NTNU

Norwegian University of
Science and Technology

Computer Vision Based Obstacle Avoidance for a Remotely Operated Vehicle

Lars Sletbakk Brusletto

Master of Science in Engineering and ICT
Submission date: July 2016
Supervisor: Martin Ludvigsen, IMT
Co-supervisor: Stein M. Nordnes, IMT
Trygve Olav Fossum, IMT

Norwegian University of Science and Technology
Department of Marine Technology

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MASTER'S THESIS IN MARINE CYBERNETICS

SPRING 2016

for

STUD. TECH. LARS SLETBAKK BRUSLETTO

Computer Vision Based Obstacle Avoidance for a Remotely Operated Vehicle

Work Description

The scope of this thesis is to see if it is possible to develop a computer vision based obstacle avoidance system for Remotely Operated Vehicles (ROVs).

Scope of work

1. Algorithm development
    a. Create an obstacle avoidance algorithm that uses **Computer Vision** to decide if there is an obstacle in front.
    b. Develop an obstacle avoidance algorithm that works on a single camera and use machine learning to decide when something other than "ocean" is in front of the ROV.
2. Test the developed algorithm on an ROV in Trondheimsfjorden. The ROV should avoid a transponder tower placed in its way. The program should be able to communicate to the **LabVIEW** program using **UDP**. The mission is used to verify the performance of the Obstacle Avoidance System.
3. Simulation
    a. Simulate the mission in Trondheimsfjorden
    b. Compare different algorithms developed for Obstacle Avoidance in the simulator environment.

The report will be written in English and edited as a research report including literature survey, description of mathematical models, simulation results, model test results, discussion and a conclusion, including a proposal for further work. Source code will be provided online on GitHub. Department of Marine Technology, NTNU, can use the results freely in its research work, unless otherwise agreed upon, by referring to the student's work. The thesis will be submitted in three copies on July 16.

Adviser: Ph.D. Stein M. Nordnes and Trygve Olav Fossum

Professor Martin Ludvigsen
Supervisor

Preface

This master's thesis is a part of the study program "Engineering and ICT" at the Faculty of Engineering Science and Technology, with specialization within Marine Cybernetics offered by the Department of Marine Technology. The work was carried out in the spring semester of 2016. This thesis work is concerned with the development of Computer Vision based Obstacle Avoidance system for an (ROV). Part of the work tackles the development of a simulation environment to test different computer vision methods for (AUR-lab). The simulation environment is called AURlabCVsimulator.

Trondheim, June 20, 2016

Lars Sletbakk Brusletto

Abstract

Remotely Operated Underwater Vehicle (ROV) has been operating for many years. The use of ROV is increasing with the increased activities in sub-sea operations. These operations are today dominated by oil and gas offshore activities, aquaculture and mining industry.

There exist a large potential in automating repetitive tasks, for example getting the ROV to the operational destination. It is a good starting point for developing a system to automate that process. Such an ROV‘s are operated by a person. The purpose of this thesis is to develop a system that can allow this ROV to move towards its desired destination and to find a method to avoid obstacles which might be present under water

A common sensor for almost all ROV‘s is the camera. Therefore, it is advantageous to develop a system that utilizes these cameras. Another benefit is the huge amount of research and tools that are already accessible from the field of computer vision. The camera is a cheap sensor, and it is already on all ROV‘s. The main benefits are the vast amount of research and tools already accessible from the field of computer vision.

The challenge is how to benefit from this sensor. This sensor can observe lights in color and brightness. This thesis evaluates different methods of analyzing images and developing methods to use the signals to avoid obstacles when it moves underwater

Three methods have been developed. One method uses two cameras(disparity method). The two other methods need only one camera. Part of the work was concerned with developing a simulation environment to test different computer vision methods for (AUR-lab). This simulation environment is called AURlabCVsimulator. All methods have been successfully tested in the simulation while the disparity method has been proved to work underwater in Trondheimsfjorden. During this trial in Trondheimsfjorden, the ROV operated and detected an obstacle and its center and gave directions to the control program to exit path.

This trial demonstrates that cameras can be used for underwater obstacle avoidance. Further work can be extended to more sophisticated methods and learn how to read textures such as pipelines, subsea equipment and fishing equipment.

There are many interesting applications and options for further work on the methods in this thesis.

The code developed is available and can easily be extended for other users.

Acknowledgements

I would like to thank my supervisor Prof. Martin Ludvigsen for arranging the mission in Trondheimsfjorden. He provided me with feedback during the project period. He also gave me space to develop myself independently, and this has given me great confidence in myself.

Further, I want to thank my co-supervisors, Ph.D. candidate Stein M. Nordnes, and Ph.D. candidate Trygve Olav Fossum for the academic discussions and help during the work of this thesis.

Stein M. Nordnes has taken care of the hardware of coupling the electronics between the camera and the underwater pressure chamber. He also helped me in the discussion on computer vision. He always had the time to answer my questions and pointed me the right direction when needed.

Trygve Olav Fossum helped immensely in discussions around how the "computer vision program" and the "path program" should be coupled.

I want to thank the M.Sc. student Ida Rist-Christensen for the motivating dialogues concerning the mission where we had to cooperate, as well as in deciding upon the format for communication between the "computer vision program" and the "path program."

I am grateful to the crew of R/V Gunnerus who made the sea trials possible. They were always friendly and helpful and made the period on the ship a great experience.

I also want to thank Adrian Rosebrock for sharing his knowledge on his computer vision blog http://www.pyimagesearch.com/; it has given me new ideas and motivation throughout this thesis.

Finally, I would like to thank the open source community of OpenCV, which has shared open source algorithms and tutorials on how one can implement computer vision methods in python. The OpenCV community has provided open source algorithms and packages vital for the success of this thesis.

Lars Brusletto

Table of contents

List of Figures

List of Tables

Nomenclature

$AUR-lab$	Applied Underwater Robotics Laboratory
$AurlabCV simulator$	Applied Underwater Robotics Laboratory Computer Vision Simulator
AUV	Remotly Operated Vehicle
AVT	Allied Vision Technologies
$BRISK$	Binary Robust Invariant Scalable Keypoints
CPU	Central Processing Unit
DP	Dynamic Positioning
DVL	Doppler Velocity Log
$GLCM$	Gray Level Co-occurrence Matrix
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HIL	Hardware In the Loop
HSV	Hue, Saturation, and Value
$LabVIEW$	Laboratory Virtual Instrument Engineering Workbench
LBP	Local Binary Pattern
ML	Machine Learning
$OpenCV$	Open Source Computer Vision

RGB	Red Green Blue
ROV	Remotly Operated Vehicle
SDK	Software Development Kit
SDK	Software Development Kit
$SIFT$	Scale Invariant Feature Transform
$SLIC$	Simple Linear Iterative Clustering
SSD	Sum of Squared Differences
$SURF$	Speeded Up Robust Features
SVM	Support Vector Machines
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Chapter 1

Introduction

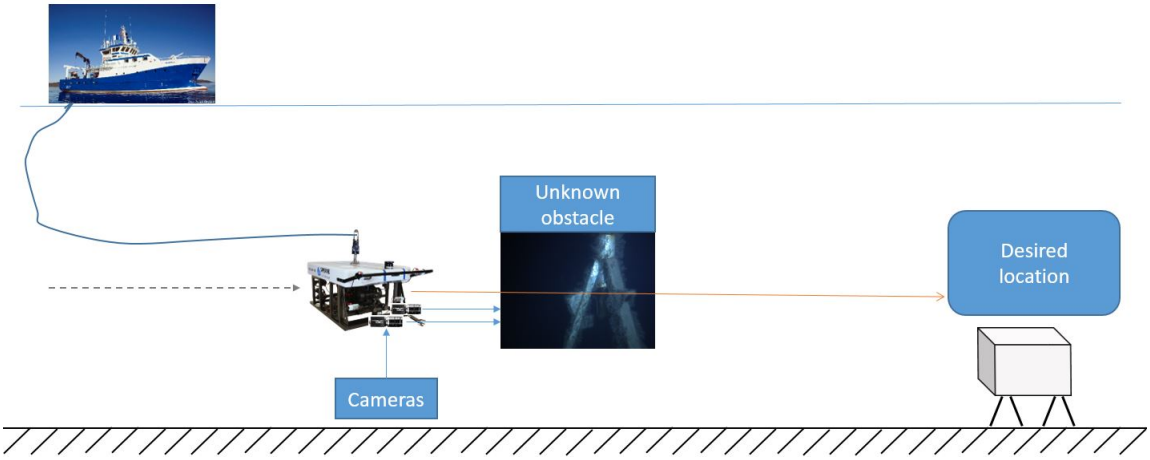


Figure 1.1: Goal of the work

In figure 1.1, a possible scenario in an underwater operation is visualized, the Remotely Operated Vehicle is autonomously controlled to move to its desired location. Unfortunately, there might be unknown obstacles in it‘s path. This thesis is concerned with the development of software that achieves real-time underwater obstacle avoidance in such situations.

This work is about obstacle avoidance methods that work under abnormal conditions, in particular, low light, scattering, blurring or imprecise camera calibration.

1.1 Motivation

If we want to get from A to B safely, we must make sure to avoid all obstacles in the way. Sonar could work well in such a setting. However, the benefits of using the camera sensor is the substantial amount of information captured. One such example is how well we can navigate using the information from our eyes and understand our environment. A camera is also inexpensive since they are manufactured at low costs in comparison to other sensors. The camera based obstacle avoidance system for the AUR-lab is a starting point for further development of camera-based marine applications. This is advantageous for implementing artificial intelligence for underwater robotics.

1.1.1 Autonomy Group

Professor Martin Ludvigsen started a project in the spring quarter of 2016 to create a system that can perform robotic autonomous interventions. The group consists of the following members

Table 1.1: Members of Autonomy Group

Members
Professor Martin Ludvigsen
PhD candidate Stein M. Nornes and
PhD candidate Trygve O. Fossum
MsC student Ida Rist-Christensen
MsC student Lars Brusletto

The authors contribution were to implement a computer vision system that streams and processes image data. It is a part of the autonomy framework of the group. The result of the group's work is presented in the paper A for the Ocean's 16 conferences. The author has cooperated with MsC student Ida Rist-Christensen on the part of the project where the computer vision system sends information to the LabVIEW program as explained further in the chapter 4.

1.1.2 Overview of goal of autonomy group

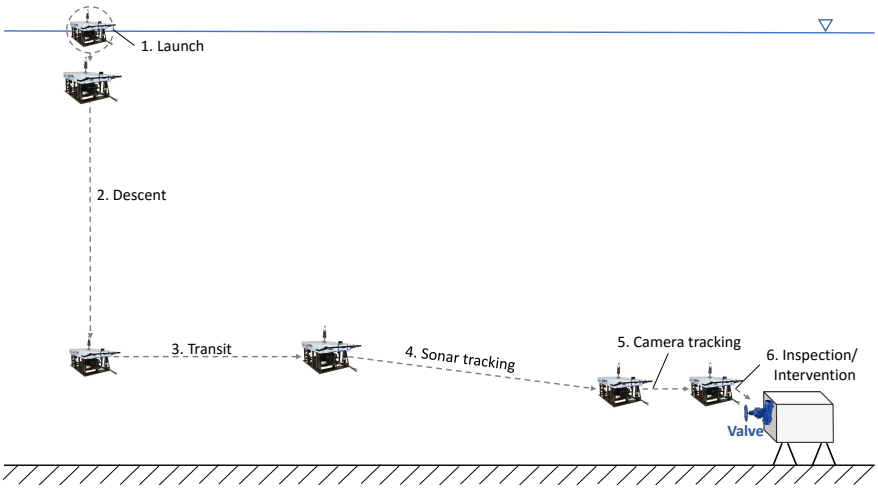


Figure 1.2: Plan of mission. Courtesy of (Rist-christensen, 2016).

The figure 1.2 illustrates main (ROV) tasks. The computer vision system will be used to avoid obstacles it meets on it's way under the following steps transit, sonar detection and tracking, camera detection and tracking and inspection/intervention. These steps are enumerated as 3,4,5,6 in the figure 1.2.

1.2 Scope and Limitations

The goal of this thesis is to develop an obstacle avoidance system for NTNUs camera based underwater robots. The obstacle avoidance system utilizes the camera as its only sensor. This thesis revolves around developing this system for the ROV 30 k, with its camera aimed straight forward. A sea trial was conducted to test the performance of the scheme and to gather images for further simulations.

Several obstacle avoidance systems have been developed for drones. However, they weren't designed to face our challenges. In fact, in an underwater environment, images lack salient features and suffer from noisy blur and backscatter.

1.2.1 Objectives

The primary objectives of this thesis are presented in the list below:

- Design and development of the obstacle avoidance method.
- Development of simulation environment module.
- Conduct a mission with the ROV to test its performances in real life conditions.

The details of each objective will be investigated further after the necessary background material and theory has been introduced to the reader.

1.3 Contributions

The contributions are presented in the list below:

- Development of the Disparity method. Section 3.1 presents the algorithm in details.
- Development of the LBP method. An algorithm for obstacle avoidance using ML has been developed. Section 3.2 presents the algorithm in details.
- Development of the Haralick method. An algorithm for obstacle avoidance using ML has been developed. Section 3.3 presents the algorithm in details.
- Development of simulation environment module that has been used to test computer vision methods that, i.e., can be used for obstacle avoidance.An overview is given in section 4.2.

1.4 Organization of the Thesis

Chapter 1 introduction of the thesis.

Chapter 2 introduces the background material for obstacle avoidance. A short introduction to research attempts on obstacle avoidance and other relevant work is given.

Chapter 3 describes and illustrates the methods developed in this thesis.

Chapter 4 describes how the methods developed in chapter 3 has been tested in field test in the Trondheimsfjord and simulations.

Chapter 5 presents the results acquired throughout this thesis

Chapter 6 discuss results and errors in the results chapter.

Chapter 7 concludes the thesis and suggests further work.

Appendix A paper proposed for the Ocean 16 conference.

Appendix B lists the attachments that are delivered electronically in DAIM.

Chapter 2

Background

This chapter is an introduction to software, hardware, theory and previous work.

2.1 Previous Work related to underwater obstacle avoidance

Autonomous underwater obstacle avoidance is a challenging problem. If solved robustly, it will result in several applications in the field of robotics and marine engineering. Previous work that inspired this thesis is presented. The literature can be split into over-water and underwater obstacle avoidance. The fundamental difference between the two is that an underwater environment has a sparse set of reliable features that can be exploited for obstacle avoidance.

2.1.1 Over-water obstacle avoidance

Pushburst (Barry and Russ 2015) method

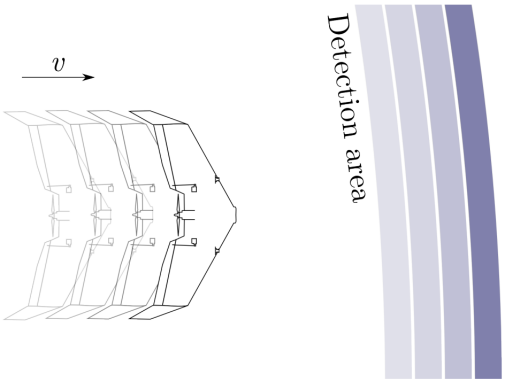


Figure 2.1: Conncept of the pushboom method (Barry & Tedrake, 2015).

High-speed obstacle avoidance for drones has been developed using Block matching algorithm as in (Barry & Tedrake, 2015). In their paper, they describe how they detect the single depth (marked in (Figure 2.1) as dark blue) and integrate the vehicle's odometry and its past detection (marked as lighter blue in the figure). Thus, the method builds a full map of obstacles in front of the vehicle in real time.

(Mori, Tomoyuki, and Sebastian 2013) method

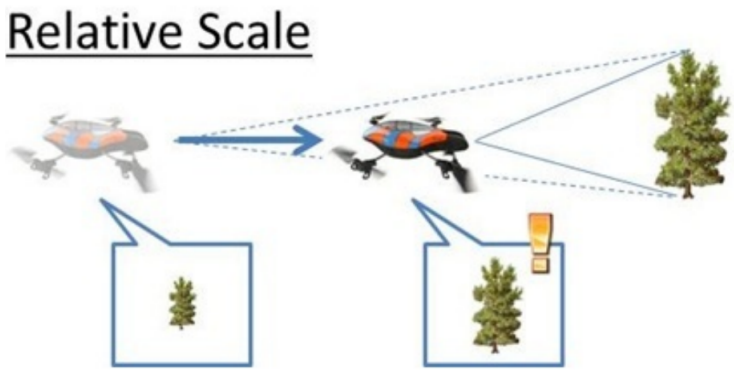


Figure 2.2: Figure of consept using relative scale (Mori & Scherer, 2013).

Another method utilizes feature matching based on SURF(Speeded Up Robust Features) descriptor, combined with template matching to compare relative obstacle sizes with different image spacing. This method is described in detail in (Mori & Scherer, 2013) , see (Figure 2.2). Unfortunately, it was only tested on a single type of obstacle configuration and hence is not a robust solution.

2.1.2 Underwater obstacle avoidance

(Kostavelis, Nalpantidis, and Gasteratos, 2009.) method

Figure 2.3: Disparity map and image divided into 3 parts (Kostavelis et al., 2009).

This work is also inspired by (Kostavelis et al., 2009). Their method is to calculate the disparity map and then calculate the mean value of pixels in 3 parts of the image, left, center, and right. From the calculated mean value, it detects the dominant obstacle before it applies an exit strategy. See (Figure 2.3) to observe their decision strategy. Unfortunately, their method is quite slow. It needs 2.317 seconds to complete its disparity method and make a decision.

