lines are detected.

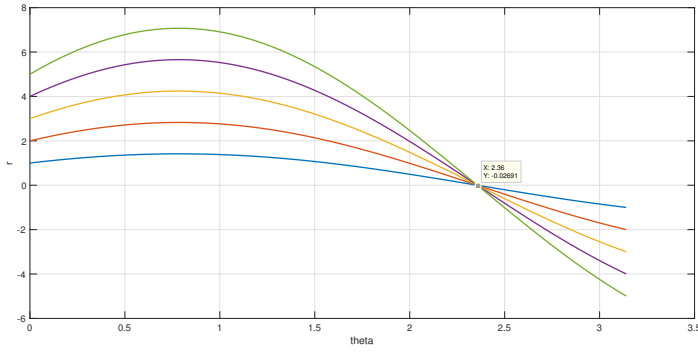


Figure 2.2: Mathematical principle behind the Hough line transform.The intersection point in the $r\theta$-plane represents a line in the xy-plane.

2.1.3 SLIC Algorithm for Superpixel Generation

Superpixels are pixels grouped together based on having similar properties. Simple linear iterative clustering (SLIC) is an algorithm that groups pixels based on pixel color and pixel closeness. The clustering is done in the 5D space $[L, a, b, x, y]$. The first three elements represent color, L ranges from dark to light, a from green to red and b from blue to yellow. The last two elements are the pixel coordinates x and y. Equations for the Euclidean norm for the Lab color space d_{lab} and the Euclidean norm for the x, y pixel coordinates d_{xy} are listed below. Also, an equation for the distance measurement used in the SLIC 5D space D_s is listed, where d_{xy} is normalized to account for different image sizes.

$$d_{lab} = \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \tag{2.6}$$

$$d_{xy} = \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \tag{2.7}$$

$$D_s = d_{lab} + \frac{m}{S} d_{xy} \tag{2.8}$$

where m is the compactness, higher m will weigh pixel closeness more compared to pixel color and therefore result in more compact clusters. The value for m is in the interval [1,20]. $S = \sqrt{N/K}$ where N is the number pixels in the image and K is the number of superpixels (Achanta et al., 2010).

2.1.4 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical method that can be used to find the direction with the highest variation in the dataset. The procedure computes the eigenvectors, which are the principal components, and eigenvalues which determines the size of the principal components. This section will show how to calculate the eigenvalues and eigenvectors in the 2D case. First, the mean in both dimension x and y are calculated:

$$\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} = \frac{1}{n} \begin{bmatrix} \Sigma_{i=1}^{n} x_i \\ \Sigma_{i=1}^{n} y_i \end{bmatrix} \tag{2.9}$$

Next the covariance matrix is calculated:

$$\mathbf{C} = \frac{1}{n-1} (\begin{bmatrix} \mathbf{x} & \mathbf{y} \end{bmatrix} - \mathbf{h} \begin{bmatrix} \mu_x & \mu_y \end{bmatrix})^* \cdot (\begin{bmatrix} \mathbf{x} & \mathbf{y} \end{bmatrix} - \mathbf{h} \begin{bmatrix} \mu_x & \mu_y \end{bmatrix}) \tag{2.10}$$

where * is the conjugate transpose operator (OpenCV, n.d.c), $\mathbf{h}$ is a vector of ones of size $n \times 1$, $\mathbf{x}$ and $\mathbf{y}$ are the dataset and are also both of size $n \times 1$.

Then the eigenvectors will be the columns of V and eigenvalues will be on the diagonal to the matrix D in the following expression:

$$V^{-1} C V = D \tag{2.11}$$

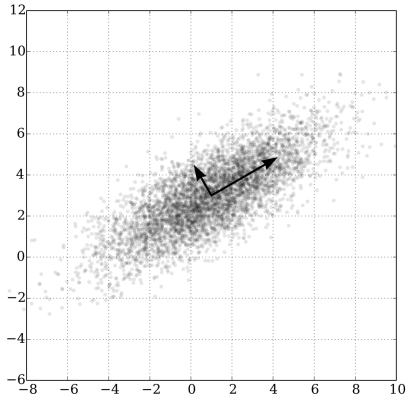


Figure 2.3: Principal Component Analysis example (Nicoguaro, 2016).

2.2 Reconstruction of Camera Coordinates from Image Coordinates

To reconstruct camera coordinates $\mathbf{x^c} = \begin{bmatrix} x^c \\ y^c \\ z^c \end{bmatrix}$ from image coordinates $\mathbf{u} = \begin{bmatrix} u \\ v \end{bmatrix}$ the following formula can be used (Haavardsholm, 2019):

$$\mathbf{x^c} = \pi_p^{-1}(\mathbf{u}, z; K) = z \begin{bmatrix} \frac{u - c_u}{f_u} \\ \frac{v - c_v}{f_v} \\ 1 \end{bmatrix} \tag{2.12}$$

where π_p^{-1} is the backprojection function, z is the depth and K, shown in equation (2.13), is the intrinsic camera matrix.

$$K = \begin{bmatrix} f_u & s_\theta & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13}$$

c_u and c_v represents the principal point in pixel coordinates. f_u and f_v can be computed by $f_u = f \cdot s_u$ and $f_v = f \cdot s_v$. f is the focal length (metric unit), s_u and s_v are the horizontal and vertical scaling factors that describes pixel density in pixels per metric unit. s_θ is the skew factor and assumed to be zero (Haavardsholm, 2019).

In figure 2.4 a camera coordinate system is shown with origin in $\mathcal{F}_c$, x-axis pointing right, y-axis pointing down and z-axis pointing forward. This convention is the one used in

this report. Also in this figure a pair of corresponding image coordinates and camera coordinates can be seen, the red line illustrates this projection.

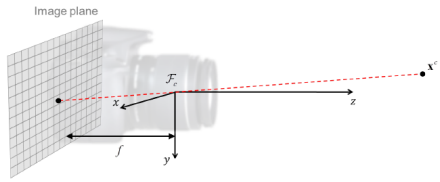


Figure 2.4: Camera coordinates (Haavardsholm, 2019).

2.3 Artificial Neural Networks (ANN)

Artificial Neural Networks is a type of machine learning that resembles the learning process in the human brain. Neurons are structured in layers and connected to form a network. The connections between two neurons are called weights. Inside each neuron, to calculate its output, an activation function is applied to the sum of inputs multiplied by weights. The activation function used is typically the sigmoid function. A simple neural network with one hidden layer is shown in figure 2.5.

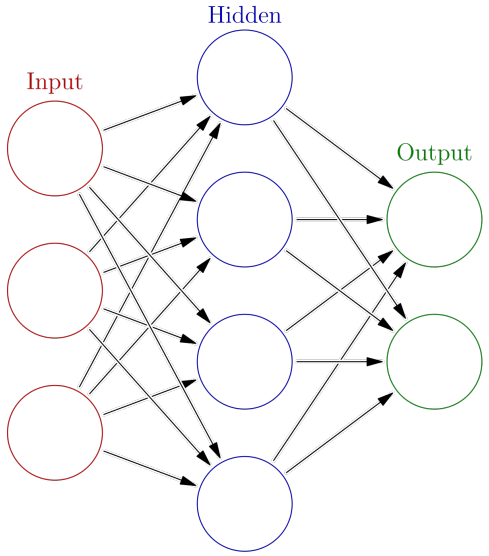


Figure 2.5: A simple Neural Network (Glosser.ca, 2013)

To train a neural network, a dataset and corresponding labels are needed. An example would be to train a neural network to be able to classify whether there is a dog or a cat

in an image. For this task dog and cat images would be needed as well as labels for each image containing information about whether there is a dog or a cat in the image. The information in the labels are what you desire the network to be able to output by itself after training is completed. So in this example the network is desired to output whether there is a cat or a dog in an image. The network also outputs a probability for each class.

The network learns by computing the error, in other words the difference between the networks output and the label. Then updating the weights such that the error decreases. This method is called backpropagation. If the network has trained successfully, the final weights can be used to classify whether there is a cat or a dog in a new image. The example above is classification. If in addition the network is trained to tell where in the image the object is, that would be called localization if one object was present or object detection if multiple objects were present. Figure 2.6 illustrates the difference.

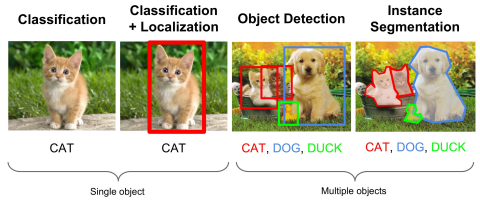


Figure 2.6: Classification vs Detection vs Segmentation (Li et al., 2016).

2.3.1 Convolutional Neural Networks (CNN)

CNNs have proven very effective in for example image classification and localization (Ujjwal, 2016). CNNs are neural networks that have alternating layers of convolutional layers and pooling layers. Convolutional layers slides a kernel over the image and computes the dot product. This operation is shown in figure 2.7. Different kernels (filters) will detect different features in an image. The values used in the filters will be adjusted during training. The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images (Ujjwal, 2016).

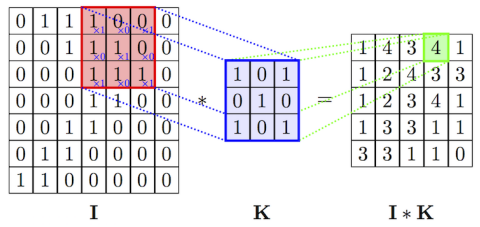


Figure 2.7: Convolution on an image I with a kernel K (Jiang, 2018).

Two alternative methods for pooling are max pooling and average pooling as shown in

figure 2.8. Max pooling extracts the maximum value whereas average pooling extracts the average value. In practice max pooling has been shown to work better (Ujjwal, 2016). The pooling layer reduces the number of parameters and computations needed in the network, and in this way overfitting is also prevented. Overfitting is when the network is performing very well on the training data, but does not generalize well.

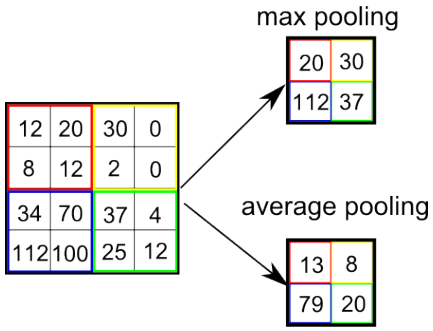


Figure 2.8: Max Pooling and Average Pooling (Anderson, 2015).

2.4 Axis Convention for Surface Vehicles

This specialization project is about pose estimation, in particular the yaw angle of a surface vehicle. The yaw angle is defined as the counter- clockwise rotation about the axis pointing upwards from the center of gravity, shown in figure 2.9.

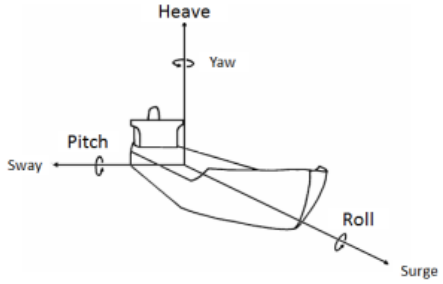


Figure 2.9: Axis convention for surface vehicles (Pocket Mariner, 2016).

Chapter 3

Unity Game Engine and Autoferry Environment

Unity is a game engine which can be used on several platforms. Both 2D, 3D, virtual reality and augmented reality games and simulations are possible in the Unity game engine. The Autoferry game was developed in the Autoferry village in the NTNU course Experts in teamwork (EiT). The Unity game engine was used to develop the Autoferry game. The game is situated in Trondheim harbour, and a screenshot from the game can be seen in figure 3.1.

Figure 3.1: Screenshot taken in the Autoferry game.

In this specialization project, the goal is to estimate the yaw angle between the camera and the ferry. This angle is available in Unity, and has been marked with a red circle in figure 3.2. This makes it easy to check the accuracy of the estimated yaw angles. Measurements and corresponding error calculations are presented in section 5.4. In the Unity settings, as shown in figure 3.2, it is also possible to set camera parameters like focal length and

sensor size (in X and Y direction). These were kept constant during the whole project, at 50 mm, 36 mm and 24 mm, respectively. Another advantage is how easy one can control the game environment, like the ferry, background, waves, camera pose, etc. Additionally, numerous pictures can quickly be taken in desired conditions in a simulated environment. Another advantage is that certain proofs of concept can be demonstrated before testing in the real world. One disadvantage is that, even though the game strives to be realistic, it can be difficult to make synthetic camera data realistic enough to an extent such that it can be used as a replacement for data sampled in the real world. However, synthetic data can be used as a supplement to for example existing real data sampled in the Trondheim harbour by using transfer learning.

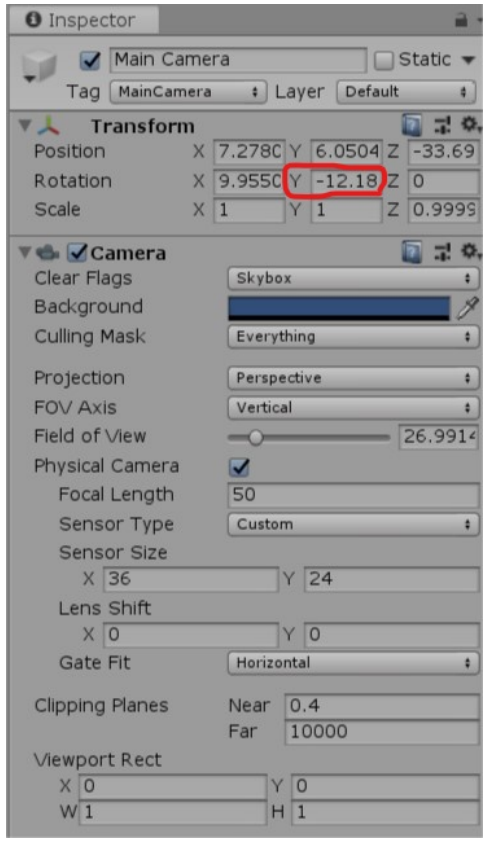


Figure 3.2: Camera settings in Unity/Autoferry.

The axis convention for Unity 3D is shown in figure 3.3. As shown in the figure, the Y axis is in the upwards direction, so the angle of interest, that is the yaw angle, will be a rotation around the Y axis, as highlighted in figure 3.2.