The (Rodriguez-Teiles et al. 2014) method

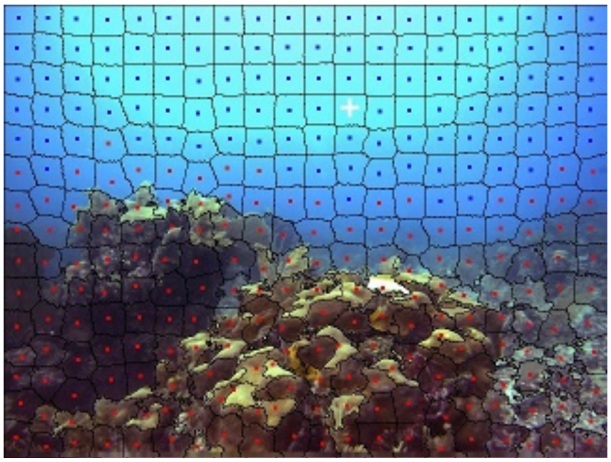


Figure 2.4: Super pixel method applied to image of a reef and ocean (Rodriguez-Teiles et al., 2014).

In their work, they made an obstacle avoidance system for a robot so it autonomously could move around a coral reef without crashing. Their method segments the image and retrieves a histogram for each segment. The latter is used to predict whether the segment belongs to an obstacle. See figure 2.4. The prediction is performed using a pre-trained classifier. Unfortunately, their method seems to work poorly in different lighting condition, this is since the histogram of lab space is not illumination invariant. In the following figure 2.5, notice how bad it is at classifying in poor lighting conditions

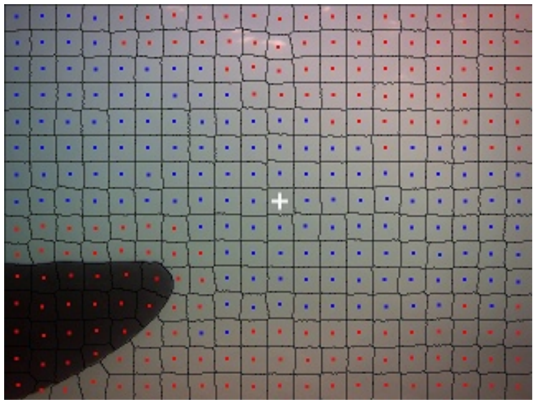


Figure 2.5: Super pixel method applied to image with a lighter background (closer to surface) (Rodriguez-Teiles et al., 2014).

In the figure 2.5, the red dots represent segments centers classified as obstacles; blue dots is classified as "ocean".

The SLIC Superpixel method allows the user to adjust to the local water conditions by training the classifier to those conditions before it's mission.

2.1.3 Summary of related work

There are several papers on obstacle avoidance and computer vision, in this section the reader has been introduced to the documents that inspired the work of this thesis the most.

To summarize, the methods all have pros and cons, the underwater methods have been tested and works in an marine environment, and therefore seems as a great starting point for further work, while the over-water methods has given the thesis inspiration for might be possible underwater. Especially the advanced method described in (Barry & Tedrake, 2015) has given ideas for how to improve disparity calculation.

2.2 Software

There are different software that has been chosen as the tools to develop the obstacle avoidance methods. In this section a short presentation of them is given.

2.2.1 OpenCV

(OpenCV) is an image processing library originally developed by Intel, now maintained by Willow Garage. It is open-source, multi-platform and is implemented in C and C++, and there are wrappers available for several languages, including C#, Python, Ruby, and Java

2.2.2 Mahotas

Mahotas is a computer vision package created for cellular image analysis. However, the package includes traditional image processing functionalities as well as more advanced computer vision functions such as feature computation. In this thesis, the Haralick feature descriptor has been of interest after reading in (R. M. Haralick et al., 1973) how it has been used for classification from satellite images. Furthermore, the interface of Mahotas is in python, which makes it easy to use, and it is fast since the underlying code is implemented using C++ and optimized for speed. Among other algorithms, the implementation of the Haralick texture descriptor (documented in (Coelho, 2013a)) is faster than the one implemented in Scikit-image

2.2.3 Scikit-image

Scikits-image is a collection of algorithms for image processing (*scikit-image: Image processing in Python — scikit-image*, 2016). It is written in Python and Cython. It is used for the feature extraction of the Local Binary Pattern algorithm and for implementing the SLIC Superpixel Segmentation as described in the (Rodriguez-Teiles et al., 2014) paper.

2.2.4 Scikit-learn

Scikit-learn is a fast library for support vector classification. In the paper (Pedregosa et al., 2012), is is documented that it has a faster run-time than the following

library's: mlpy, pybrain, pymvpa, mdp and shogun .

LinearSVC inside of Scikit-learn is written using LibSVM and implemented by (Chang and Lin, 2001)

2.2.5 Python

Python is an open-source, high-level programming language. It allows for the use of multiple programming paradigms, ranging from object-oriented to functional programming. In addition to OpenCV, two libraries for Python have been utilized, NumPy and Matplotlib. NumPy is an open-source math library. It includes support for arrays, matrices and a large set of mathematical functions for these. Matplotlib is a 2D/3D plotting library. The reason for using Python is the extensiveness of its standard library. Moreover, as a scripting language, it is well suited for rapid prototyping. It is not as fast as C++, but in this situation, quick development is more important than processing speed.

2.2.6 Profiling with SnakeViz

SnakeViz is a open source project available on github at:

https://github.com/jiffyclub/snakeviz.

SnakeViz is a profiling software. "Profiling is a form of dynamic program analysis that measures, for example, the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls" (*Profiling (computer programming) - Wikipedia, the free encyclopedia*, 2016).

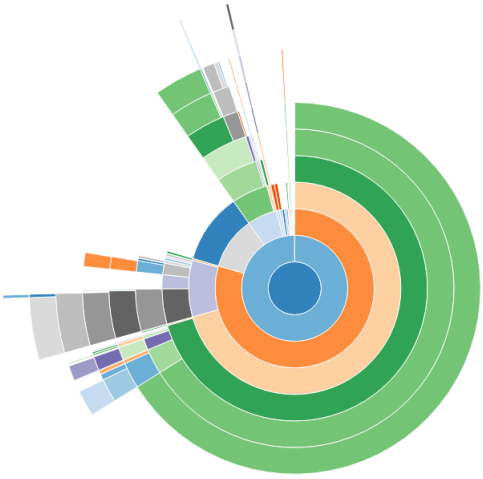


Figure 2.6: SnakeViz illustates the profiling of different parts of the program

In figure 2.6, we can see that snakeViz allows a graphical runtime analysis of Python programs. SnakeViz has been used for optimization of the programs, and to compare their runtime.

2.2.7 Vimba Software Development Kit

Vimba SDK is an SDK maintained and developed by Allied Vision Technologies (AVT). The software is used to communicate and capture images from the AVT cameras.

2.2.8 Pymba

Pymba is a open source project available on github at: https://github.com/morefigs/pymba

Pymba is a Python wrapper for the Vimba SDK written in c++.

To make the computer communicate and receive images from the camera we first needed to install the Vimba SDK from AVT(Allied Vision Technologies), and then install the AVTDriverinstaller to install the AVT Vimba drivers on my computer.

We then used "pymba" (*morefigs/pymba: Python wrapper for the Allied Vision Technologies (AVT) Vimba C API*, 2016) a Python wrapper for the Vimba

SDK written in c++, so we could do all the communication using Python. The pymba folder is then downloaded and placed in the same repository as the project.

2.2.9 Summary

Software libraries used are summarized in table 2.1 below.

Table 2.1: Software libraries

library characteristics
Python: 2.7.11 —Anaconda 2.5.0 (64-bit)
scipy: 0.17.0
numpy: 1.10.4
matplotlib: 1.5.1
pandas: 0.17.1
sklearn: 0.17
pymba

2.3 Equipment for the test

To be able to do the field test in Trondheimsfjorden (section 4.1) the program used certain equipment and will therefore be presented.

2.3.1 PC PLATFORM

The main development and testing were all done on Acer Aspire VN7-792G Here is an overview of the computer hardware used in this thesis.

Table 2.2: Computer configuration used for this thesis

Computer characteristics	
Processor	Intel Core i7-6700HQ CPU @ 2.60GHz , 2601 Mhz, 4 Core(s), 8 Logical Processor(s)
PC Model	Aspire VN7-792G
RAM	8 GiB (DDR3), 1600MHz
OS	Windows 10 64-bit
Graphic card	NVIDIA GeForce GTX 960M

2.3.2 Boat and ROV

(a) *Image of Gunnerus*

(b) *Image of ROV SUB-fighter 30k*

Figure 2.7: *Boat and ROV. Courtesy of Aur-lab*

The computer vision system was tested April 2016 in the Trondheim Fjord onboard of the boat Gunnerus seen in figure 2.7a. The cameras were mounted front facing on the ROV SUB-Fighter 30k from Sperre AS seen in 2.7b.

The ROV seen in figure 2.7b is operated from an onboard control room within a container on Gunnerus. From the control room on can control the vehicle in manual and automatic control modes. On top of the container, there is placed a winch that holds the tether of the ROV.

The ROV operator could take over the control of the ROV at any time in case something would happen. When the ROV operator takes control of the ROV, the ROV fist goes into “stationkeeping” mode. A mode where the ROV stops, I.e its keeps its position at the same location and depth.

2.4 Camera

The camera is a relatively cheap sensor that captures extensive information in the form of an image or a sequence of images.

Figure 2.8: GC1380 camera model, courtesy of ("High sensitivity 1.4 Megapixel CCD camera with GigE Vision", n.d.)

The model name of the camera used on the ROV is Prosilica gc1380c (seen in figure 2.8) and is produced by Allied Vision Technologies.

Figure 2.9: Image of ethernet port

In figure 2.9, the ethernet and power port is displayed. The ethernet cable is connected to from the computer to the camera.

A selection of specs from the specification in ("High sensitivity 1.4 Megapixel CCD camera with GigE Vision", n.d.) is shown below:

- The camera resolution is 1360×1024
- the resolution is 1.4 megapixel
- it's a global shutter (see section 2.4.2)
- Sony ICX285 2/3" Progressive scan CCD
- ethernet cable entry

2.4.1 Stereo rig

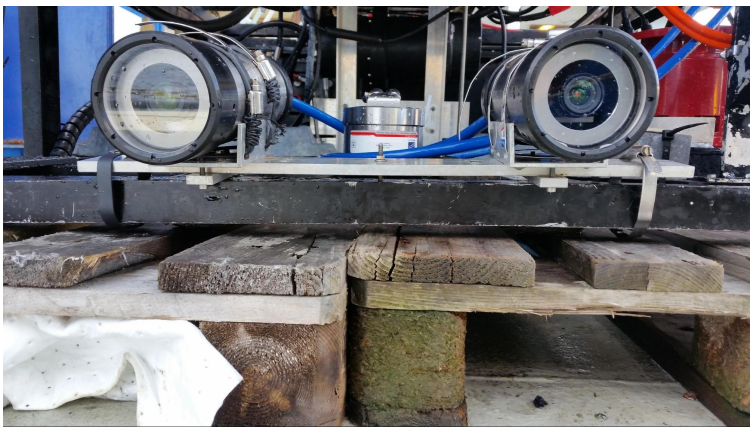


Figure 2.10: cameraRigMinerva

The stereo rig used in the disparity method is shown in figure 2.10. The method needs two cameras to extract depth infromation from the scene. Only the camera to the right in figure 2.10 has been used for the LBP and Haralick method.

We use two of these cameras (figure 2.8) and they are placed inside camera housings as shown above in figure 2.10.

They are placed within a camera housing since it protects them from water leaks and high pressure that can destroy the camera.

Please note how the camera is mounted front facing in on the ROV in figure 2.10.

2.4.2 Global shutter

A camera has typically either a rolling or global shutter. The important difference is that the global shutter allows all the pixels on its sensor to capture light at the same time. This means that if something moves in the image, one will not get any skew and stretched image effects that might happen using a rolling shutter.

(a) *rolling shutter* (b) *global shutter*

Figure 2.11: *rolling compared to global shutter, courtesy of* `http://www.red.com/learn/red-101/global-rolling-shutter`

In figure 2.11, two images of a gun is captured with the two different methods. Please note how the rolling shutter captures distortion effects.

Distortion effects caused by rolling shutter as mentioned in (*Rolling shutter - Wikipedia, the free encyclopedia*, 2016).

- Wobble
- Skew
- Smear
- Partial Exposure

Because of these problems, it‘s advantageous for the obstacle avoidance program to use a camera with a global shutter.

2.5 Pinhole camera

The pinhole camera is a simple concept and it serves as a useful model to describe the functionality of a camera.

The sensor used in computer vision is the camera and since the field estimates the world around us, one need to relate a 2D image to the real 3D world. A simple model of this relationship can be exemplified by the pinhole camera and its model.

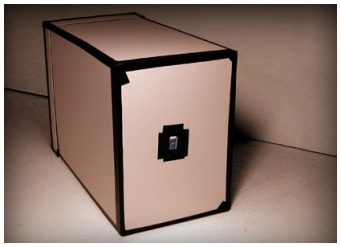


Figure 2.12: pinhole camera, image from http://www.lbrainerd.com/courses/digital-photography/pinhole-cameras

In figure 2.12 a basic pinhole camera is shown, it is basically a box that only lets light through a tiny hole (that usually has been made using a pin) that hits a piece of film inside the box.

It is a simple concept and is at the basis of what a modern camera is. In modern camera the film is replaced by digital image sensors that captures the light that gets through its lens. In a pinhole camera there is no lens.

2.5.1 Modeling of pin-hole camera

In this section we will first introduce the intrinsic camera parameters, this is the parameters describing the camera‘s inside. Then we introduce the extrinsic parameters that models the heading(rotation) and position(translation) of the camera.

Intrinsic camera parameters

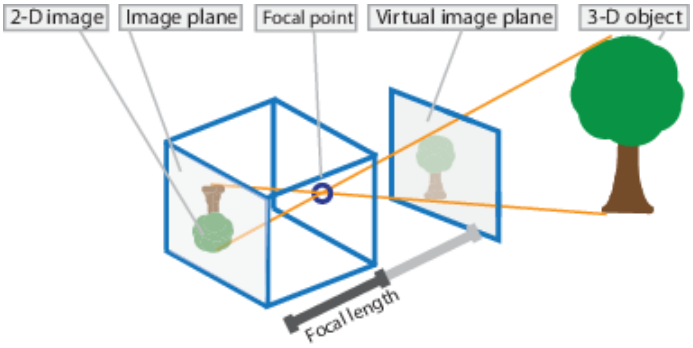


Figure 2.13: Pinhole model. Image from http://se.mathworks.com/help/vision/ug/camera-calibration.html

In figure 2.13, the relationship between a pinhole camera viewing a 3D object and the projected 2D object is illustrated. One needs a mathematical model to relate the real 3D world with the 2D image plane projected (often called perspective projection) onto the camera.

The parameters describing the camera‘s inside is called intrinsic. The parameters represent the focal length, the optical center(principal point) and the skew coefficient.

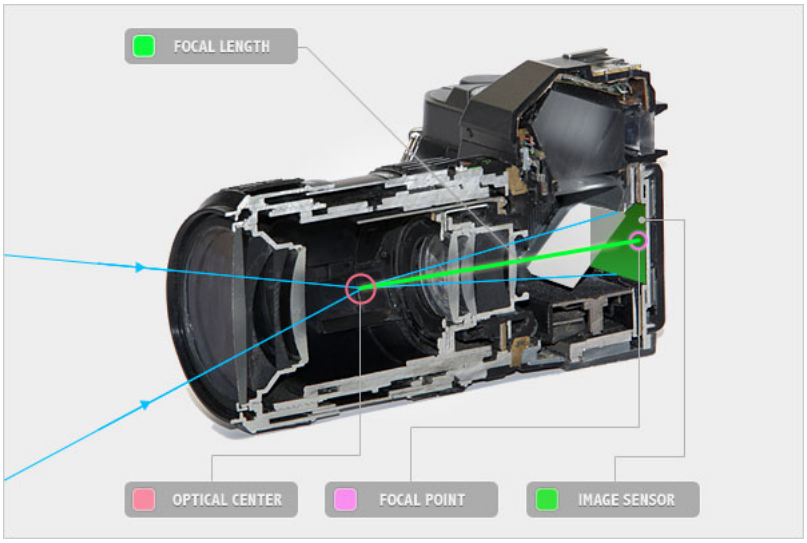


Figure 2.14: Example of inside of a camera. Courtecy of (*Lens Basics — Understanding Camera Lenses*, 2016)

Focal length: f_x, f_y

In figure 2.14 the focal length is illustrated with the green line. The focal length is the distance from the optical center (notice that this is the same as the point of the pin hole in the pinhole camera) to the focal point on the image sensor.