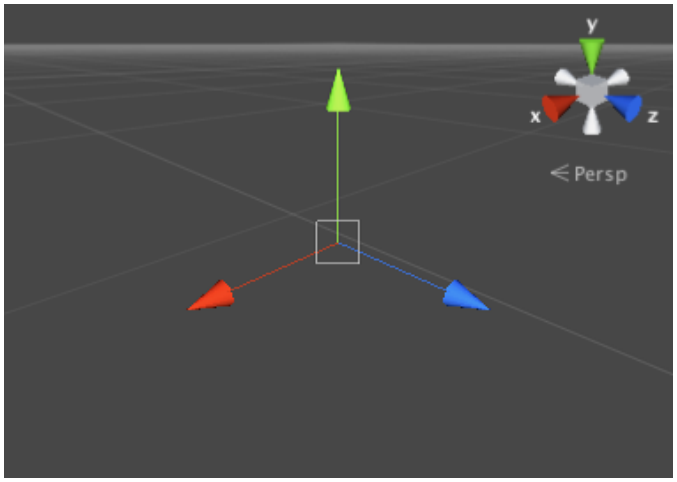


Figure 3.3: Axis convention in Unity (Unity, 2017).

Chapter 4

Implementation

To reach the goal of this project, of estimating the yaw angle from an image, three different approaches were considered. Harris corner detector and Hough line transform were two methods partially implemented in this project. The third method was superpixels combined with a CNN and PCA, and this approach was fully implemented. The theory behind these methods was described in sections 2.1.1, 2.1.2, 2.1.3 and 2.1.4. The implementation of these three methods is presented in sections 4.1, 4.2 and 4.3. Independent of which approach used among these three, the result would be a line assumed to be on the water surface. The triangulation method presented in section 4.4 can then be applied to the result from either of these three mentioned methods to give an estimate of the yaw angle of the surface vehicle.

4.1 Harris Corner Detector

The Harris corner detector was implemented by using the Open Source Computer Vision (OpenCV) library in C++ by following the tutorial in (OpenCV, n.d.a). In this program one could easily adjust the threshold while the program was running and see the resulting corners detected at that threshold. A bounding box around the object of interest is useful to minimize non-relevant corners, i.e. from waves, etc. One approach can be to use a neural network trained on detecting surface vehicles to construct such a bounding box on each image before using the Harris corner detector. However, bounding boxes were made manually in this project in order to save development time. The two lowermost corners could then be chosen and assumed to be on a line at the water surface. The triangulation method in section 4.4 can then be applied.

4.2 Hough Line Transform

The Hough line transform was also implemented by using the Open Source Computer Vision (OpenCV) library in C++. The tutorial found in (OpenCV, n.d.b) was used to

perform the Hough line transform. The process is similar to the one for Harris corner detector; a neural network could have been used for object detection, but bounding boxes were here cut manually instead. By applying the Hough line transform, the lowermost line can be found. This line can then be used to calculate the yaw angle, as described with the triangulation method in section 4.4. Both the standard Hough line transform and the probabilistic Hough line transform was implemented. No threshold was set for the standard version. For the probabilistic version 50 points was the minimum amount required to form a line and 10 was the maximum gap possible between two points on the same line.

4.3 Superpixels and PCA

4.3.1 Superpixels

The tutorial found in (Rosebrock, 2014) was used for superpixel segmentation and accessing individual superpixels written in Python. The output of running the tutorial would be a pop-up window displaying an image the same size as the original where everything would be black except the superpixel, like the upper left image in figure 4.1. There would be such a pop-up image for every individual superpixel. This code was modified to find the bounding box of the superpixel and crop the image to only contain this bounding box, four examples are shown in figure 4.2. The code was also modified to save the cropped image of each individual superpixel instead of displaying it. While generating this dataset of superpixels, the compactness was mostly set to the default $m = 10$ and the number of superpixels was mostly set to $K = 300$. Some images were however segmented using different parameter values. For example, smaller images were segmented with a lower K value to keep the superpixels more similar in size.

Only images from the Autoferry game were used when creating the superpixel-dataset. The images only contained the ferry in the foreground, as well as ocean and sky in the background. No images containing buildings, land or other objects were used in the dataset. Furthermore, the pictures were taken under good weather conditions and in calm ocean in the simulated environment.

4.3.2 Labelling of Superpixels

To label the dataset of superpixel-images now created, a simple labelling tool was made as shown in figure 4.1. The labelling tool would display images of the superpixel on black background as well as the original image with all superpixel-segments. The labelling tool would then prompt the user to input ”0” if the superpixel was part of the background, ”1” if it was a part of the surface vehicle and ”2” to skip labelling. This process would continue through all superpixels in one image. To speed up the process one could input an image of only surface vehicle superpixels or only background superpixels then labelling would be done automatically. The result would be a new text file with the same name as the corresponding superpixel-image file containing ”0” if the picture is part of the background

or ”1” if the picture is part of the surface vehicle.

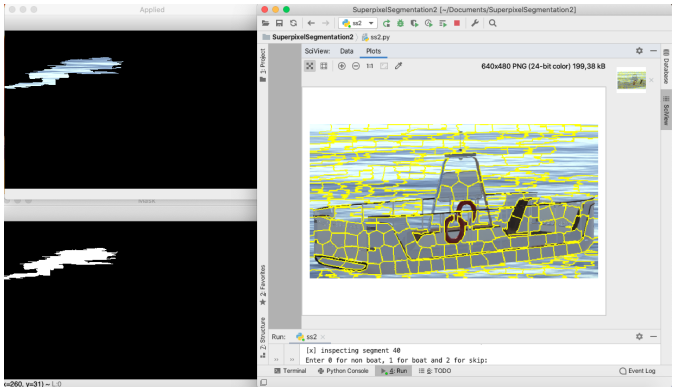


Figure 4.1: Simple labelling tool for superpixels.

To fix labels for the YOLO detector, a program was made for appending ”0.5 0.5 1.0 1.0” to all the label files in order to convert the labels to the YOLO detector format $< class >< x - center >< y - center >< width >< height >$. The four appended values tells YOLO that the object is the whole image, because all four values are floating point values between 0 and 1. A YOLO classifier was also trained, then the image filenames needed to be changed to contain the labels. The image filenames were then on the format $< uniquenumber >\_< classname >$ for example $39\_boat.png$.

As indicated earlier, two classes were used, one for surface vehicles and one for background. Background was defined to be every pixel that is not part of the surface vehicle. Approximately 3300 superpixel images were sampled and labelled of the surface vehicle class, and approximately 5100 superpixel images of the background class. The convention used when labelling was that superpixels containing both two classes were skipped and that one should label a superpixel as the foremost object. Figure 4.3 is divided into superpixels and the numbers 1 to 4 illustrates some edge cases. The superpixel marked in the example as number 1 (the rightmost box) would be skipped during labelling as it contains both the ferry and background. The superpixel marked as number 2 (the box in low left) would be labelled background since the ocean is the foremost object. An example of the opposite case can be seen with the superpixel marked as number 3 (top left) where the boat is the foremost object, and the superpixel would be labelled boat. The superpixel marked as number 4 (middle) is an example of where it is possible to see ocean through openings in the boat, and this superpixel would be labelled as background.

4.3.3 Training of CNN for Segmentation

Two CNNs were trained, one for object detection and one for classification, the difference can be seen in figure 2.6. The YOLO detector was used for object detection, whereas the YOLO classifier was used for classification. Implementation details for these two CNNs

Figure 4.2: Examples of superpixels used to train the CNN.