In an ideal camera, the length of f_x is equal to f_y, but one often get a different result, this is caused by flaws in the digital camera sensor and can be the following:

- Distortion from lens
- Noise from inaccurate digital camera sensor.
- Inaccurate camera calibration

Principal Point Offset (optical center): c_x, c_y

Even though the optical center is not on the image plane in figure 2.14, it is represented in (x,y) direction as it was the focal point on the image sensor. Acctualy it lays on the principal plane and is parallel to the image plane

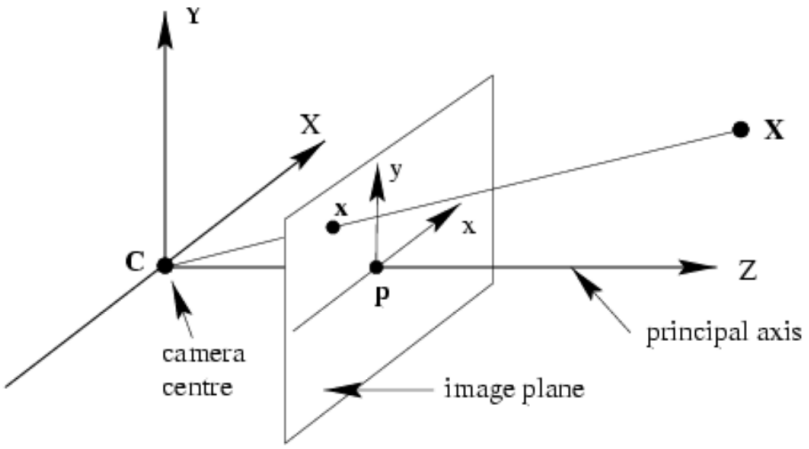


Figure 2.15: Example Principal Point. Courtesy of (Navab, 2009)

located at the center of the image In figure 2.15 the principal point offset is exemplified, it is placed at center of the image at point p.

Axis Skew: s

Skew causes shear distortion in the projected image.

When there is no skew the pixels in the ismage sensor are all perfectly squared as the ratio between height and width is 1 : 1.

Intrinsic matrix

The intrinsic matrix is represented as in (Hartley & Zisserman, 2004) as:

$$K = \begin{bmatrix} f_x & s & c_X \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Please note that the matrix K is the way OpenCV wants to represent the intrinsic matrix and is used in 3.1.3 when the program loads in the camera calibration parameters.

Extrinsic camera parameters

Extrinsic means external or outside of, and in this case it is how the camera is located in the world or 3D scene. The external components is the rotation matrix and the translation matrix.

There are many ways to represent the extrinsic parameters according to (*Dissecting the Camera Matrix, Part 2: The Extrinsic Matrix*, 2016) and is summarized bellow.

- World centric − how the world changes relative to camera movement
- Camera-centric − how the camera changes relative to the world
- Look-at − how the camera's orientation is in terms of what it is looking at

The world centric view can be thought of as the rotation matrix rotates the camera from its center. And the tangential moves the camera up-down, left-right, forward-backwards.

The extrinsic matrix in square form is given as:

$$\left[\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} I & t \\ \hline 0 & 1 \end{array}\right] \times \left[\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right]$$

Where the rotation matrix "R" is given as:

$$R = \begin{bmatrix} r_{1,1} & r_{1,2} & r_{1,3} \\ r_{2,1} & r_{2,2} & r_{2,3} \\ r_{3,1} & r_{3,2} & r_{3,3} \end{bmatrix}$$

And the translation vector "t" (the position of the world origin in camera coordinates) is given as

$$t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

The extrinsic matrix describes how to transform the points in the the 3D world to camera coordinates. It's a way to describe how the world is transformed relative to the camera.

In section 3.1.1 the result of the camera calibration is given and it shows the calculated rotation and translation parameters.

2.6 Distortion

The pinhole camera model presented in section 2.5 is an approximation to the real world. But that model does not take into account the distortion caused by the lens. The reason is that the pinhole model only look at how the light from a 3D world is projected onto the 2D image plane. It gets a whole lot more complicated when light rays get bent by the lens. In underwater image calibration it is even more difficult, since we have to deal with refraction as explained in more detail in section 2.6.3.

The distortion coefficients calculated from the camera calibration is presented in 3.1.1. The way these parameters are loaded into OpenCV using python is explained in 3.1.3.

2.6.1 Radial distortion

Radial distortion is a common distortion in cameras and comes from the shape(symmetry) of the lens.

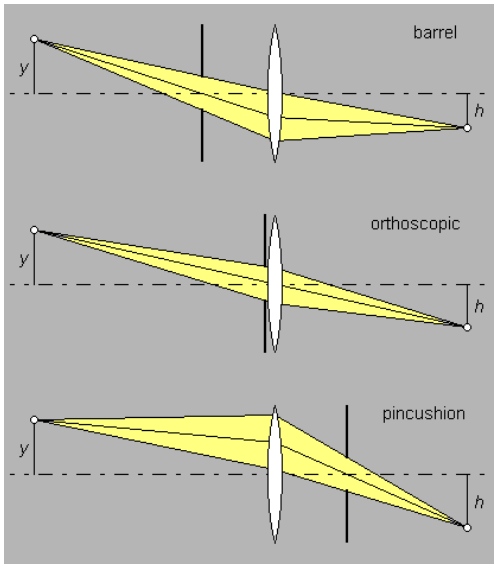


Figure 2.16: Example of barrel, orthoscopic and pincushion effect. Image from http://toothwalker.org/optics/distortion.html

In figure 2.16 the barrel and pincushion distortion causes is illustrated. When the lens is not at the optimal spot it leads to these kinds of distortions. In the barrel and pincushion distortion it caused by the light to go through either side(

closer to the edges) of the optical center. Orthoscopic means free from optical distortion and happens when the light rays go through the optical center.

Radial distortion coefficients

In (*What Is Camera Calibration? - MATLAB & Simulink - MathWorks Nordic*, 2016) it is given a good explanation of the radial distortion coefficients, and the following are from that source.

> The radial distortion coefficients model this type of distortion. The distorted points are denoted as $(x_{distorted}, y_{distorted})$:
>
> $$x_{distorted} = x(1 + k_1 * r_2 + k_2 * r_4 + k_3 * r_6)$$
>
> $$y_{distorted} = y(1 + k_1 * r_2 + k_2 * r_4 + k_3 * r_6)$$
>
> - x, y — Undistorted pixel locations. x and y are in normalized image coordinates. Normalized image coordinates are calculated from pixel coordinates by translating to the optical center and dividing by the focal length in pixels. Thus, x and y are dimensionless.
> - k_1, k_2, and k_3 — Radial distortion coefficients of the lens.
> - $r^2 : x^2 + y^2$
>
> Typically, two coefficients are sufficient for calibration. For severe distortion, such as in wide-angle lenses, you can select 3 coefficients to include k_3.

Barrel distortion

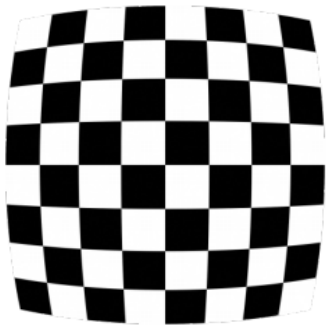


Figure 2.17: Barrel distortion

In figure 2.17, it looks as if the image has been stretched around a ball. Note that this effect is utilized in typical fisheye lenses like gopro that gives a wide lens angle, in that case the advantage is that it captures more since it has a wider field of view. But if one want to utilize image for computer vision one wishes to have a close as possible approximation to the real world.

Pincushion distortion

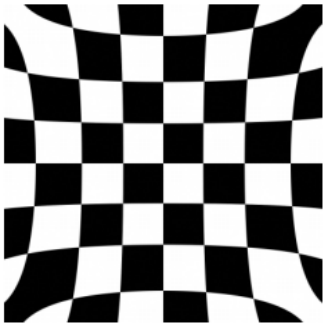


Figure 2.18: Pincushion distortion

In figure 2.17, it looks as if the image has been stretched on the inside og a ball, it is the opposite effect of the barrel distortion. And this also exemplifies that they they are the opposite poles of each other.

2.6.2 Tangential distortion

The lens and the image plane needs to be parallel. When they are not, tangential distortion appears. The tangential distortion coefficient describe how they are non parallel.

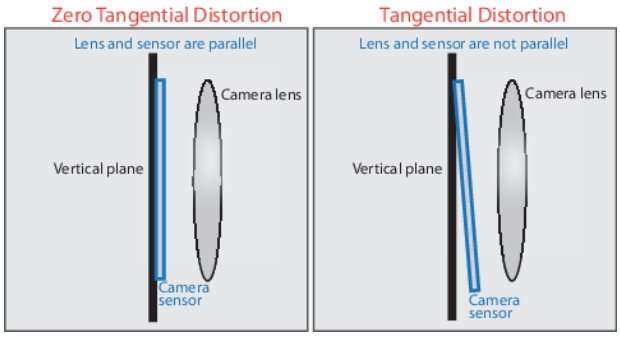


Figure 2.19: Cause of tangential distortion. Courtesy of http://se.mathworks.com/help/vision/ug/camera-calibration.html

The cause of tangential distortion comes from nonparallel lens and image plane and is showed in figure 2.19 above.

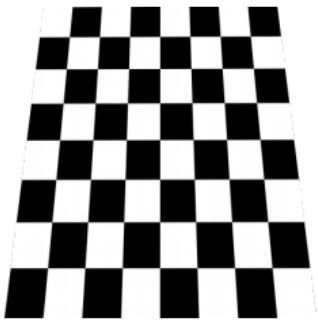


Figure 2.20: Tangential distortion

In figure 2.20, an example of how an image of a checkerboard gets images when the camera is affected by tangential distortion.

Tangential distortion coefficients

In (*What Is Camera Calibration? - MATLAB & Simulink - MathWorks Nordic*, 2016) it is given a good explanation of the Tangential distortion coefficients, and the following are from that source.

The distorted points are denoted as $(x_{distorted}, y_{distorted})$:

$$x_{distorted} = x + [2 * p_1 * x * y + p_2 * (r_2 + 2 * x_2)]$$

$$y_{distorted} = y + [p_1 * (r_2 + 2 * y_2) + 2 * p2 * x * y]$$

- x, y — Undistorted pixel locations. x and y are in normalized image coordinates. Normalized image coordinates are calculated from pixel coordinates by translating to the optical center and dividing by the focal length in pixels. Thus, x and y are dimensionless.
- p_1 and p_2 — Tangential distortion coefficients of the lens.
- $r^2 = x^2 + y^2$

2.6.3 Refraction

Figure 2.21: Refraction:air-glass-water image. Courtesy of Carroll Foster/Hot Eye Photography

When light goes from one medium to another it bends as illustrated in figure 2.21.

The underwater camera housing has a lens and is underwater, the light will therefore before it hits the light sensor within the camera. This causes difficult distortion parameters, and can cause bad calibration result as such as setup is not widely used, and therefore not taken into account in most camera calibration software.

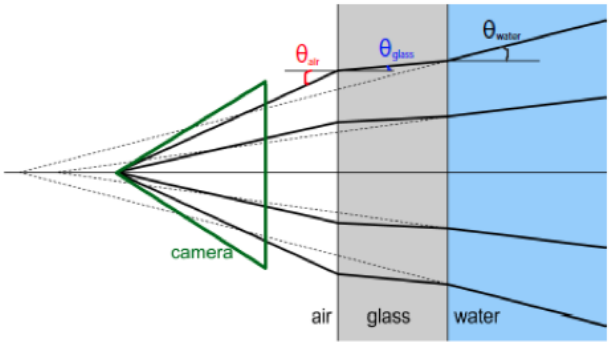


Figure 2.22: Refraction:air-glass-water interface (Sedlazeck and Koch, 2011)

As seen in figure 2.22 we can see that light breaks and change its direction by different angles when it crosses different mediums. Refraction is described by Snell's law in (Wesley, 2000)

$$n_{air}\sin(\Theta_{air}) = n_{glass}\sin(\Theta glass) = n_{water}\sin(\Theta water) \tag{2.1}$$

In equation 2.1 the n_{air} and $n_{glass}b$ are a ratio of refraction while Θ_{air} and $\Theta glass$ are the angles the ray enters the material from it's medium as seen in figure 2.22.

This refraction causes image points to be placed outward in the image plane.

2.7 Calibration software

Camera calibration is a way to estimate the intrinsic and extrinsic camera parameters and the distortion coefficient. The distortion coefficients is used to improve the projected 2d image after it is captured by doing a process called rectification.

Calculating the camera parameters is a complicated task, the reader will be introduced to four different software while an evaluation of these will be described in section 3.1.1.

2.7.1 The general idea

By taking pictures of objects we already know the size of it is possible to estimate these parameters. A checkerboard is a very popular object to use in such

a task, and has been used in this thesis. There are different way to calculate the parameters. The user can calibrate the camera based on a set of checkerboard images automatically or calibrate the camera in a step by step procedure. The disadvantage of the automatic approach give the user less control, while the step by step gives the user more control but it's also more time consuming.

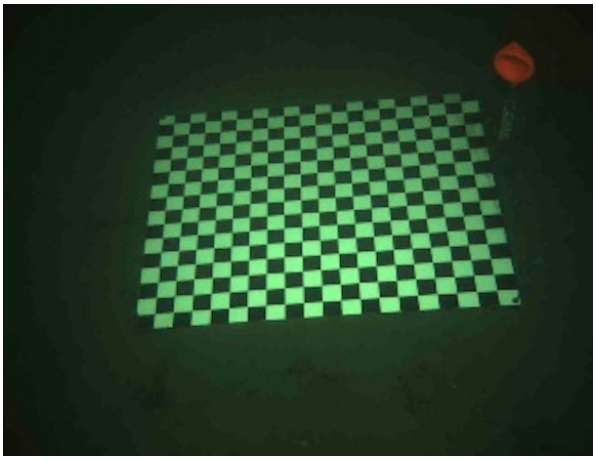


Figure 2.23: Checkerboard is used underwater to calibrate camera for distortion

In figure 2.23 you can see the checkerboard at approximately 70 meters of depth. It is used for calibration of the stereo camera. Please note that the stereo camera in this example creates a pincushion distortion as in figure 2.18.

In the checkerboard example in figure 2.23 the squares are 80mm by 80mm.

2.7.2 MathWorks calibration toolbox

MathWorks calibration toolbox is Matlab's built-in calibration toolbox.

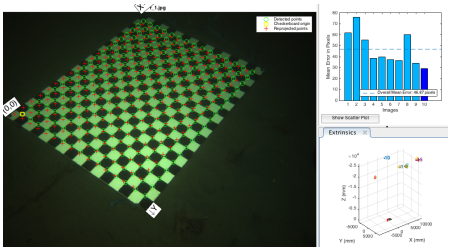


Figure 2.24: Camera calibration in Matlab with mathworks toolbox

In this toolbox the user write "cameraCalibrator" for single camera calibration

or "stereoCameraCalibrator" for stereo camera calibration in the command interface. A GUI will pop up, and the user follows the instructions.

2.7.3 Caltech calibration toolbox

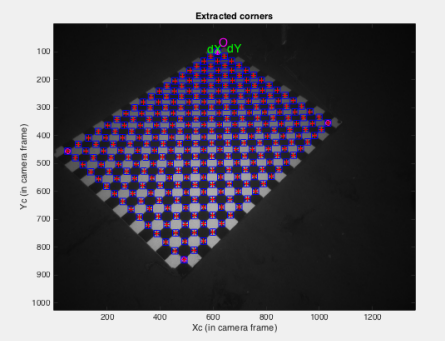


Figure 2.25: Camera calibration in Matlab with calib toolbox

The Caltech calibration toolbox is made by Jean-Yves Bouguet and is implemented in Matlab. The main difference between MathWorks toolbox and this one is that this has more options, and you can supervise the calibration by clicking on corners and other functionality (*Camera Calibration Toolbox for Matlab*, 2016a).

2.7.4 OpenCV calibration

OpenCV has implemented the same calibration algorithms from Jean-Yves Bouguet, but does not contain a GUI functionality like the Jean-Yves Bouguet toolbox implemented in Matlab.

2.7.5 Agisoft PhotoScan calibration

Agisoft PhotoScan (*Agisoft PhotoScan*, 2016) is a commercial software and therefore its hard to know exactly how the calibration is done.

2.8 Underwater camera problems/uncertainties

There are certain difficulties when it comes to underwater imagery one should keep in mind when developing computer vision for underwater applications and in general.

2.8.1 Scatter in underwater images

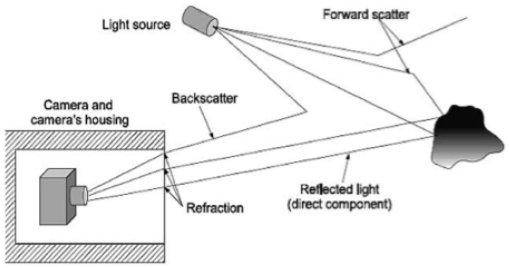


Figure 2.26: Illustration of scatter

To take images underwater, one needs light, therefore at greater depth one need external lights so that the camera can see the environment. These external light source hit particles in the water and they "light up" in the scene in front of the background, see figure 2.26. This is a problem because it makes computer vision techniques such as optical flow and feature detection error prone. We can divide the source of these errors into *back scatter* and *forward scatter*.

The primary source of these errors is scattering. Scattering is the change of the direction of photons due to the different sizes of the particles forming the water. There are two kinds of scattering. *Back-scatter* is the reflection of the light from the light source back to the lens of the camera. A consequence of *back-scatter* is the so-called marine-snow. This consists of small observable particles floating in the water, which creates false positive in the feature extraction and matching processes. *Forward scatter* occurs when a small angle deflects the path of the light source. This leads to reduced image contrast and blurring of object edges.