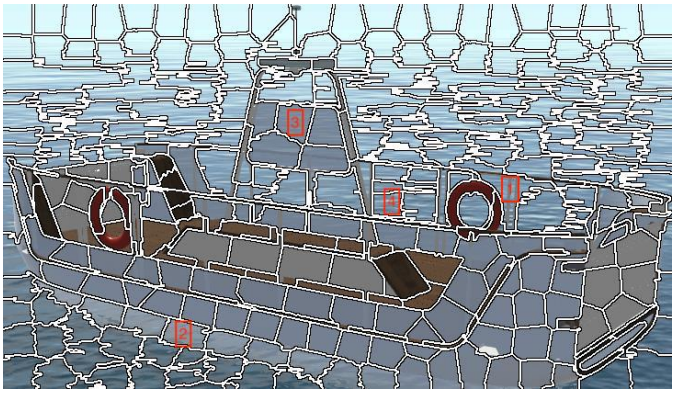


Figure 4.3: Labelling convention, here illustrated with four numbered boxes.

are presented in this subsection.

Preparation for training

The Darknet fork found in (AlexeyAB, 2019) was used as the neural network framework, and YOLO version 3 was used for classification of whether a superpixel was a surface vehicle or not. Information about the computer and software used are specified in tables 4.1 and 4.2. Specifications used were based on the requirements also found in (AlexeyAB, 2019). Darknet was compiled with GPU and cuDNN for accelerating the training of the neural network.

Custom objects were used in this project and therefore some changes in configuration files were needed. For the YOLO detector, a detailed explanation can be found under ”How to train (to detect your custom objects)” in (AlexeyAB, 2019). The only difference was that subdivision size was set to 64 instead of 8, due to an out of memory error. For the YOLO classifier a guide can be found in (Redmon, 2013–2016a), but in the last convolutional layer only two filters should be used since we only have two classes. Additional changes were also made to use the dataset with superpixels instead of the example dataset.

CPU	Intel Core i7-6700 3.4GHz
GPU	GeForce GTX 1080 Ti
Operating System	Linux Ubuntu 16.04

Table 4.1: Hardware specifications.

CMake	3.15.5
CUDA	10.0
OpenCV	4.1.2
cuDNN for CUDA 10.0	7.6.3
GCC	5.4.0

Table 4.2: Software specifications.

Furthermore the dataset of superpixel-images were split randomly into two sets, one containing approximately 80 % of the pictures and another the remaining 20 %. A program was made to generate one text file with image paths to each of the training images, as well as another text file with image paths to each of the validation images. The set containing 80 % of the images was used for training the neural network, whereas the set containing 20 % of the images was used for validating the results by testing the performance of weights on unseen images. The network was trained using pretrained weights downloaded from (Redmon, 2013–2016b).

Training

The training of the neural network was completed on a Linux computer with specifications as in table 4.1 and table 4.2. A computer with GPU was necessary for the training of the YOLO detector to complete within reasonable time. First training was tried on a Windows computer with only CPU, estimated time for training was then 30 days, whereas it completed in approximately 5 hours with GPU. The YOLO detector was trained on different dataset sizes, the largest at 8400 images. The Yolo classifier was trained on a dataset of 3200 images.

After Training

For the YOLO detector the weigths after every 1000th iteration were compared and the one with best results on the validation sets were used further. For the YOLO classifier the weigths after 2000, 3000, 4000 and 5000 had very similar results on the validation tests, therefore the weights after 5000 iterations were used.

4.3.4 Principal Component Analysis

Before performing PCA, the CNN was used together with the chosen weights to detect which superpixel-images were part of the surface vehicle and which were not. This re-

sulted in a binary image were the neural network's predictions of the boat was white and the background was black. This binary image could then undergo PCA, the program was written in C++ based on the OpenCV tutorial found in (OpenCV, n.d.c).

4.4 Yaw Angle Calculation

Each of the three methods presented in this report will, if successful, result in a line indicating the yaw angle of the vehicle. The calculation presented in this chapter is based on assuming that this line is on the water surface.

If two points (u_1, v_1) and (u_2, v_2) in the image are on the line indicating the yaw angle of the vehicle, it is assumed that the corresponding camera coordinates (x_1, y_1, z_1) and (x_2, y_2, z_2) will be on the water surface. The y-axis in the camera coordinate system is pointing downwards as seen in figure 2.4. The y camera coordinates y_1 and y_2 can then be approximated to be equal to the height of the camera above water.

$$y_i = h \qquad i \in 1, 2 \tag{4.1}$$

where h is the height of the camera above water. Inserting the second component in the equation for camera coordinates, equation 2.12, into equation 4.1 results in :

$$z_i \frac{v_i - c_v}{f_v} = h \qquad i \in 1, 2 \tag{4.2}$$

$$z_i = h \frac{f_v}{v_i - c_v} \qquad i \in 1, 2 \tag{4.3}$$

Then z_1 and z_2 can be found from equation 4.3 since the camera height, the focal length, the pixel coordinate and the principal point is known. The next step will then be to find x_1 and x_2, these coordinates can be found directly from the first component in equation 2.12, repeated here in the following equation:

$$x_i = z_i \frac{u_i - c_u}{f_u} \qquad i \in 1, 2 \tag{4.4}$$

Finally the yaw angle ψ can be estimated by the following expressions:

$$\psi_1 = \arctan \frac{z_2 - z_1}{x_2 - x_1} \tag{4.5}$$

$$\psi_2 = \psi_1 + 180° \tag{4.6}$$

This method gives two alternatives for the yaw angle. The difference between the angles will be 180°. Further analyses would be needed to narrow it down to one solution. Note that tilting the camera is not accounted for in this calculation, so keeping the camera horizontal is required. The camera is allowed to be rotated around the vertical axis as the

calculation provides the relative angle between the ferry and the camera. Rotation matrices can be added to the calculation to find the world coordinates, as well as allow all camera rotations. Note also that images cannot be cropped as the calculation depends on image size and image coordinates.

Chapter 5

Results and Discussion

5.1 Comparison of Harris Corner Detector, Hough Line Transform and Superpixels

To compare the three approaches presented in this report, all three methods are tested on the same image. The results of the comparisons can be seen in figure 5.1, figure 5.2 and figure 5.3.

For the Harris corner approach in figure 5.1, the original image is on the left, the middle image has the threshold for corner detection set to 121 out of 255 while the right image has lowered the threshold to 95, and therefore far more corners are detected. The image with the highest threshold would perform better for our purpose when drawing a line between the two lowermost corners requiring some minimum distance between these two corners. The right image would however get a large error by using this approach. The right image was produced by lowering the threshold such that an erroneous corner would appear as one of the two lowermost points. Even though the corner detection works quite nicely in these conditions, it is suspected to be a more difficult problem in more realistic conditions. Different light conditions or objects in the water could make an erroneous corner, like the lowest one on the right image, appear at higher threshold. The fact that one erroneous corner could make the solution very wrong makes the Harris corner approach less robust for our purpose.