2.8.2 Scale of the world in the image plane

It is difficult for a computer to understand how far something is from looking at a image. Depth information can be extracted by using two cameras, similar to

us humans who have two eyes. If the computer knows the "extrinsic" parameters of a camera pair it is possible to estimate the distance of objects in the image.

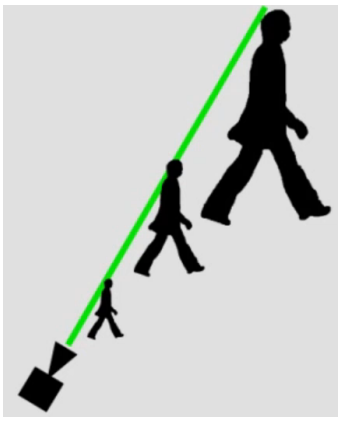


Figure 2.27: example of scaling (*Visual Odometry AUTONAVx Courseware — edX*, n.d.)

In figure 2.27 an illustration of the problem caused by using one camera to try to determine the size of a man in the image.

2.9 Communication between computers

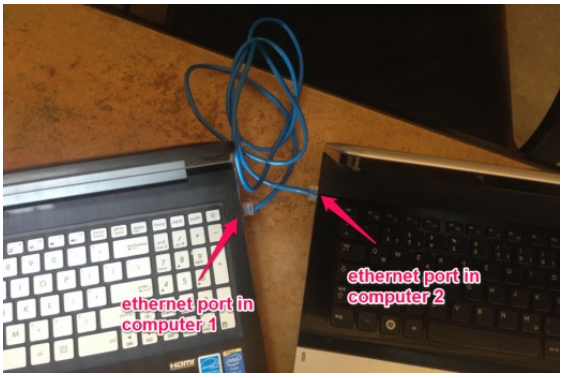


Figure 2.28: Example image of how two computers are connected with ethernet cable. Courtecy of https://usercontent1.hubstatic.com/9072770_f520.jpg

In figure 2.28, it is illustrated how two computers have a ethernet cable connecting their two. Such a cable has been used to make the computer vision program send messages to the LabVIEW program.

Computers that are connected with an Ethernet cable can communicate over different higher level end-to-end protocols (Ethernet being in itself a layer two protocol in the OSI protocol model). (UDP) is a layer-4 or end-to-end transport layer protocol has been used in this thesis so the reader will be introduced to the theory.

2.9.1 User Datagram Protocol

UDP is a:

- simple message-oriented network
- protocol for transferring information
- no message transmission is guaranteed
- no message saved receiving end
- does not require a restart when a connection is lost, but rather a timeout
- system returns to sending and receiving messages after this timeout

As seen in the list above, UDP is an unreliable transport protocol, but it gives almost perfect results, due to the very low bit-error-rate make possible by Ethernet connections.

Even though (TCP) is considered more reliable, it was inconvenient to implement on the LabVIEW side of the communication concerning the time frame. UDP was chosen as it was simpler to implement and it makes the programs less dependent on each other.

2.10 Histograms

Histograms is a way to quantify the frequency distribution of numerical data. Histograms are widely used and is among other used in image search engines often called content-based image retrieval.

Histograms can be used for several things:

- Thresholding
- White balancing
- Tracking object of a distinct color
- Features

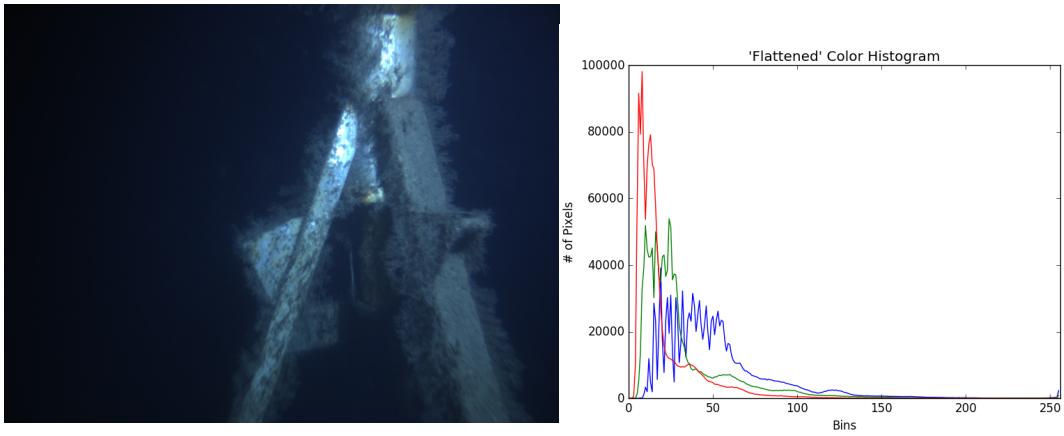
Histograms has been useful to store the LBP and Haralick features for their respective methods.

2.10.1 Color Histograms

With a histogram, one can compare images by similar color distributions.

In a Red Green Blue (RGB) image we have three color channels. When we compute the histogram of each channel we call it a color histogram. In the image below, three color histograms calculated for the picture is illustrated.

To use the histogram of an image is a type of image descriptor.



(a) *Image that we compute the color histogram of*

(b) *color histogram*

Figure 2.29: *Color histogram*

In figure 2.29, an example of an color histogram is shown. A color histogram is a count of the occurrence of the values $0 - 255$ in each channel. Thereby. It has no concept of the shape or texture of objects in the image.

Light sensitive

The lighting is important to note since the histogram of an RGB image will be very different under different lighting conditions. Just think of how a tree looks at night, compared to daytime.

To compare histograms, the images should be taken under the same lighting conditions. comparisons of histograms work best if one acquires the images under same lighting conditions.

Noise sensitive

Color histograms are sensitive to "noise", such as changes in lighting in the environment the image was captured under and quantization errors (selecting which bin to increment). Some of these limitations can potentially be mitigated by using a different color space than RGB (such as (HSV) or L*a*b*).

2.11 Color channel statistics

One way to quantify an image is by using a "color descriptor". There are various ways of describing the color distribution of an image:

- Means
- Standard deviations
- Skew
- Kurtosis

An image is often represented as RGB, and one can look at each channel and compute the statistics to create a list called the feature vector. The mean and standard deviation is a way to try to describe the intensity of an area of the image (a set of pixels).

The paper (Rodriguez-Teiles et al., 2014) describes an interesting method. They compute two feature vectors, one using the standard deviation, and one using a normalized (SSD) measure (though only using the a and b color space). The smaller the Euclidean distance between the two vectors, the more similar images are.

2.12 Feature Extraction

This section explains how one can use image descriptors, feature descriptors, and feature vectors to quantify and abstractly represent an image as a vector of numbers.

One can use feature extraction for multiple computer vision areas. One could compare the extracted features from images to see how simular they are. A typical example is to compare a celebrity picture with a picture of oneself and decide how "similar" in % the images are. One could also use feature extraction to search for an item, i.e., find Waldo in an image.

Features are classified into two types:

- Global - these are a function of the whole image.
- Local - these have a position and are a function of a local image region.

2.12.1 Semantic gap

How do one make a computer understand the same as a human perceives when looking at the same image. The computer represents the image as a matrix with numbers while humans look at images at another level. A computer will have a problem with understanding the difference between a dog in an image and a cat. One need to represent the information in the image in a clever way to be able to make computers understand what it is looking at (*Semantic gap - Wikipedia, the free encyclopedia*, 2016).

2.12.2 Feature Vector

On can quantify information in the image as vectors, i.e., [0.34, 0.44, 0.08 ...] and one can extract the following features:

- Describe the shape
- Describe the color
- Describe the texture

These features are just lists of numbers used to quantify the image, but if we use them together with machine learning, we can derive meaning or classification.

Alternatively, a combination of two or three features can be used. An example is presented in the figure below.

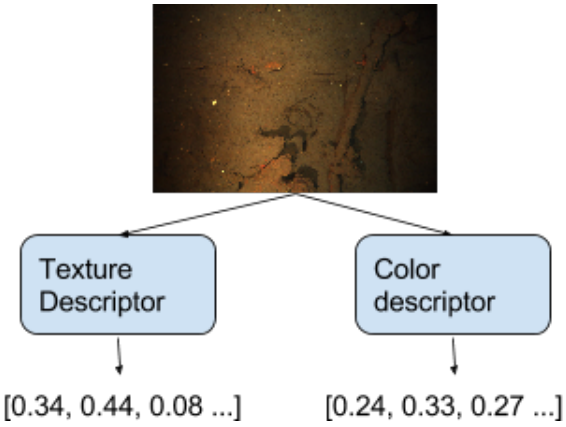


Figure 2.30: texture and color descriptor

Here in figure 2.30 one can see a picture of the sea bottom in the Trondheimsfjord; one could represent the sand bottom with a color descriptor and a texture descriptor.

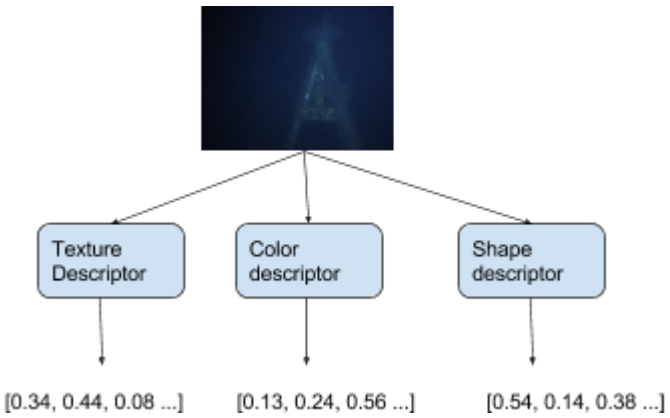


Figure 2.31: texture, color and shape descriptor

In this figure 2.31, the transponder illustrates that one could extract the features shape, color, and texture to be able to represent the transponder tower.

2.12.3 Image descriptor

To be able to extract the feature vector, one need to apply an "image descriptor", it is a form of image preprocessing to be able to quantify the content of an image numerically. An example of such an "image descriptor" is to calculate the

histogram of the image, the histogram returned is the "feature vector". Some examples of image descriptors are:

- Local Binary Patterns
- Histogram
- Histogram of colors
- Edges

2.12.4 Feature Descriptor

It takes an image and returns multiple feature vectors by looking at different regions in the image. This method tries to find the areas of the picture that it considers "special." One can find these "special" areas using a "keypoint descriptor."

Typical "keypoint descriptors":

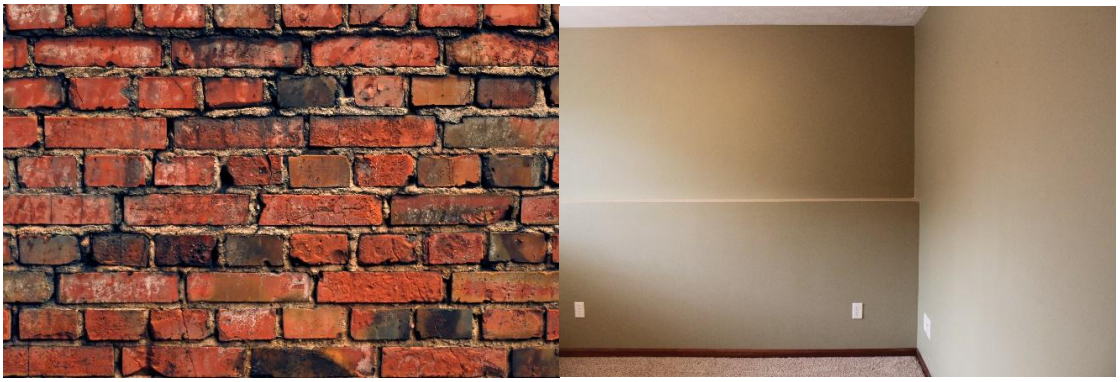
- (SIFT)
- (SURF)
- (BRISK)

When these "special" regions of the image are detected, the method will send in these areas almost as separate images to the "image" descriptor to return separate "feature vectors."

Please note, image descriptors are used to quantify an image globally, while feature descriptors are used to quantify an image locally

2.13 Texture Descriptors

Texture descriptor is a type feature vector and will be explained with an example below.



(a) *Rough texture* (b) *Smooth texture*

Figure 2.32: *Smooth texture*

Observe in figure 2.32, there is a brick wall in 2.32a and a smooth wall in 2.32b. When one compares the two, one can see that there is a lot more "detail" to the brick wall. The reason we perceive it this way is since there is a larger difference in pixel intensities, and also a higher distance between different areas of similar intensities. On the other side the smooth surface has petite differences between its intensities, and if there are any small change in the "texture", it is between short distances in the intensities. In the computer vision world, these intensities are described in brightness values, often quantified in a gray-scale image as gray levels.

Textures in images quantify:

- Grey level differences (contrast)
- Directionality or lack of it
- Defined size of area where change occurs (window)

2.13.1 Local Binary Patterns

Local Binary Patterns are the basis of the LBP method presented in section 3.2 and the background theory is therefore presented here.

Local Binary Patterns is a texture descriptor. It has even proven to be useful in face recognition as in (Ahonen, Hadid, & Pietikäinen, 2004) and also proven to work well as a texture-based method for modeling the background and detecting moving objects in (Heikkilä & Pietikäinen, 2006).

The interested reader is recommended to read the slides in (Pietikäinen & Heikkilä,

2011), as they give an in depth explanation of the LBPs theory and application areas.

Local Binary Patterns compares the pixels around its center using a kernel that evaluates its neighborhood. The name is built up of Local meaning the size of the matrix it looks at. Binary pattern comes from that the threshold of the "local" area makes the pixels become either "1" or "0" and the process it uses to calculate the value that will be further explained below.

Computing the Local Binary Pattern

In table 2.3, the general steps of calculating the LBP is shown. It is important to note step 3, that the weights are calculated differently for different kinds of pattern one choose to use, this will be explained later in 2.13.1.

Table 2.3: Local Binary Pattern descriptor steps

LBP descriptor workflow
1 - Look at center pixel in a neighborhood
2 - Threshold the values in the neighborhood based on the center pixel
3 - Calculate the weights of the threshold values (based on the type of pattern used)
4 - Sum the weights
5 - Set this value as the new centerpixel value in the LBP image

To illustrate the points in the table, a figure and an example is given below.

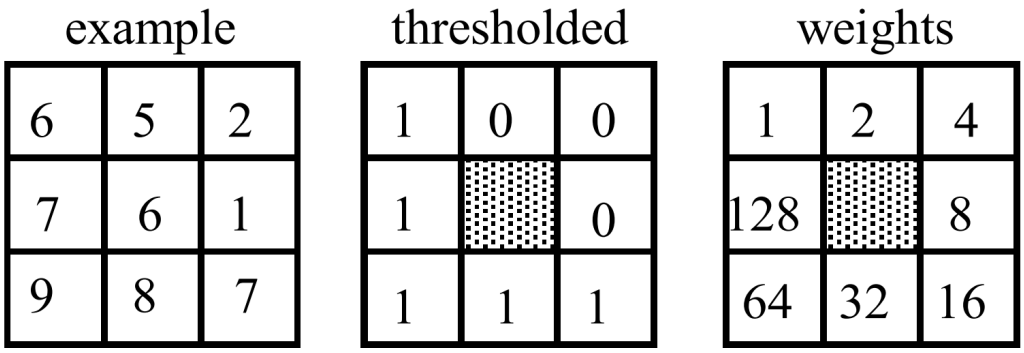


Figure 2.33: LBP calculation example. Courtesy of (Pietikäinen & Heikkilä, 2011)

In figure 2.33 an example of a 3x3 matrix to compute the LBP is shown.

The first step is to look at the center pixel.

The second step is to see which surrounding pixel values are equal or higher then the center pixel (they are set to "1"). If the values are lower they are set to 0.

The third step is to calculate the weights. In this example there are 8 surrounding pixels the first one is $2^0 = 1$, second $2^1 = 2$, third $2^2 = 4$, fourth $2^3 = 8$, fifth $2^4 = 16$ and so on.

The fourth step is then to sum the weights that the threshold gave the value "1", so in this example we get:

$$2^0 + 2^4 + 2^5 + 2^6 + 2^7 = 1 + 16 + 32 + 64 + 128 = 241$$

The calculated value will be placed in the output LBP image with the same height and width as the input image(the image that LBP is calculated on). Please note how computationally simple this LBP is and since it only compares the center pixels with the surrounding pixels it is also invariant to any monotonic gray level change (Pietikäinen & Heikkilä, 2011).

Only looking at the 3x3 matrix was proposed by the original LBP paper (Ojala, Pietikäinen, & Mäenpää, 2002) and has later been extended to be able to have other sizes of the neighborhoods. The advantage of such a small neighborhood is that the calculation goes faster than a bigger area. If the area is small the LBP is able to capture small details in the image, than if it uses a bigger neighborhood.

In the extension of the original method, the method handles varying neighborhood sizes. The new method introduces a number of points(pixel intensities) "p" and instead of looking at a square matrix, it defines a circular neighborhood given a radius "r". The radius r can change with different scales.

It is important to note that if one increase "r" without increasing "p", there are very few points for a big neighborhood and hence it loses the local discriminative power.

Different threshold patterns and their meaning

Since it is possible to set the radius r and the points p, it gives the LBPs the possibility to tune them to get the best possible results. It should be noted that the greater they are, the bigger neighbourhood, and therefore the computational cost will increase. In the new improved method the neighbourhoods different kinds of "patterns" are called "prototypes". The number of possible "prototypes" is is defines by "r" and "p".

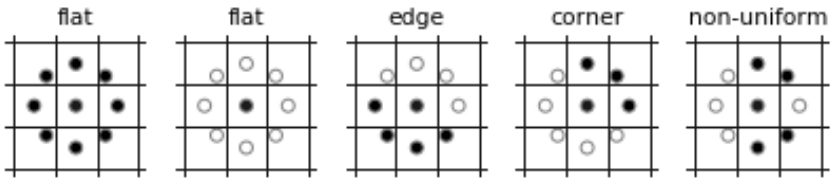


Figure 2.34: flat, edge, corner and non-uniform

In figure 2.34 the black dots are 0 and white dots are 1. This is to illustrate what gets classified as flat, edge, corner and non-uniform patterns. The flat,edge and corner information has been used to evaluate the training data in sub-subsection 3.2.1 and the interested reader is recommended to look at how the figures highlight the flat, edge and corner areas of the image.

Uniform pattern

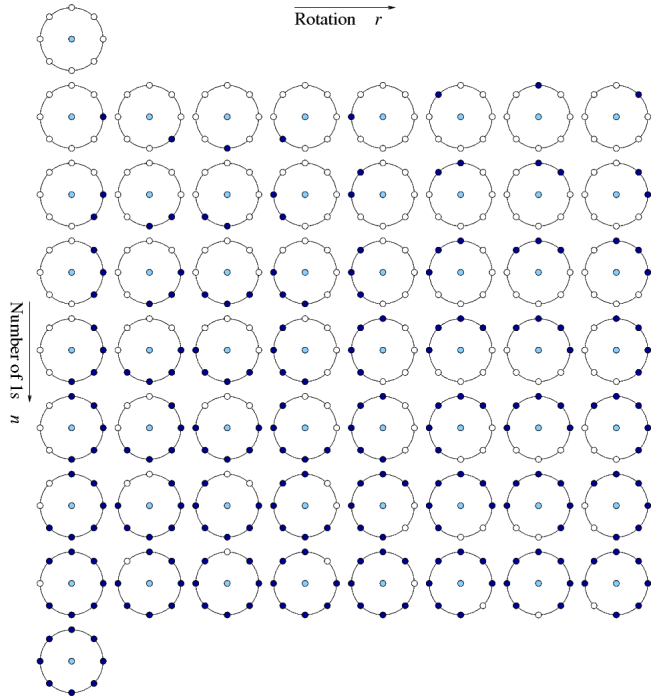


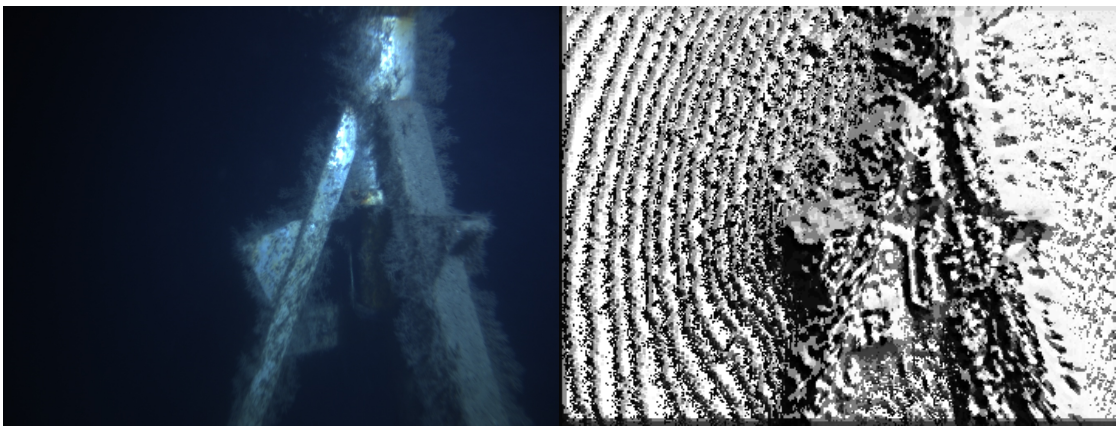
Figure 2.35: 58 prototypes when $p = 8$. Courtesy of (Pietikäinen & Heikkilä, 2011)

In figure 2.35, there are 58 rotation invariant binary patterns. The 9 patterns in

column 1, is an important subset of these, representing the **uniform patterns**. These are the ones used in the uniform LBP and are used for the LBP method developed in this thesis. The uniform patterns are the most robust and are characterized by that have less than three 1-0 or 0-1 changes in their pattern. Rotation invariant means that any arbitrary rotation of the image will not change the feature vector created from the neighborhood.

As in the example figure above, the value of p defines the amount of uniform prototypes. The $numberofuniformbinarypatterns = p + 1$. The feature vector created by the LBP also counts the number of prototypes that is not uniform, these are the last entry in the histogram by the uniform descriptor. The number of patterns can change by the dimensionality, and to exactly now the number of features in the vector one should look into the paper (Ojala et al., 2002).

Example image



(a) *Original image* (b) *Default" LBP image*

Figure 2.36: *From color image to gray scale resized image*

The figure 2.36b was made by calculating the "default" local binary pattern of image 2.36a. Note that the method enhances edges. The "default" LBP calculated the flat, edge, corner and non-uniform features(as seen in figure 2.34) of the image.

Summery of Local Binary Pattern descriptor

The LBP contains many different patterns and there are a lot of ways to tune the descriptor. Because of this one needs to take into account that for some parameters of "r" and "p" values one could get a large feature vector that will make predictions slower than another set of these values. On the other hand, if the feature vector is short it will be quite fast. The extension of the orginal LBP, especially the uniform method is rotationally invariant and this method is available in scikit-image.

2.13.2 Haralick descriptor

Haralick texture features are used to quantify the texture of an image. Haralick texture features are great at distinguishing between smooth and rough texture as in figure 2.32.

The Haralick texture descriptor has proven to work well in distinguishing and classifying six different types of pore structure in reservoir rocks on a microphotographic level in (R. Haralick & Shanmugam, 1973).

There is an additional benefits of using the Haralick descriptor in the obstacle avoidance method. Since it is a texture descriptor one could further build upon the algorithm to classify other textures that the ROV typically encounters, such as sand, reef, old fish nets and so on.

Steps in the Mahotas implemented method

The table 2.4 below displays the step done inside the Harlick descriptor used in this thesis provided by the Mahotas library.

Table 2.4: Haralick steps

Haralick workflow
1 - Image converted to grayscale
2 - Compute the GLCM in 4 different directions
3 - Compute the Haralick descriptor of each of the 4 GLCM
4 - Take the average of the 4 Haralick descriptor

Gray Level Co-Occurrence Matrix

A GLCM is a histogram of a gray scale image where the method counts the co-occurring values(pixel intensity) at a given offset over an image. Co-occurring is the same as horizontally adjacent, defined as " to the left of a pixel".

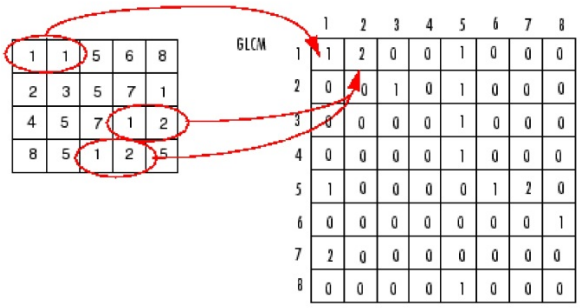


Figure 2.37: Image to describe the process of calculating the gray co-occurrence matrix. Courtesy of (*Create gray-level co-occurrence matrix from image - MATLAB graycomatrix - MathWorks Nordic*, n.d.)

In figure 2.37, the left matrix is the gray scale image and the right matrix is the product after calculating the GLCM. There are three co-occurring values in the figure. The first is [1,1] and since it only occurs one time in the gray scale image "1" is placed at position (1,1) in the GLCM. The two other co-occuring values are both [1,2]. Since they are two co-occuring values, the position (1,2) gets the value 2 in the GLCM.

In figure 2.37, we defined co-adjecent as being "to the left of a pixel". However, we could define co-adjacent in different ways. We could define it as:

- "to the right of a pixel."
- "upwards of a pixel."
- "downwards of a pixel."

This gives a total of four directions. Computing the GLCM using all four directions gives four GLCM.

Haralick features

The four GLCM will then be used to compute four sets of Haralick features.

In table 2.5 the Haralick features that compute the statistics from the GLCM

are shown. The table lists the features as in the original paper by Haralick (R. M. Haralick et al., 1973).

Table 2.5: Haralick features

Haralick features
1 - angular second moment
2 - contrast
3 - correlation
4 - sum of squares: variance
5 - inverse difference moment
6 - sum average
7 - sum variance
8 - sum entropy
9 - entropy
10 - difference variance
11 - difference entropy
12,13 - information measures of correlation
14 - maximal correlation coefficient

Interested reader are recommend to read more about them in (R. M. Haralick et al., 1973) and (Miyamoto & Jr., 2011)

According to (Coelho, 2013b) their library only use the 13 first features as they argue that the 14th feature is considered unstable.

Improve accuracy and Achieve Some Rotational Invariance

After the four GLCMs have been turned into four sets of Haralick features, we average all the features so that we end up with one feature with 13 dimensionality. To average all of them is good since it improves accuracy and it improves it rotational invariance (the average of four directions).

Summary of Haralick Descriptor

Haralick Texture Feature is a global method that looks at a whole gray scale image computes the image‘s GLCM and then the Haralick features. It’s therefore different from the LBP method, which only looks at the value of the image matrix given a certain radius and number of points.

The Haralick feature is easy to use, since there are no parameters to tune, since it looks at the whole image it makes into it‘s GLCM.

Even though the method achieves some rotational invariance when it is computed in four directions and averaged, it is not at the same level of rotational invariance as the LBP method.

Since the operations are done on a gray scale image and looks at it‘s adjacent pixels it is quite sensitive to noise, only small areas of noise in the image will change the construction of the GLCM and therefore the resulting Haralick feature vector as well.

2.14 Classification

Classification is simply to put something into a category. In computer vision, classification would be to assign a label to an image from a pre-defined set of categories or classes.

For instance, one task might be to determine if a ball is a basketball. To do classification, it needs to be able to find what distinguishes basketballs from other kinds of balls. The most distinct features of a basketball might be its color, shape, volume and texture. These features are more distinct than let's say, whether the ball is round or bounces. So if one where to classify a ball like a basketball, one might first look at the volume of the ball, to filter out all small balls, i.e., might be used for a tennis-ball. Then one can look at the texture to determine that it has the typical basketball surface.

2.14.1 Supervised Learning

We already know what a basketball is, as well as its unique features. Hence, we train a classifier to recognize the features of a basketball. The feature data “X” is labeled with basketball and sent into the classification algorithm. This process of labeling the known input is called supervised learning.

2.14.2 Training data

Please note that there exist a variety of different kinds of basketballs in the world, and when we train our classifier on only a few basketballs, this set of basketballs is called the training data. It is important to note that one should not test the

model on its training set, but instead on new “unused” basketballs. The function made from the training data is referred to as a model.

Keep two data set separate

To do regular classification with machine learning, one needs at least two sets of data (which in the case of computer vision would be images).

Table 2.6: Datasets

Type of data sets in machine learning
Data-set 1 : Training data set
Data-set 2 : Testing data set

In table 2.6, it is shown that we divide the dataset into a Training and testing data set. The training data set is used to train the machine learning algorithm while the testing data set is used to train the model.

We need to keep the dataset separate, since if we train the model on the same data set we are testing it on would be ”cheating”. Only when we run the algorithm on data the model has never seen before, can we tell how good the classifier model is.

2.14.3 Train the classifier

To train the classifier, a function “f” is used to extract the characteristics of a sample of the data “X” and use that as input into a classifier (like a linear support vector classification) and out comes the class.

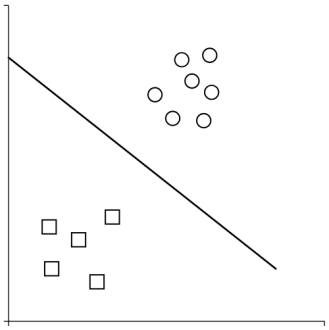


Figure 2.38: An example data-set

The sample "X" is represented as a vector. If we were trying to determine if a ball is a basketball, each value would be feature of the basketball. Based on these features, the function "f" creates the class. In figure 2.38 two different classes are shown. Let's say that the basketballs are the circles in the figure. When function "f" calculates these features on an unknown sample, it gets a probability of what class the sample belongs to. It does this by comparing the features in its vector to the features of the samples that has already been classified during the supervised learning stage. The algorithm compares the texture of the new unknown ball with the texture of the balls it knows are basketballs. Let's say they are 99% similar, then the unknown ball would get classified as a basketball(in figure 2.38 it would be placed in the upper right hand corner).

2.15 Machine Learning

Arthur Samuel defined machine learning (ML) as a "Field of study that gives computers the ability to learn without being explicitly programmed" (*Machine learning - Wikipedia, the free encyclopedia*, 2016).

In ML a computer learns to make predictions based on a set of training data. It can then use its trained model to classify unknown data into classes or clusters

2.15.1 Binary Classification

In Binary Classification an algorithm classifies some label as one of only two classes

Common methods used for classification:

- Decision trees
- Random forests
- Bayesian networks
- Support vector machines
- Neural networks
- Logistic regression

The support vector machine is widely used. It is used in this work and will therefore be explained further.

Support Vector Machines

Support Vector Machines (SVM) is explained in detail in (*Support vector machine - Wikipedia, the free encyclopedia*, 2016). The main concepts will be introduced in this section. The interested reader is recommended to read the paper (Joachims, 1998) to see examples of how on can use it in text classification.
A Tutorial on Support Vector Machines for Pattern Recognition is given by (Burges, 1998). The technology was first intended for text recognition, but was later adopted to other domains, such as computer vision.

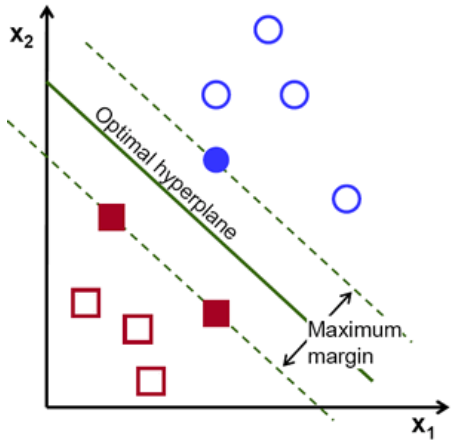


Figure 2.39: Optimal hyperplane example. Courtesy of (*Introduction to Support Vector Machines — OpenCV 2.4.13.0 documentation*, 2016)

In figure 2.39, to classes are illustrated in red and blue. The classes are divided by the optimal hyperplane. The optimal hyperplane is defined as the plane that separates the two classes with maximum margin(see the arrows normal to the plane in figure 2.39)

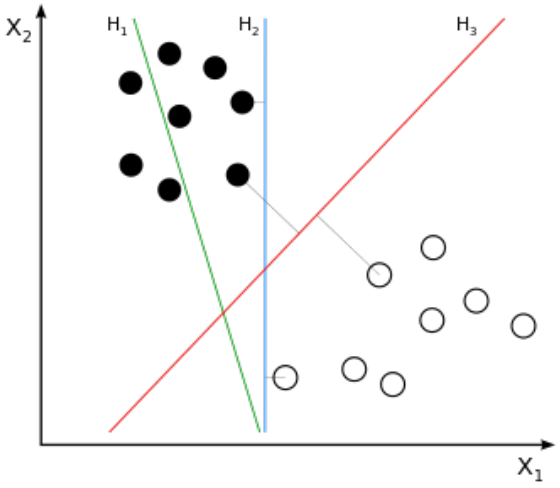


Figure 2.40: Optimal hyperplane example. Courtesy of (*Support vector machine - Wikipedia, the free encyclopedia*, 2016)

To understand what means by the optimal hyperplane, another example is shown in figure 2.40, where the classes are divided into black dots and white dots. There are three separting line shown in the figure: H1,H2 and H3.

- H1 does not separate the classes
- H2 separates the classes with only with a tiny margin.
- H3 separates the classes them with the maximum margin and therefore H3 is the optimal hyperplane.

As one can see from figure 2.39, there are two red squares and one blue circle on each respective maximum margin distance to the optimal hyperplane (these points are called the "support vectors").As one can see, the optimal hyperplane is only defined by those points closest to it. It is important to note that this is a big uncertainty in SVMs. Because the optimal hyperplane is only defined by the points closest to it, one bad data point can result in an incorrect optimal hyperplane. This can then lead to bad classification results.

Multiclass Classification – One vs All

To classify instances into more than two classes is called multiclass classification. Multiclass classification can be thought of as comparing one class with the rest of the "n" classes, often called the "one-vs-rest" method. Please note that there is another method called one-vs-one, but we will focus on the one-vs-rest.

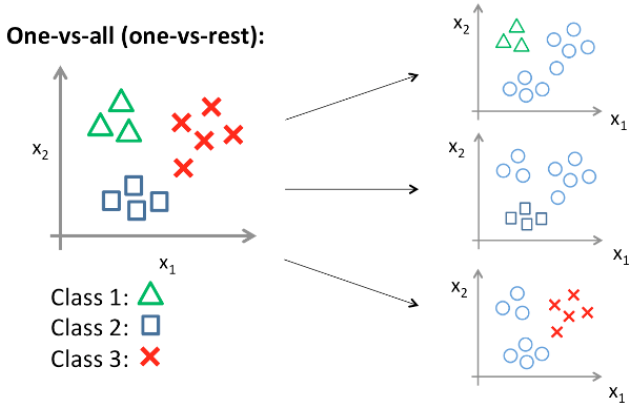


Figure 2.41: One vs all. Courtesy of (Ng, n.d.)