Resulting images from the Hough line tranform approach is shown in figure 5.2. On the left, the same original image as used for the Harris corner detector can be seen. In the middle, it can be seen that the standard Hough line transform detects a lot of lines in the water, and the only possible candidate would be the line on the boat railing. However it can cause some error since we have assumed this line to be on the water surface. The probabilistic Hough line transform, shown in the right image, has more lines on the boat. It can be seen that also for the probabilistic Hough line transform there is potential for

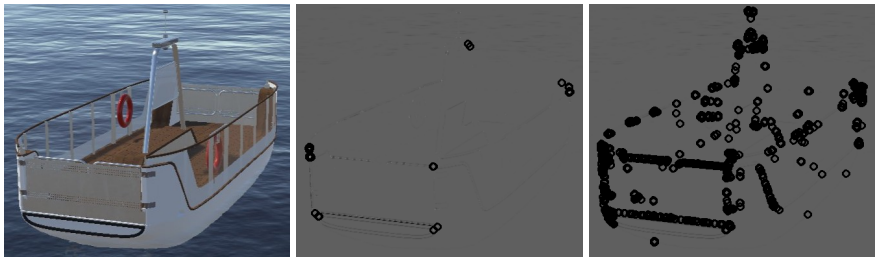


Figure 5.1: Original image, Harris corner detector with thresh 121/255, Harris corner detector with thresh 95/255.

error. The lowermost line is based on reflections in the water and will by eye inspection clearly provide a wrong angle. The two lines above on the boat would have provided better results. However, like for the Harris Corner we can see that the Hough line approach is sensitive to one wrong line, and the Hough line transform is therefore not as robust as we would like it to be either.

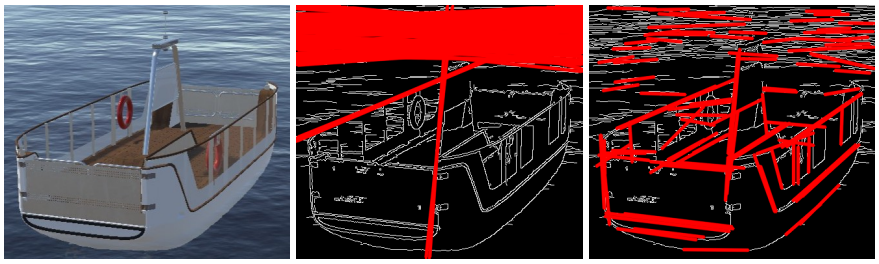


Figure 5.2: Original image, standard Hough line transform with no threshold, probabilistic Hough line transform.

The third approach is using a CNN trained on superpixel-images to perform segmentation, then applying PCA to the resulting binary image. Results for this approach, using the same original image as the Harris corner detector and Hough line transform, can be seen in figure 5.3. The PCA gives only one line to consider. The line could be reasonable, but this can be difficult to conclude just by eye inspection. Calculations and further conclusions about accuracy of this method can be found in section 5.4.

It was quite early in the process concluded that Harris Corner Detector and Hough line transform would be less robust compared to the Superpixel and PCA approach in realistic conditions. This was due to how the Harris corner detector and the Hough line transform would be affected in different light conditions, reflections, weather conditions and if other objects hid part of the object of interest. The superpixel and PCA approach was believed to be robust against a few erroneous superpixel classifications. The superpixel and PCA method was therefore focused on and accuracy calculations were performed for this approach as can be seen in section 5.4.

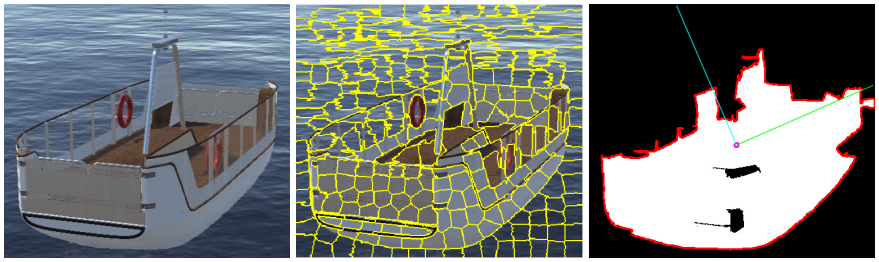


Figure 5.3: Original image, Superpixels, PCA on the segmented image.

5.2 Yolo Detector vs Yolo Classifier

At first, a lot of effort was made to get good segmentation results by using the YOLO detector. The YOLO detector was chosen due to more available information online and a belief that it might be performing just as well with classification, but providing additional data. The information from the detector about where in the image each class was located was believed to be useful if there was a superpixel with both classes. Then this information could be used for further processing of this superpixel. To make the YOLO detector, a large network with 75 convolutional layers was used, and weights obtained after different iterations were tested. Because non-satisfactory results were obtained, the dataset was increased two times and the neural network retrained, to check if this would provide better results. One of the better results of the YOLO detector is shown in figure 5.4, as seen a lot of wrong classifications are still present.

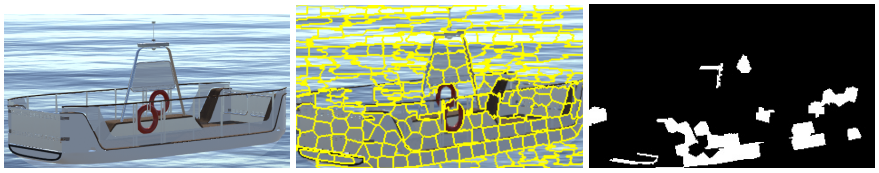


Figure 5.4: Segmentation result by using the YOLO detector.

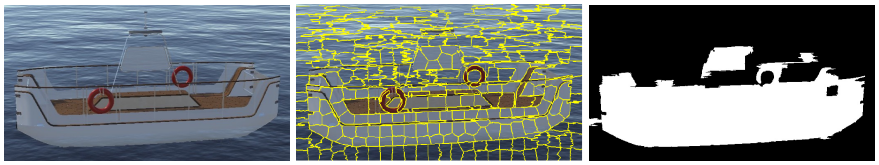


Figure 5.5: Segmentation result by using the YOLO classifier.

Results after segmenting the ferry with a YOLO classifier CNN can be seen in figure 5.5. Segmentation results by using the YOLO classifier can also be seen in the third image in each sub-figure in figures 5.6 and 5.7, where the original image is in the first image in each sub-figure. Results from the YOLO classifier can also be seen in 5.11 and 5.12.

As seen, the YOLO classifier performs significantly better than the YOLO detector, and only classifies a few superpixels wrong. The YOLO classifier worked with less testing of different weights, smaller dataset and also a smaller network. The YOLO classifier only had 4 convolutional layers and 2 max pooling layers.

It is interesting how much better the small classifier network performed compared to the much larger detector network, as a deeper network needs to know the problem better in order to give good results. However as the need in this project only was classification, it can in hindsight seem like the classifier should have been an obvious choice.

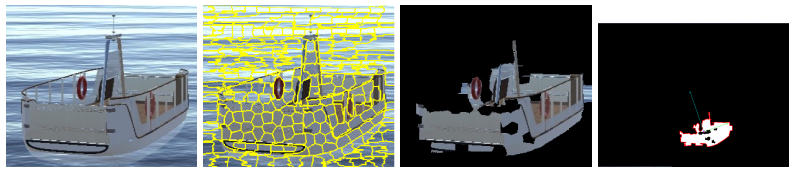
5.3 Superpixel, CNN segmentation and PCA results

In this section results for the SLIC superpixel algorithm, segmentation by using the YOLO classifier CNN and PCA will be presented. Image examples of the ferry with 0° to 160° between boat and camera are presented, at 20° intervals. Due to symmetry of the ferry, results for angles between 180° to 360° will not be presented. When the front of the boat and camera points in the same direction like in 5.6a the angle is 0°. A positive clockwise rotation is used when defining an angle for the ferry images, see also corresponding caption to the sub-figures for the true angle.

The SLIC superpixel algorithm performs well in most parts of the image and the superpixels mostly contain either only ferry-pixels or only background-pixels. The difficult parts are often the openings in the ferry and the reflection of the ferry in the ocean. Often the railing on the ferry and the navigation light mast are inside a superpixel with background. The same is the case for reflections of the ferry and the ferry itself. These pixels were skipped during labelling so the network has not been learned what to do in these situations. A solution could be to divide a superpixel into several superpixels for the superpixels where the network outputs lower probability for the classification. The YOLO classifier CNN results are presented as the third image in each sub-figure, as mentioned earlier the majority of superpixel-classifications are correct.

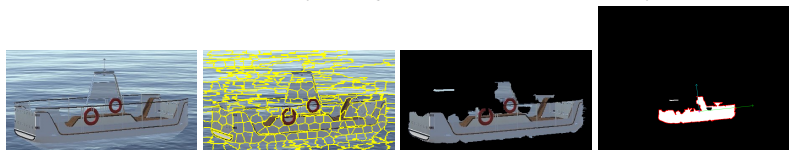
In the fourth image in each sub-figure the binary version of the third image has undergone PCA. Without further inspection we can see that the largest principal component from the PCA in 5.6b will clearly deviate from the true value of 20°. This is believed to be due to the square appearance when the ferry is at this angle, and in addition there are some erroneous classifications in the back of the boat. A longer boat would possibly have given better result at this angle. Another alternative could have been to change the camera pose, the camera could be tilted down and/or moved to a higher position. The principal components for 20° to 160° appear to be better, however further calculations are required, these can be found in section 5.4

(a) Boat with 0° yaw angle between camera and ferry.



(b) Boat with 20° yaw angle between camera and ferry.

(c) Boat with 40° yaw angle between camera and ferry.



(d) Boat with 60° yaw angle between camera and ferry.

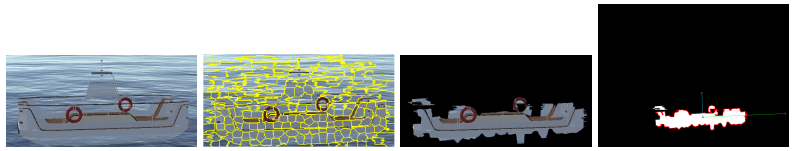
Figure 5.6: Comparison of bounding box of ferry, superpixels, segmented ferry and PCA for different angles.

5.4 Accuracy for Estimation of Yaw Angle

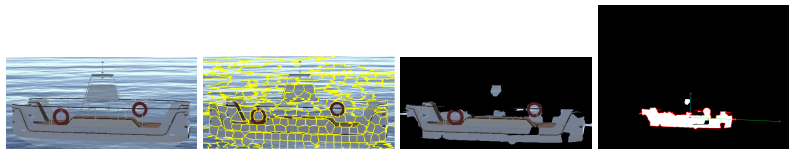
In this section accuracy is presented and discussed. The method consists of several steps and each step contributes with some error. The steps in this method are, as mentioned before, segmenting an image in superpixels, classifying which are part of the boat by using a YOLO classifier, letting the image undergo PCA and using the longest resulting principal component to calculate the yaw angle.

5.4.1 Accuracy Calculation

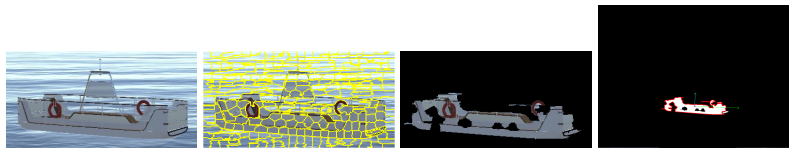
The calculations based on each sub-figure in 5.6 and 5.7 are presented in table 5.1. As seen from the table, errors are in the range 1.2° to 158.1°. The average error is 25°, the median error is 7.3° and the standard deviation is 50.3°. More estimations are however needed to make the statics more reliable, as these calculations are only based on nine samples.



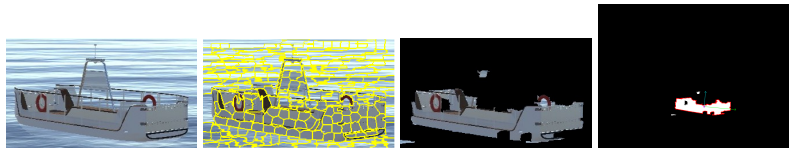
(a) Boat with 80° yaw angle between camera and ferry.



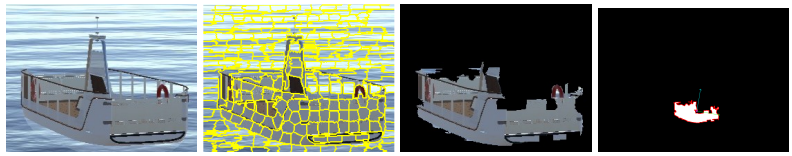
(b) Boat with 100° yaw angle between camera and ferry.



(c) Boat with 120° yaw angle between camera and ferry.



(d) Boat with 140° yaw angle between camera and ferry.



(e) Boat with 160° yaw angle between camera and ferry.

Figure 5.7: Comparison of bounding box of ferry, superpixels, segmented ferry and PCA for different yaw angles.

The two values that stand out in a negative way are the ones where the true angle are 20° and 160°, particularly 160°. The explanation for these poor results can be seen by considering the principle components. This can be seen in the right image in figure 5.6b and figure 5.7e, but also in figure 5.8 where the images are enlarged to easier see the details. When calculating the yaw angle the largest principal component (longest line) is used. As we can see the green lines are much better candidates, however since the blue lines are longer, those are used and we get large errors. Using the green line for calculation of the yaw angle gave a result of 23.8° when the true angle was 20°, in other words only an error of 3.8°. For the sample when the true angle was 160° using the smaller principal component (green line) gave an estimated angle of 146.6 ° and therefore an error of 13.4 °.

True angle	Estimated angle	Error
0°	1.2°	1.2°
20°	-0.5°	20.5°
40°	33.8°	6.2°
60°	52.7°	7.3°
80°	77.7°	2.3°
100°	111.4°	11.4°
120°	114.0°	6.0 °
140°	129.5°	10.5°
160°	1.9°	158.1°

Table 5.1: True vs Estimated Yaw Angle.

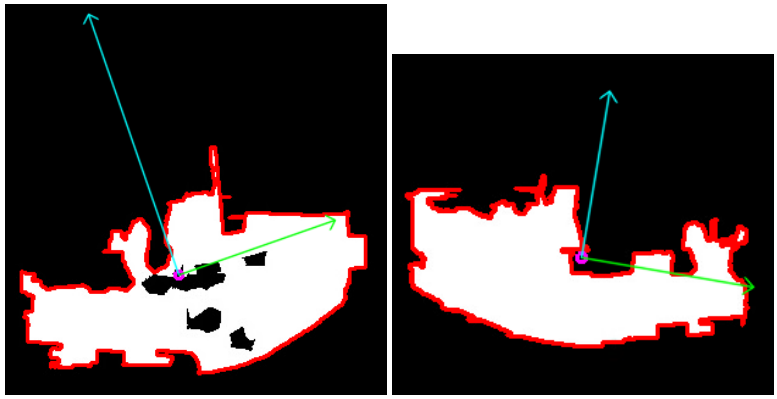


Figure 5.8: Enlarged result of PCA for 20° angle on the left image and for 160° angle on the right image.