In figure 2.41, an illustration of how the one-vs-all algorithm compares each class against the rest is shown. The algorithm actually divides the problem into several binary classification problems. Occasionally a sample will get classified as several classes. In this case the sample will be classified as the class it has the highest probability of being. The goal of the SVM algorithm is to maximize the margin of the optimal hyperplane between the classes. Note, this functionality is available in the Scikit-learn classification function.

Chapter 3

Obstacle Avoidance Methods

This chapter will present three different obstacle detection methods that have been developed in this thesis.

First, is the "Disparity Method", then there are two very similar methods, the "Local Binary Pattern" method and the "Haralick" method. The last two methods are called texture-based methods since they both use image descriptors to analyze the texture of sub-regions within the image. The difference between the two lies in the image descriptor used.

3.1 Disparity Method

Overview of disparity method work flow can be seen in table 3.1.

Table 3.1: Disparity method steps

Disparity workflow
1 - Load camera parameters
2 - Grayscale conversion
3 - Image pair rectification
4 - Block matching
5 - Remove error margin (from challenging calibration parameters)

It is important to note that in step 1 in 3.1, the method needs to have the camera parameters precalculated so they can be loaded directly.

3.1.1 Calculating the camera parameters

Calculating the camera parameters is a complicated task, the reader will be introduced to three different software tested in this process.

- OpenCV calibration
- Bouguet camera calibration toolbox for Matlab
- Agisoft PhotoScan calibration

Evaluation of calibration toolboxes

Camera calibration in OpenCV isn't a trivial task. In fact, the built-in methods require almost perfect conditions to work properly. Firstly, the camera resolution can restrict the rotation angle of the checkerboard as some corners might be difficult to locate while rotating the checkerboard. Secondly, images must have a constant background, with no luminosity/contrast variations. Finally, as the calibration is based on corner matching, all corners must be visible in each image, which may make the process of finding an acceptable checkerboard pose a struggle. An example image is presented in the figure 3.1 below.

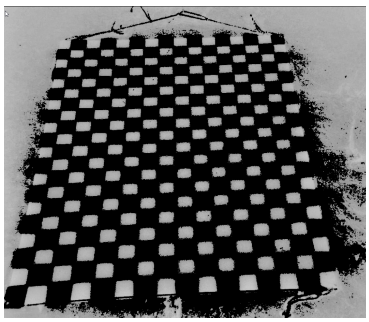


Figure 3.1

In order to make OpenCV functions find corners correctly, the background contrast to the checkerboard was manipulated, in an attempt to improve corner detection.

However, since OpenCV was unable to find the corners, we decided to use the Jean-Yves Bouguet camera calibration toolbox. Please note that the MathWorks built in calibration toolbox for Matlab was also unable to find the corners correctly. The calibration steps (for Jean-Yves Bouguet camera calibration toolbox) are well illustrated with text and illustration images in (*Camera Calibration*

Toolbox for Matlab, 2016b). The downside is that one has to click each corner of the checkerboard manually. We also performed a calibration using Agisoft Photoscan (*Agisoft PhotoScan*, 2016).

When we compare the results from rectification with the parameters from Agisoft Photoscan and the Jean-Yves Bouguet camera calibration toolbox, one clearly sees that the Jean-Yves Bouguet shown in figure 3.2 was the only one that removed distortion at an acceptable level (please compare to figure 3.3).

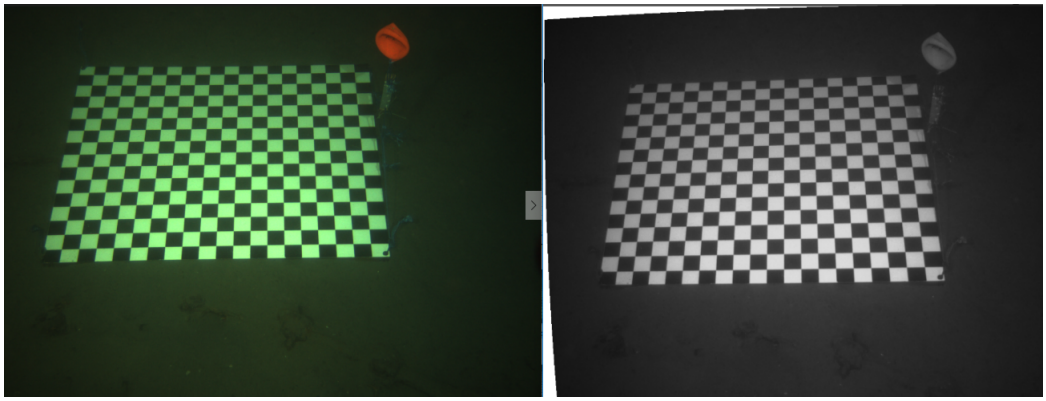


Figure 3.2: Checkerboard image before and after undistortion. The undistroted image can be seen to the right

As seen in figure 3.2, the left image is prior to rectification, while the right hand side of the image is after this process. Please note that the lines in the checkerboard is more straight than prior to rectification. Lastly, this process was done using the Jean-Yves Bouguet camera calibration toolbox calculated parameters.

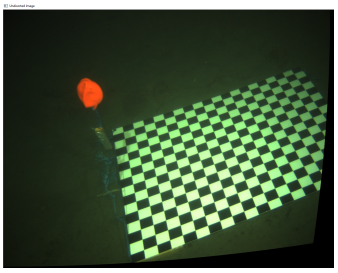


Figure 3.3: Undistortion done with the parameters calculated by Agisoft

In figure 3.3, one can see a rectification done with the parameters from Agisoft. Please note that the lines of the checkerboard are distorted (not straight) and hence unacceptable.

Camera parameters from calibration

The table below is the resulting camera parameters from the calibration with the Jean-Yves Bouguet camera calibration toolbox. Please notice that the skew and tangential distortion (p_1, p_2, p_3 and p_4 for left and right camera respectively) coefficient are 0.

Table 3.2: Camera parameters

Table 3.3	Table 3.4
Left camera:	Right camera
$fxL = 2222.72426$	$fxR = 2226.10095$
$fyL = 2190.48031$	$fyR = 2195.17250$
$k1L = 0.27724$	$k1R = 0.29407$
$k2L = 0.28163$	$k2R = 0.29892$
$k3L = -0.06867$	$k3R = -0.08315$
$k4L = 0.00358$	$k4R = -0.01218$
$k5L = 0.00000$	$k5R = 0.00000$
$cxL = 681.42537$	$cxR = 637.64260$
$cyL = -22.08306$	$cyR = -33.60849$
$skewL = 0$	$skewR = 0$
$p1L = 0$	$p1R = 0$
$p2L = 0$	$p2R = 0$
$p3L = 0$	$p3R = 0$
$p4L = 0$	$p4R = 0$

Extrinsic parameters (position of right camera with respect to left camera):

Rotation vector:

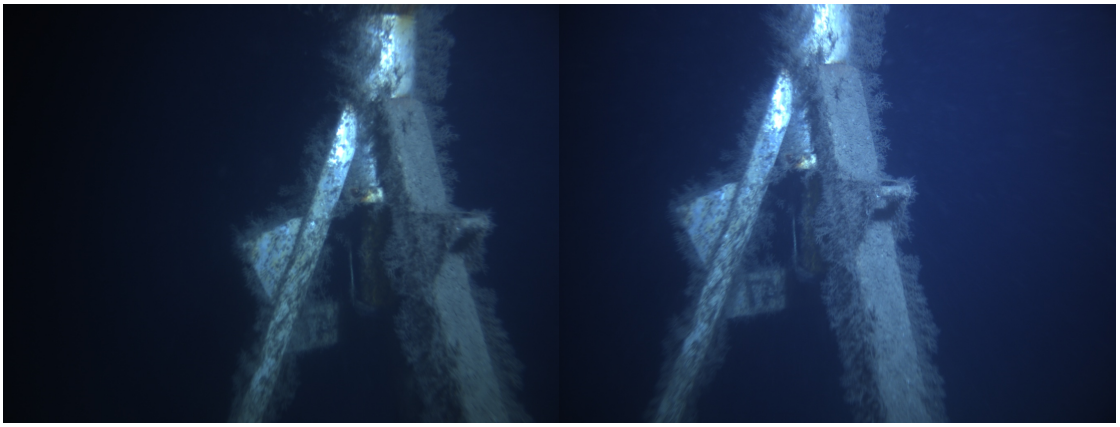
$$R = [-0.04129, 0.01292, -0.09670] + -[0.00969, 0.005520.00071]$$

Translation vector:

$$T = [303.48269, -19.19528, 29.06076] + -[2.17912, 5.0909618.22290]$$

3.1.2 Starting with an image pair

The process starts with an image pair as in figure 3.4.



(a) *Left.* (b) *Right.*

Figure 3.4: *Stereo image pair.*

The comparison of an image pair as seen in figure 3.4, is the basis for this method. Please note that they have to be captured at the same time to be an acceptable basis.

3.1.3 Load Calibration Parameters

The calibration is done with Jean-Yves Bouguet camera calibration toolbox (*Camera Calibration Toolbox for Matlab*, n.d.) and converted into OpenCV format. In order to be able to perform "Stereo Block matching", the images need to be rectified by using the camera calibration parameters.

Below is an example of how the calibration parameters from table 3.2 is converted into OpenCV acceptable format using numpy.

```
1 intrinsic_matrixL = np.matrix([[fxL, skewL, x0], [0, fyL, y0], [0, 0,
      1]])
2 intrinsic_matrixR = np.matrix([[fxR, skewR, x0], [0, fyR, y0], [0, 0,
      1]])
```

Listing 3.1: python example for how the intrinsic parameters are loaded

In the listing above, that we load the parameters for the intrinsic matrix of both the left(line 1) and right(line 2) camera

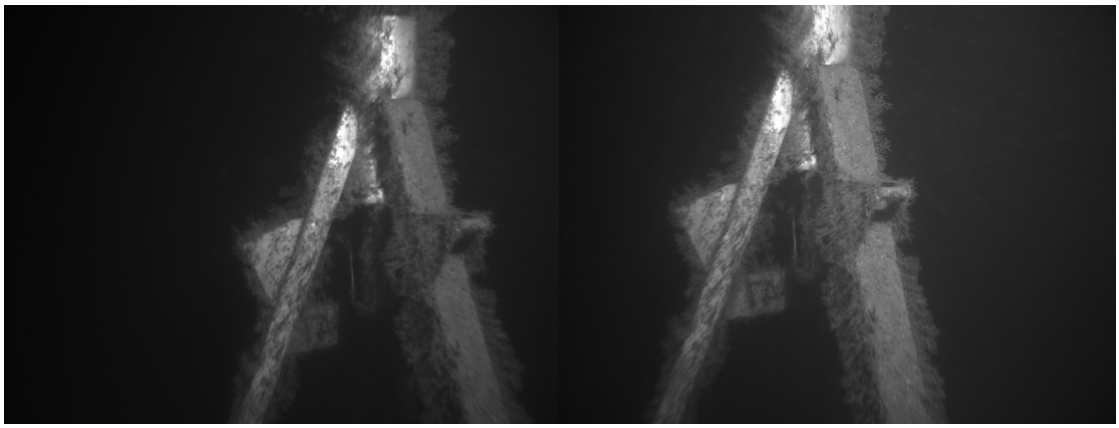
```
1 distCoeffL = np.matrix([k1L, k2L, p1L, p2L, k3L])
2 distCoeffR = np.matrix([k1R, k2R, p1R, p2R, k3R])
```

Listing 3.2: python example for how the ditortion parameters are loaded

In the listing above we load the distortion coefficient from both radial(k_1, k_2, and k_3) and tangential(p_1 and p_2) distortion for the left and right camera respectively.

Please note in the listing above that we only use the three radial distortion coefficients k_1, k_2, k_3 and hence discard k_4 and k_5 that was calculated using Jean-Yves Bouguet camera calibration toolbox. This is necessary since the OpenCV function undistort (used in section 3.1.5 for rectification) only accepts these parameters.

3.1.4 Make image pair Grayscale



(a) *Left* (b) *right*

Figure 3.5: *Gray scale image pair.*

In figure 3.5, it is demonstrated that the stereo pair in figure 3.4 is converted to grayscale color space.

3.1.5 Rectify

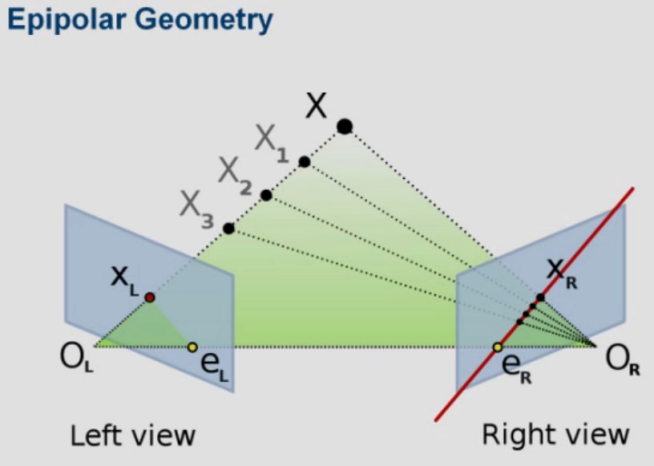
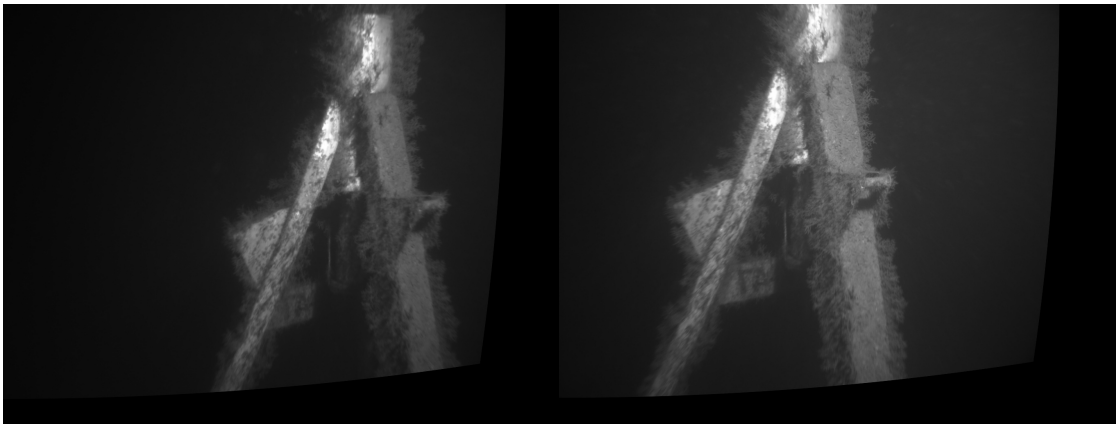


Figure 3.6: Aligning to images.

In figure 3.6, a quick overview of the importance of rectification is illustrated. Rectification of the images is the process of aligning the images. If one does not rectify the images before running the "stereo block matching" algorithm, the point X_L and X_R will not be aligned, and therefore would give untrue matches in the case where the images are distorted.



(a) *Left.* (b) *Right.*

Figure 3.7: *Undistorted image pair*

In figure 3.7, the image pair are aligned with one another after rectification is preformed.

3.1.6 Stereo block matching

Tuning the Stereo matching variables

Stereo Block matching can be tuned with four parameters, some of which are held constant during all simulations.

Stereo Block matching can be tuned by changing the parameters given in table 3.5.

Table 3.5: Stereo block matching parameters, explained in detail in the documentation found in (*Welcome to opencv documentation! — OpenCV 2.4.9.0 documentation*, n.d.).

Parameters
SADWindowSize is set to 9
preFilterType is set to 1
preFilterSize is set to 5
preFilterCap is set to 61
minDisparity is set to -39
numberOfDisparities is set to 112
textureThreshold is set to 507
uniquenessRatio is set to 0
speckleRange is set to 8
speckleWindowSize is set to 0

By tuning these parameters the program will get different results. The parameters used in the program were chosen after a tedious process of testing several parameters, and choosing the ones that gave the best result.

The parameters that gave the biggest changes when tunning was the "numberOfDisparities". the higher this value became, the less disparities it found in the image. The specleWindowSize decides how many pixels must be close to each other for the algorithm to keep them.

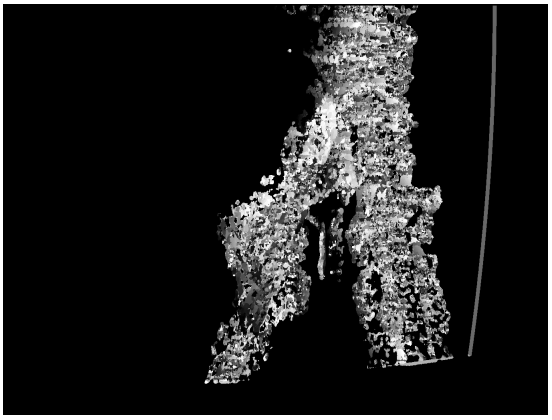


Figure 3.8: Disparity image.