5.4.2 Effect of PCA on Differently Segmented Images

In figure 5.9 the result of applying PCA on three differently segmented images can be seen. All images are based on the left image in figure 5.7b. Figure 5.9a was segmented by the YOLO classifier CNN, same result as shown in the third image in figure 5.7b. Figure 5.9b was segmented manually to contain all the ferry pixels. In figure 5.9c the navigation light mast was removed. This comparison illustrates how different segmentation would affect the accuracy. Figure 5.9a has a difference between true and estimated yaw angle of 11.4° as mentioned in table 5.1. The estimated yaw angle of figure 5.9b was 95.4° which resulted in an error of 4.6°. The estimated yaw angle of figure 5.9c was 95.8° which resulted in an error of 4.2°. Even though it is not desired for the CNN to classify such that the segmentation becomes like in figure 5.9b or figure 5.9c as they do contain background pix-

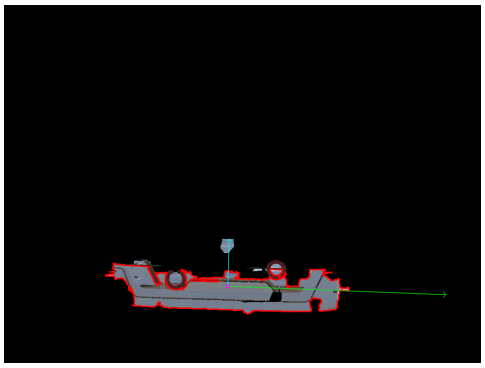
els, something more similar would be desired. However, as mentioned the openings in the ferry makes it harder for the CNN to perform the classification. Quite a few superpixels are wrongly classified at the lower left and upper right of the ferry. If those superpixels where added, the longest principal component (in this figure the green line) would be pointing a bit more upwards and moved closer to 100° instead of the current 110°. Note that the manually segmented images on the other hand here predicts lower angles than the true one.

The principal components in figure 5.9b are almost equally long, by measurement the green line is only a bit longer. This can cause large errors if the boat is believed to point in the direction of the longest principal component. This is as mentioned what happened when the true angle was 20 deg and 160 deg, this resulted in large errors as seen in table 5.1. In 5.9c the effect of removing the navigation light mast is shown to reduce one of the principal component and increases the other one. This result gives an indication that this method may perform better at oil tankers and similar and possibly worse on sail boats and similar if not countermeasures are taken. The ferry used for analysis is not that long relative to the mast, and may appear square around 0° and 180°.

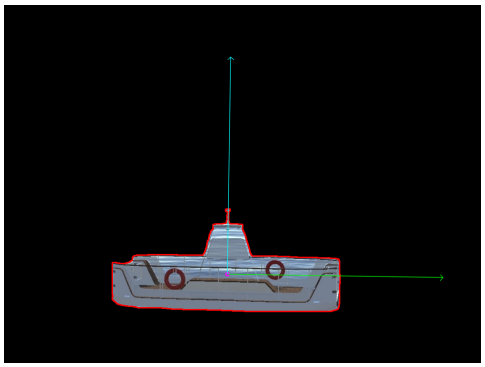
5.4.3 Effect of Tilted Camera

Training images were taken with a tilted camera, however the images in figure 5.6 and figure 5.7 were taken without tilting the camera. An example of the difference between tilted and not tilted camera can be seen in figure 5.10. The images both have 0° between the ferry and the camera, however the ferries look quite different. When the camera is tilted the ferry appears longer whereas when the camera is not tilted the ferry appears more square which could be a challenge when performing PCA in regards to which principal component becomes longest. When the camera is tilted more wooden floor is visible. It is most likely easier for the CNN to distinguish this wooden brown from the ocean compared to the task of distinguishing the grey-white outside of the boat to the ocean. Also the light conditions are different in the two images which may be due to images taken in different locations in the simulated environment or due to the camera being tilted.

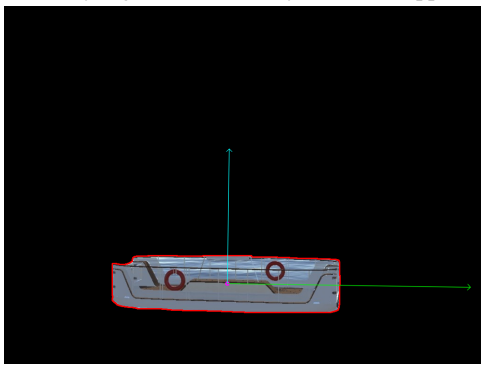
Images with tilted camera are shown in figure 5.11 and figure 5.12. The images are taken at the same angles between the ferry and the camera as for figure 5.6 and figure 5.7. By inspecting the difference, the images taken with tilted camera have better segmentation results in general. It would be interesting to figure out whether training with non-tilted camera images would have given improved results or if tilting the camera makes it easier for the CNN to perform segmentation. The difference in result may also be due to some other reason like image quality, location in environment or light conditions.



(a) Ferry segmented by YOLO classifier CNN with PCA applied



(b) Ferry segmented manually with PCA applied



(c) Ferry segmented manually with navigation light mast removed with PCA applied

Figure 5.9

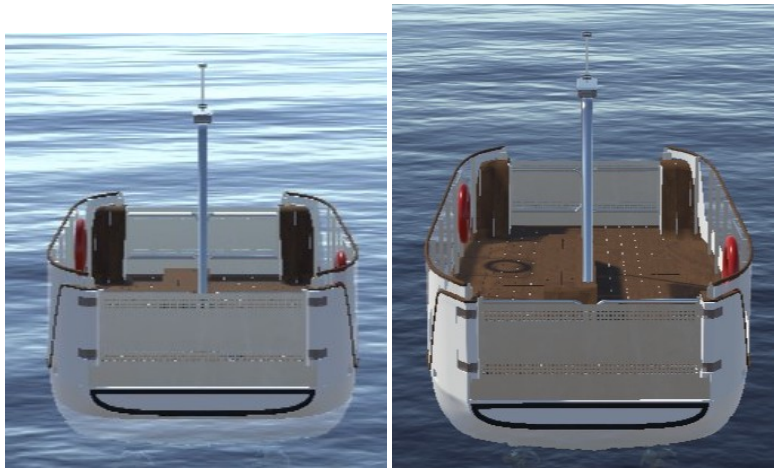
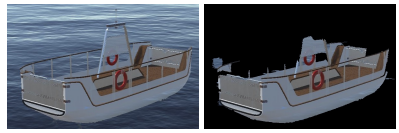


Figure 5.10: Image taken with camera kept horizontal on the left and image taken with tilted camera on the right.

(a) Boat with 0° yaw angle between camera and ferry.

(b) Boat with 20° yaw angle between camera and ferry.



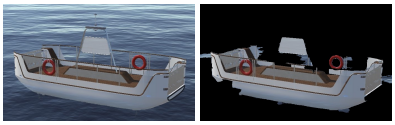
(c) Boat with 40° yaw angle between camera and ferry.

(d) Boat with 60° yaw angle between camera and ferry.

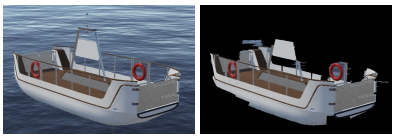
Figure 5.11: Segmentation results when images were taken with a tilted camera.

(a) Boat with 80° yaw angle between camera and ferry.

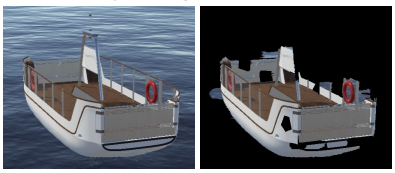
(b) Boat with 100° yaw angle between camera and ferry.



(c) Boat with 120° yaw angle between camera and ferry.



(d) Boat with 140° yaw angle between camera and ferry.



(e) Boat with 160° yaw angle between camera and ferry.

Figure 5.12: Segmentation results when images were taken with a tilted camera.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

In this project, three methods for estimating the yaw angle was compared. The method indicated to be the most robust was performing segmentation with the use of superpixels and a CNN, after which PCA and triangulation was applied. The classification process seemed to be the cause of some errors. Another cause of errors was that the PCA had some difficulties for angles around 0° and 180° presumably as the 2D projection of the ferry then appeared more square. Tilting the camera gave better segmentation results and longer boats with less mast performed better under PCA.

Even though some significant errors are present, the method is promising and exciting. There are a lot of further work that can be done to improve the result. Some suggestions for further work are presented in 6.2.

6.2 Future Work

The following tasks are suggested for future work:

The YOLO classifier CNN for superpixel-images could be explored more. A larger dataset could be used to train the CNN and more layers could have been added to form a deeper network.

The dataset could be expanded to add different types of surface vehicles and the images could be taken under different weather and light conditions. Also images with buildings and land in the background could be added.

A class could be added for buildings when labelling and training the network. It has been shown in a master thesis that training on a building class can have significant positive ef-

fects (Grini, 2019). It would be interesting to figure out how good a classifier is able to distinguish between boat and building superpixels, or whether it becomes too difficult if many different buildings and boats are used.

Real images could be added to the dataset and transfer learning could be used to train a CNN, then the results could be tested in a real environment instead of simulated.

For the cases when the neural network has low probabilities on classifications, possibly in cases were a superpixel contains several classes, further analysis could be done to distinguish the classes. This could result in better segmentation result.

The calculation of the yaw angle could be used as an input to a CNN together with the original image. This CNN could be used to decide which of the two alternative angles are correct, or even possibly correct the calculation in cases like the one we had in figure 5.7e. The results of this grey box neural network could be compared to a black box neural network.

Bibliography

Achanta, R., Shaji, A., Smith, K., Lucchi, A., Fua, P., S*ü*sstrunk, S., 2010. Slic superpixels. Tech. Rep. 149300, EPFL.

AlexeyAB, 2019. Yolo-v3 and yolo-v2 for windows and linux. `https://github.com/AlexeyAB/darknet`, accessed: 2019-11-10.

Anderson, P., 2015. `https://panderson.me/images/Caffe.pdf`, accessed: 2019-10-06.

Fossen, T. I., 2013. Handbook of Marine Craft Hydrodynamics and Motion Control. Wiley.

Glosser.ca, 2013. `https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg`, accessed: 2019-12-03.

Grini, S., 2019. Systematic training and testing of detection methods for vessels in camera images. Master's thesis, NTNU, accessed: 2019-08-28.

Haavardsholm, T., 2019. A handbook in visual slam. Lecture notes in course on Visual SLAM at NTNU.

Helgesen, O. K., 2019. Sensor fusion for autonomous ferry. Master's thesis, NTNU, accessed: 2019-08-30.

Hem, A. G., Leineb, D., Opheim, H. V., Vasstein, K., Skarshaug, T., 2019. Digital tvillingInstitution: NTNU, accessed: 2020-01-04.

Hølland, E. H., 2019. Detection of ships in infrared images. Master's thesis, NTNU, accessed: 2019-10-15.

Jiang, Z., 2018. `https://conference.ippp.dur.ac.uk/event/660/contributions/4058/attachments/3402/3717/CNN_04_04_ZihaoJiang.pdf`, accessed: 2019-11-20.

Korcz, K., 2018. Maritime radio systems for distress alerting. Journal of KONES 25 (1), 233–240.

Lambert, F., 2019. Tesla autopilot fatal crash with truck is under investigation, preliminary report released. https://electrek.co/2019/05/16/tesla-autopilot-fatal-crash-truck-investigation-preliminary-report/, accessed: 2019-11-02.

Li, F., Karpathy, A., Johnson, J., 2016. http://cs231n.stanford.edu/slides/2016/winter1516_lecture8.pdf, accessed: 2019-11-21.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollr, P., Zitnick, C. L., 2014. Microsoft coco: Common objects in context. In: European Conference on Computer Vision (ECCV). Zürich, oral.
URL /se3/wp-content/uploads/2014/09/coco_eccv.pdf,http://mscoco.org

Lopez, M. E., 2019. Extended object tracking for autonomous ferry. Master's thesis, NTNU, accessed: 2019-09-05.

Nicoguaro, 2016. https://commons.wikimedia.org/wiki/File:GaussianScatterPCA.svg, accessed: 2019-12-07.

OpenCV, n.d.a. Harris corner detector. https://docs.opencv.org/3.4/d4/d7d/tutorial_harris_detector.html, accessed: 2019-09-13.

OpenCV, n.d.b. Hough line transform. https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html, accessed: 2019-09-13.

OpenCV, n.d.c. Introduction to principal component analysis (pca). https://docs.opencv.org/3.4/d1/dee/tutorial_introduction_to_pca.html, accessed: 2019-11-03.

Pa, S. J., Yang, Q., 2010. A survey on transfer learning. IEEE Transactions on Knowledge and Data Engineering 22 (10), 1345–1359.

Pocket Mariner, 2016. Monitor vessel motion and situation in real-time with aiswatch telematics. https://www.pocketmariner.com/pm/, accessed: 2019-11-10.

Redmon, J., 2013–2016a. https://pjreddie.com/darknet/train-cifar/, accessed: 2019-12-20.

Redmon, J., 2013–2016b. https://pjreddie.com/media/files/darknet53.conv.74, accessed: 2019-10-01.

Redmon, J., 2018. https://pjreddie.com/darknet/yolo/, accessed: 2019-11-16.

Redmon, J., Farhadi, A., 2018. Yolov3: An incremental improvement. arXiv.

Rosebrock, A., 2014. Hough line transform. https://www.pyimagesearch.com/2014/12/29/accessing-individual-superpixel-segmentations-python/, accessed: 2019-10-09.

Ruud, K. A., 2018. Lidar extended object tracking of a maritime vessel using an ellipsoidal contour model. Master's thesis, NTNU, accessed: 2019-08-22.

Szeliski, R., 2005. Notes on the harris detector. https://courses.cs.washington.edu/courses/cse576/06sp/notes/HarrisDetector.pdf, accessed: 2019-09-03.

Ujjwal, U., 2016. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/, accessed: 2019-10-30.

Unity, 2017. https://docs.unity3d.com/560/Documentation/Manual/Transforms.html, accessed: 2019-10-02.