In figure 3.8, a disparity image created by the disparity method is shown. Please note the curved vertical line to the right of the transponder tower in the image. The curved vertical line is an error caused from an unsatisfactory calibration, or it could be how the cameras inside the camera housing both see the same curve of the camera housing. Either way, the error is removed as described in 3.1.7.

3.1.7 Remove error margin

Figure 3.9: Disparity image without error margin.

In this figure 3.9 the error margin is removed.

```
# this part is to remove the error from the calibration
width, height = disparity_visual.shape[:2][::-1]
margin = 200
```

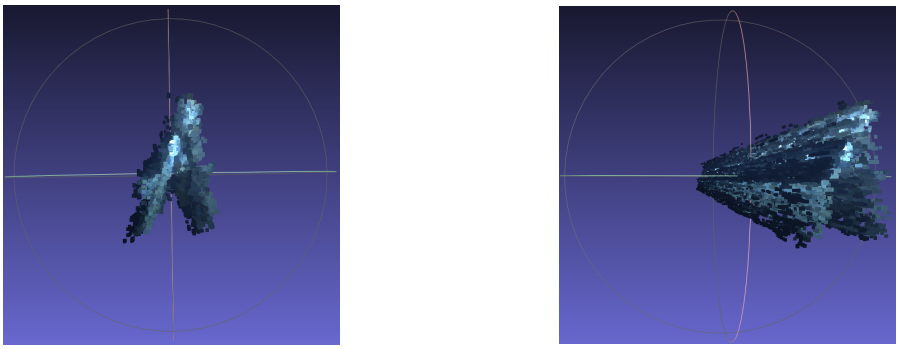
```
4
5  y1 = 0
6  y2 = height
7  x1 = margin
8  x2 = width - margin
9  # "Cropping the image"
10 disparity_visual = disparity_visual[y1:y2, x1:x2]
```

Listing 3.3: Remove error margin in Python

In the listing above, the error margin is removed by choosing a subset of the image (line 10) to avoid this error margin. On line 2 the width and height of the original image is grabbed. The $margin = 200$ is the number of pixels on the right hand side that has to be excluded. It has been found by testing several values on several pictures. The error margin is constant on all the pictures and can therefore be removed by hard-coding the appropriate margin.

Please note that the image has now a shorter width by removing a part of the image on the right hand side. This is an uncertainty and would be further discussed in section 6.1.1.

3.1.8 Create 3d point cloud



(a) *pointCloud inFront MeshLab.* (b) *pointCloud onTheSide MeshLab.*

Figure 3.10: *Point cloud from disparity.*

In order to see if it is possible to use the disparity map for depth information, a 3D point cloud is created using OpenCV, see figure 3.10. Unfortunately, the points does not give an accurate 3D figure of the transponder tower, see figure 3.10b (please note that the figure extends a lot). This means that there is too much noise in the images, or that the calibration is not accurate enough to extract this information accurately. One the other hand, the 2D image plane (i.e.,

the front) gives a good approximation to reality and can therefore be used for obstacle avoidance. Even though it is hard to extract the depth from the disparity because of the mentioned issue, we can still use the location of the obstacle to calculating the center and bounding box as will be described in section 3.4.

3.2 Local Binary Pattern Method

The goal of this method is for a given image, the method should be able to predict the segments within the image as either “ocean” or “other”(anything in the image that is not ocean.)

The Local Binary Pattern Method proposed in this thesis consist of several steps as shown in the table below.

Table 3.6: Local Binary Pattern Method steps

Local Binary Pattern Method workflow
1 - Resize image
2 - Divide Image into Blocks
3 - Extracting the segment
4 - Compute the Local Binary Pattern of each segment as a histogram
5 - Binary classification (Predict which class the histogram belongs to)
6 - Mask segment that is predicted as “other”
7 - Compute the obstacle avoidance path

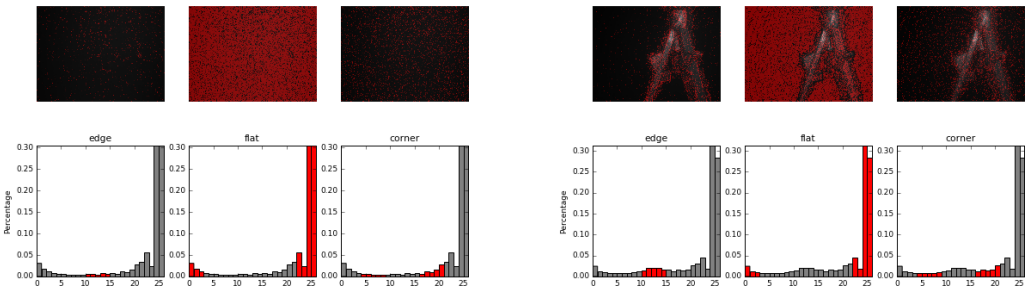
3.2.1 Training image

In table 3.6, note step 6 “Predicts which class the histogram belongs to”. To be able to predict, one need to already have trained a machine learning algorithm with a dataset as mentioned in section 2.15.

Therefore, the reader will first be presented for the process done in this thesis to find a good training image for “other” and “ocean.”

Evaluation of training images

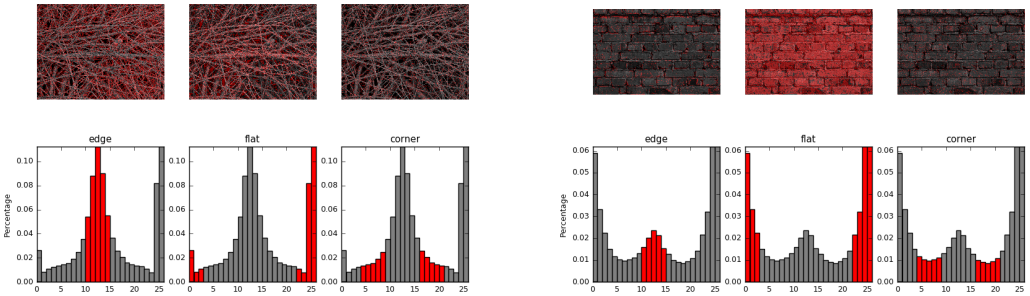
A photograph of the ocean is usually blue or greenish. But the most distinctive feature of an image of the ocean is its lack of texture. Therefore a good training image for ”other” would be one with a lot of texture and a good training image for ”ocean” would be one with the little texture.



(a) *Ocean histogram 3.15* (b) *Structure and ocean histogram*

Figure 3.11: *Histograms highlighted, created by modifying the script written in (Local Binary Pattern for texture classification — skimage v0.12dev docs, 2016)*

In figure 3.11, note how the histogram for flat is highlighted. In an image as 3.11a one can see that most of the image is full of "flat" texture. While in image 3.11b where there is an object in the image a slight decrease in the "flat" area and an increase in the "edge" area. Therefore, the "other" training image should have a lot of edges so that classification of an object is easy to distinguish.



(a) *Branches histogram* (b) *Brick histogram histogram*

Figure 3.12: *Histograms highlighted, created by modifying the script written in (Local Binary Pattern for texture classification — skimage v0.12dev docs, 2016)*

In 3.12 an example of evaluated training image for "other" is displayed. From 3.11 one can see a small but notable spike in the edge part of the histogram. Therefore, the training image for "other" should have a spike in this area, but

not as big as seen in the branches histogram displayed in 3.12a. On the other hand 3.12b gives a smaller spike in the edge area, and is therefore a better match.

Both the images in 3.12 are quite homogeneous, and lack different kinds of textures. Since we divide the image into 100 smaller segments that we analyze (in the method described in 3.2.3), it is advantageous to have an image with considerable different textures to get different kind of data points. In (Brownlee, 2013) it is shown that machine learning algorithms like linear(used in Singular Vector Machine) and logistic regression can suffer from poor performance if there are highly correlated attributes in the dataset. Therefore, the method tries to use more uncorrelated training data. The goal of this is to improve the optimal hyperplane of the SVM.

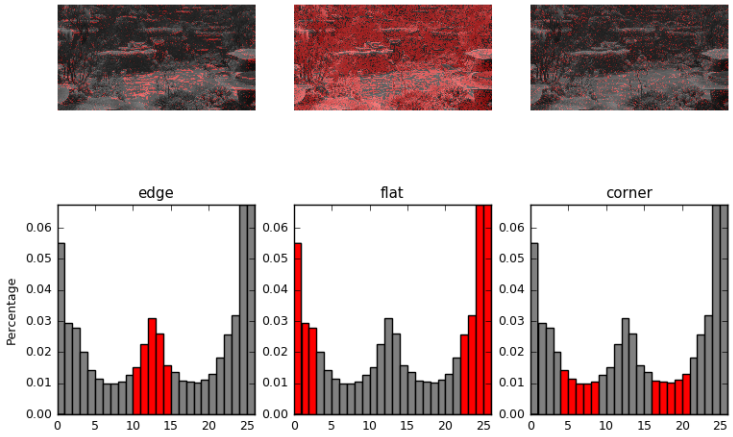


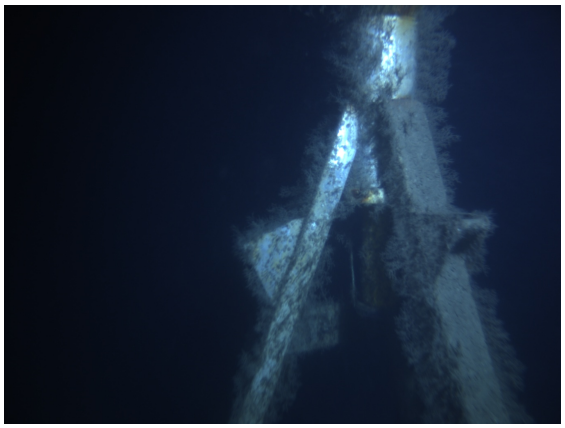
Figure 3.13: The training image used is a picture from a scene in “Finding Nemo”. Courtesy of the Disney movie “Finding Nemo”

In figure 3.13, notice how the image contains more kinds of features. This image gives better training data for the ”other” classification.

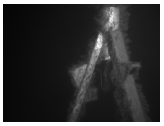
In the chapter 5, we can see the performance of using such an image. Please note, that the training image for “ocean” and “other” are each made into 100 smaller images for training the classification model.

3.2.2 Resize image

Since the image captured by the camera is 1360 × 1024 it is advantageous to speed up the predictions by sub-sampling the image to a lower resolution(it is changed to 360 × 270.). It will run faster since it will have to compute far less LBP than if the image was its original size.



(a) *Original image*



(b) *Grayscale resized image*

Figure 3.14: *From color image to grayscale resized image*

In figure 3.14, it is shown how a large image in 3.14a is turned into a smaller grayscale image as in figure 3.14b.

3.2.3 Divide Image into Blocks

The LBPs are able to encode spatial information when we calculate the LBP on blocks of the image.

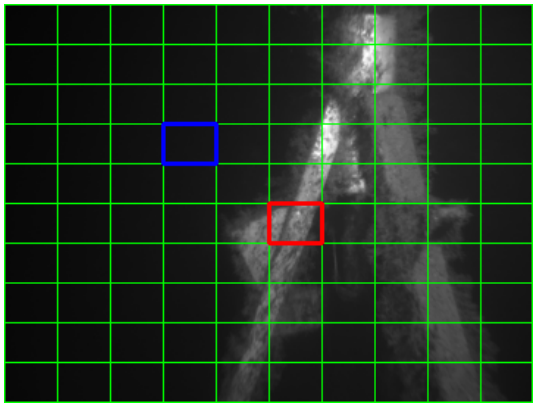
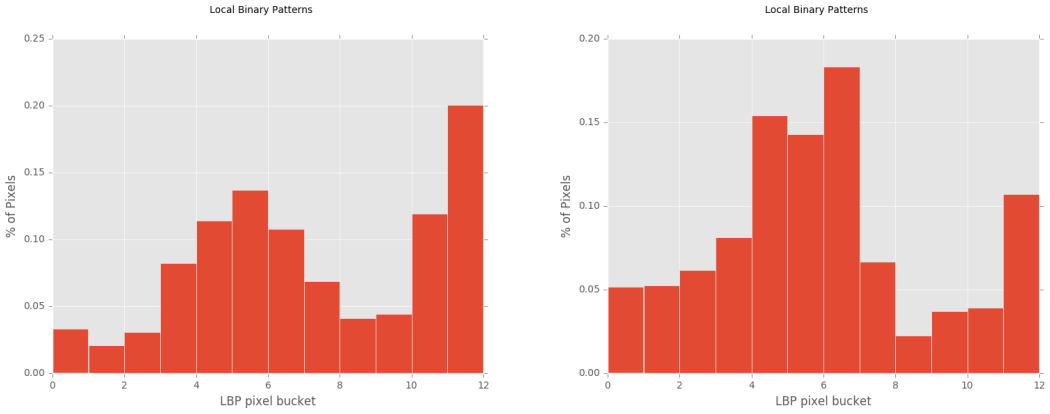


Figure 3.15: Image to describe the process of extracting the segment to compute the histogram

In the segment method, the program divides the re-sized image into 100 squares as seen in figure 3.15 and analyze the histogram computed from the extracted uniform local binary pattern features.

Extracting the segment and compute the uniform Local Binary Pattern as a histogram



(a) *Histogram of "ocean" (the blue square in figure 3.15)*

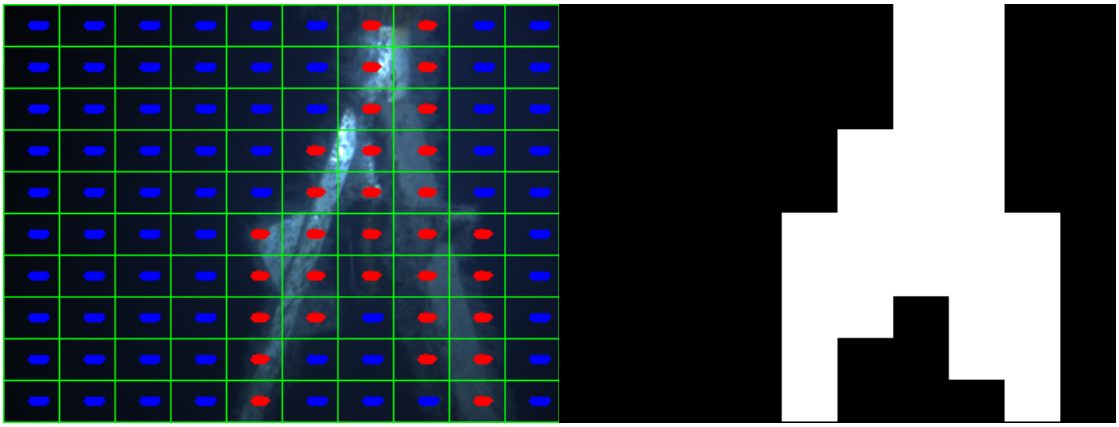
(b) *Histogram of "other" (the red square in 3.15)*

Figure 3.16: *Histograms created of chosen segment using uniform LBP*

In figure 3.16, there are two histograms created from two different Regions of Interest (ROI) of the same image. These various histograms are then fitted us-

ing the linear singular vector machine classifier that decides if they are on the "ocean" or "other" side of the fitted line between the two classes.

3.2.4 Predict each segment and mask region based on prediction



(a) *Prediction displayed* (b) *Masking image based on prediction*

Figure 3.17: *Prediction of obstacle using the LBP method*

In figure 3.17, notice the quality of the prediction with the LBP method. In image 3.17a the squares classified as "ocean" have blue dots, and those classified as "other" have red dots. The green outline the boxes where the prediction algorithm is run. There are 100 such boxes in the image.

In image 3.17b the masking procedure of the prediction is shown as a binary image(only consist of black and white). This image is then sent to the process described further in section 3.4.

.

3.3 Haralick Method

The Haralick Method proposed in this thesis consist of several steps as shown in the table below.

Table 3.7: Haralick Method steps

Haralick workflow
1 - Resize image
2 - Divide Image into Blocks
3 - Extracting the segment
4 - Compute the Haralick descriptor of each segment as a histogram
5 - Predict which class the histogram is
6 - Mask segment that is predicted as "other"
7 - Compute the obstacle avoidance path

The only difference between the table 3.7 and table 3.6 in the LBP section, is step 4. Step 4 differs, and instead of computing the LBP of each segment as a histogram, we calculate the Haralick descriptor of each segment as a histogram.

Therefore, this section will be presented with step 4, 5 and 6 from table 3.7. Step 5 and 6 is presented so that the reader can observe the result of a prediction with this method.

Please note that the training image used is the same as decided in section 3.2.1. It is chosen since it gave good results for the LBP method and since the segments contain a lot of texture.

3.3.1 Divide Image into Blocks

The Haralick method is able to encode spatial information when we calculate the Haralick descriptor on blocks of the image.

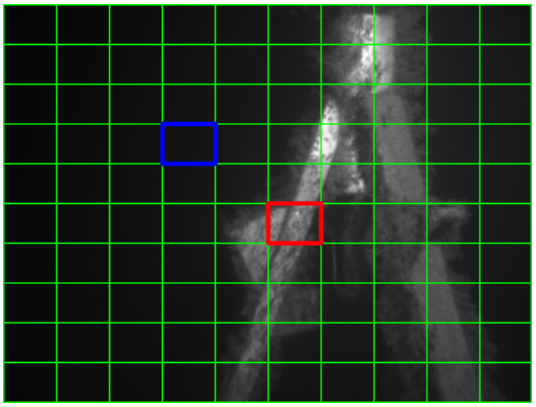
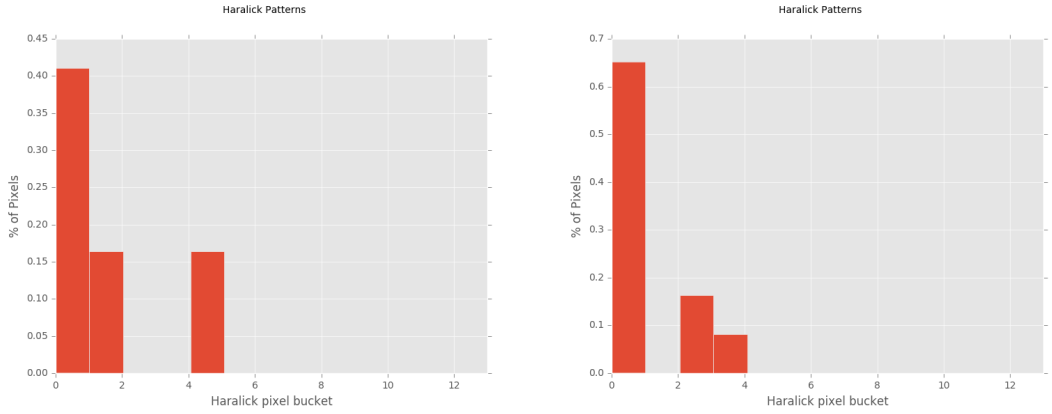


Figure 3.18: Image to describe the process of extracting the segment to compute the histogram

In the segment method, the program divided the re-sized image into 100 squares as seen in figure 3.18 and analyzes the histogram computed from the extracted local binary pattern features.

Extracting the segment and compute the Haralick descriptor as a histogram



(a) *Histogram of "ocean" (the blue square in 3.18)*

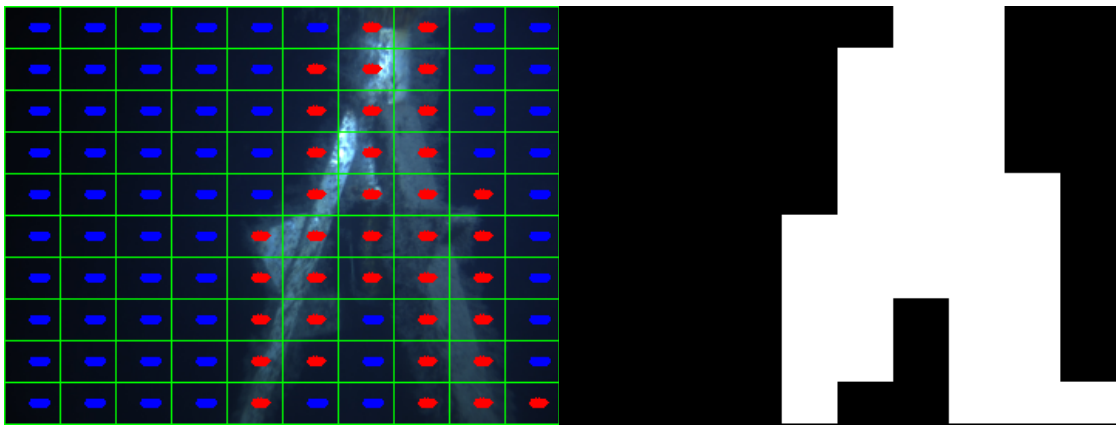
(b) *Histogram of "other" (the red sqaure in 3.18)*

Figure 3.19: *Histograms from Haralick descriptor*

As seen in figure 3.19 one can see two histograms created from two different segments of the same image. These different histograms are then fitted using the

linear Singular vector machine classifier that decides if they are on the "ocean" or "other" side of the fitted line between the two classes.

3.3.2 Predict each segment and mask region based on prediction



(a) *Prediction.* (b) *Masked image.*

Figure 3.20: *prediction of obstacle using the Haralick method .*

The quality of the prediction with the Haralick method is shown in figure 3.20. In image 3.20a the squares classified as "ocean" have blue dots, and those classified as "other" have red dots. The green outline the boxes where the prediction algorithm is run. There are 100 such boxes in the image.

In image 3.20b the masking procedure of the prediction is shown as a binary image(only consist of black and white). This image is then sent to the process described further in section 3.4.

3.4 Obstacle avoidance

The three methods discussed above (the “Disparity Method”,“Locally Binary Pattern” and “Haralick”) are ways to create a binary image that contains information about obstacles in the image. When we have created this binary image, we need to further extract the position of the center of the obstacle and to create a bounding box around a no-go zone.

3.4.1 Starting Point

(a) *Disparity image.*

(b) *Masked image after prediction from either LBP or Haralick.*

Figure 3.21: *The starting point of the obstacle position calculation.*

In figure 3.21, we can see the starting point for the Disparity method 3.21a, LBP and for the Haralick methods in 3.21b.

Please note that the image 3.21b has less detail compared to the image 3.21a since it is masked and also has a smaller resolution.

3.4.2 Contour extraction

The next step is to extract the contours of the binary images. This is done using OpenCV‘s built-in function to extract the external contours only, as we are not interested in the contours inside of a contour.

```
1
2 # calculate the external contour
3 (contours0, _) = cv2.findContours(image, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
4
5 # draw the external contour
6 cv2.drawContours(image, contours0, -1, (255, 255, 255), 2)
```

Listing 3.4: Calculate the external contour

The flag "$cv2.RETR\_EXTERNAL$" makes the function on line 3 only return the external contours, while the flag "$cv2.CHAIN\_APPROX\_SIMPLE$" is memory efficient since it returns a simpler representation of the contour.

(a) *External contours of disparity image.* (b) *External contours of masked from prediction of either LBP or Haralick.*

Figure 3.22: *Calculating the contours of the binary images calculation.*

In figure 3.22, please note that the contours have been calculated and drawn for both the Disparity method 3.22a, LBP and Haralick method in figure 3.22b (since both the LBP and Haralick give a similar masked image, they look very similar during this process).

3.4.3 Center of obstacle calculation

(a) *Center calculated from disparity image.* (b) *Center calculated after prediction from either LBP or Haralick.*

Figure 3.23: *Calculating the center of the binary images calculation.*

In figure 3.23, the center has been calculated for both the Disparity method 3.23a, LBP and Haralick method in figure 3.23b.

Observe the small white circle in the center of the "disparity object" in figure 3.23a. To find the center one takes the average with respect to x and y-coordinates of all the centroids in the image. Finding the average area center of the obstacles, sort of like a "mass center calculation." However, only the area displayed

in the binary image is taken into account and is calculates to an average area center.

Finding the center of the object

Since there are different sizes of contours, we need a way to weigh them, so we get the average center of all the obstacles in the image. The geometric center is given by summing up all then centroids respective x and y position and multiplying them by there respective "area".

The calculation can be seen bellow implemented in Python:

```
1  centroid_XList = []
2  centroid_YList = []
3
4  areaTot = 0
5  for (i, cnt) in enumerate(contours0):
6      area = cv2.contourArea(cnt)
7
8      # compute the moments of the contour
9      # use the moments to compute the "center of mass" of each contour
10     m = cv2.moments(cnt)
11     centroid_X = [area *int(round(m['m10']/m['m00']))]
12     centroid_y = [area *int(round(m['m01']/m['m00']))]
13
14     centroid_XList.append(centroid_X)
15     centroid_YList.append(centroid_y)
16
17     areaTot = areaTot + area
18
19 centroid_XList = np.asarray(centroid_XList)
20 centroid_YList = np.asarray(centroid_YList)
21
22 # take the average
23 centroid_XListCenters = centroid_XList/areaTot
24 centroid_YListCenters = centroid_YList/areaTot
25
26 # sum the points and cast to int so cv2.draw works
27 objectCenterX = int(np.sum(centroid_XListCenters))
28 objectCenterY = int(np.sum(centroid_YListCenters))
29
30 center = (objectCenterX, objectCenterY)
```

Listing 3.5: Calculate the center of the obstacle in Python

In the listing above, notice that the area of the obstacle represented in image 3.23 is calculated by the following formula at line 6:

$$\text{area} = \text{cv2} \cdot \text{contourArea}(\text{cnt}) \tag{3.1}$$

Further on the the moment for each countour "cnt" is calculated by:

$$m = \text{cv2} \cdot \text{moments}(\text{cnt}) \tag{3.2}$$

To calculate the moments in general, we use the theory of the Second moment of the area as described deeper in (*Second moment of area - Wikipedia, the free encyclopedia*, 2016). But to calculate the moments of the contours we use Green's Theorem as described more in-depth in (*Green's theorem - Wikipedia, the free encyclopedia*, 2016). Moreover, this theory is implemented in OpenCV -we can use the cv2 · moments() function to calculate the moment directly as shown on line 10 and in equation 3.2.

The "mass center" $(\bar{x}, \bar{y})$ of each moment scaled to the size of their area is calculated by (same as line 11,12):

$$\text{centroid}_x = [\text{area} \times \text{int}(\text{round}(m['m10']/m['m00']))] \tag{3.3}$$

$$\text{centroid}_y = [\text{area} \times \text{int}(\text{round}(m['m01']/m['m00']))] \tag{3.4}$$

Please note on line 19 and 20, the Numpy library (as np.) is utilized to be able to do operations on arrays to optimize for speed.

As in line 11 and 12, these scaled "mass centers" are then divided by the total area in line 23 and 24. Before the center of all the obstacles is calculated on line 27 and 28 by taking the sum of all these relative centers.

Then we sum up all the "scaled" x and y positions to give the real centers.

3.4.4 Create bounding box

(a) *Bounding box calculated from disparity image.*

(b) *Bounding box calculated after prediction from either LBP or Haralick.*

Figure 3.24: *Calculating the bounding box of the binary images calculation.*

In figure 3.24, the bounding box has been calculated for both the Disparity method 3.24a, LBP and Haralick method in 3.24b.

A simple approach has been used to calculate the bounding box. The method sums up the width in x-direction and height in the y-direction, as given in the Python implementation below.

```
(contours0, _) = cv2.findContours(clone.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

xLast, yLast = clone.shape[:2]
wLast = 0
hLast = 0

for c in contours0:
    # fit a bounding box to the contour
    (x, y, w, h) = cv2.boundingRect(c)

    if(xLast > x):
        xLast = x

    if(yLast > y):
        yLast = y

    #if (wLast < w):
    wLast = wLast + w

    #if (hLast < h):
    hLast = hLast + h

cv2.rectangle(clone, (xLast, yLast), (xLast + wLast, yLast + hLast),
(255, 255, 255), 2)
```

Listing 3.6: Calculate the bounding box in Python

3.4.5 Mean Value

The mean value is used for two things in this program. Firstly it is used to calculate the status as shown in the following subsection 3.4.6, secondly it is calculated and sent to the LabVIEW program so it later could be used as mentioned in further work in 7.2.

Mean value is simply calculated using OpenCV built in function as shown below:

```
meanValue = cv2.mean(image_binary)
```

Listing 3.7: Calculating mean value of a binary image

cv2.mean calculates an average (mean) of the array elements in the binary image

3.4.6 Status

If there is an obstacle greater than a certain threshold, the LabVIEW program will be notified. The "status" is calculated to decide if the obstacle is significant enough to be taken seriously to avoid detecting backscatter and other effects as obstacles.

Calculating the status

```
def isObsticleInFront(self):
    if self.meanValue > self.isObsticleInFrontTreshValue:
        return True
    else:
        return False
```

Listing 3.8: Calculating the status

In several experiments, different values for "isObsticleInFrontTreshValue" has been tested. From these experiment a value of

$$isObsticleInFrontTreshValue = 0.345$$

has proven to give good results for all three programs.

In the listing below, an example of how the status message is implemented is shown. This message is later sent to the LabVIEW program as part of message string that contains more information.

```
directionMessage = "status : , "

status = int(self.isObsticleInFront())
directionMessage = directionMessage + str(status) + " "
```

Listing 3.9: Making the status message

The status value meaning is easily summed up below:

- No obstacle in front of ROV : 0
- Obstacle in front of ROV : 1

3.5 Summery of methods

In this chapter, the methods control flow has been shown. The Disparity method is different to the texture methods that have a similar approach. In the 3.4 it is demonstrated how all the methods use the same methods for calculating information from the binary images.

Chapter 4

Testing methodology

This section is concerned with the joint work of the autonomy group consisting of the members in table 1.1.

4.1 Field Test in Trondheimsfjorden

(a) *Image of Gunnerus*

(b) *Image of ROV SUB-fighter 30k*

Figure 4.1: *Boat and ROV. Courtesy of Aur-lab*

The computer vision system was tested in April 2016 at the Trondheim Fjord onboard the boat Gunnerus (seen in figure 4.1a).

The cameras were mounted front-facing on the ROV SUB-Fighter 30k from Sperre AS, as seen in figure 4.1b. This ROV is operated from an onboard control room within a container on Gunnerus. The vehicle can be controlled in both

manual and automatic modes. On top of the container, there is a winch that holds the tether of the ROV.

The ROV operator could take over the control of the ROV at any time in case something happened. When the ROV operator takes control of the ROV, the ROV first goes into "stationkeeping" mode. In this mode, the ROV stops and maintains its position at the same location and depth.

The purpose of the field test was to maintain situations identical to real operations, where the goal was to go from launch to localizing the structure of interest (SOI). In our field test, the SOI was a transponder tower at 85 meters depth.

During the mission on Trondheimsfjorden, the ROV was run using its auto depth function, so it was constantly around 2 meters above the ocean floor.

The ROV was set to go from waypoint to waypoint. In this mode, it used auto-heading mode, in which the ROV holds its front at the correct angle.

The repository for the source code used in the field test is placed at:

https://github.com/larssbr/ROV˙objectAvoidance˙StereoVision

4.1.1 The camera software application programming interface Vimba

To make the computer communicate and receive images from and to the Allied vision camera(s), software has to be innstaled. First, install the Vimba SDK from AVT(Allied Vision Technologies). Second, install the *AVTDriverinstaller toll* to install the AVT Vimba SDK drivers on the computer. Finally, install pymba.

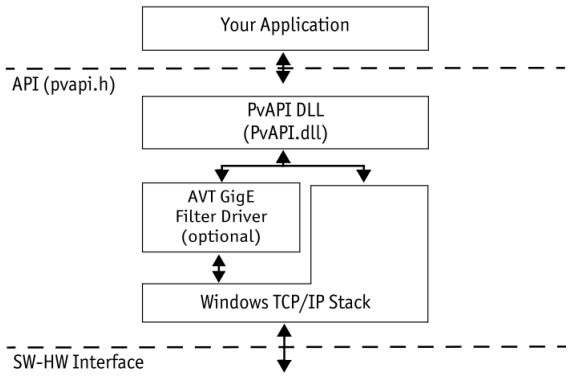


Figure 4.2: Allied Vision Technologies driver stack AVT. Courtesy of Allied Vision Technologies

In figure 4.2, an overview of the driver stack that enables the camera to connect to the software. The ".dll" files includes functions to communicate with the camera written in c and c++. Since python is the choosen language for this work, an open source package of python wrappers written for these ".dll" files called *pymba* has been utilized.

pymba is located at https://github.com/morefigs/pymba.

Pymba

Pymba is summarized in (*morefigs/pymba: Python wrapper for the Allied Vision Technologies (AVT) Vimba C API*, 2016) as: "pymba is a Python wrapper for the Allied Vision Technologies (AVT) Vimba C API. It wraps the VimbaC.dll file included in the AVT Vimba installation to provide a simple Python interface for AVT cameras. It currently supports most of the functionality provided by VimbaC.dll."

The pymba folder is then downloaded and placed in the same repository as the project. It is important to note that the user of pymba need to link the ".dll" files with pymba. An example of how the ".dll" files was linked in the vimbadll.py is seen in the listing below.

```
1
2     base = r'C:\Program     Files\Allied Vision Technologies\
    AVTVimba_1.%i\VimbaC\Bin\Win%i\VimbaC.dll'
3
4     vimbaC_path = r'C:\Program Files\Allied Vision\Vimba_2.0\VimbaC\
    Bin\Win64\VimbaC.dll'
```

Listing 4.1: change these parameters in vimbadll.py inside the pymba folder

When pymba is linked correctly, one can access the camera(s) and capture a sequence of images as seen in the listing below.

```
1  from pymba import *
2  import numpy as np
3
4      with Vimba() as vimba:
5          system = vimba.getSystem()
6          system.runFeatureCommand("GeVDiscoveryAllOnce")
7          time.sleep(0.2)
8          camera_ids = vimba.getCameraIds()
9          for cam_id in camera_ids:
10             print "Camera found: ", cam_id
11
12         c1 = vimba.getCamera(camera_ids[0])
13         c1.openCamera()
14                 try:
15             #gigE camera
```

```
16              print c1.GevSCPSPacketSize
17              print c1.StreamBytesPerSecond  # Bandwidth allocation can
    be controlled by StreamBytesPerSecond, or by register SCPD0.
18              c1.StreamBytesPerSecond = 35000000  # --> taking the half
    of MAXIMUM = 124000000, and gives a margin  # gigabyte ethernet
    cable
19          except:
20              #not a gigE camera
21              pass
22
23          #set pixel format
24          # colorFormat
25          c1.PixelFormat="BGR8Packed"  # OPENCV DEFAULT  #c0.
    PixelFormat="Mono8" #
26
27          # give the camera a short break
28          time.sleep(0.2)
29
30          #c0.ExposureTimeAbs=60000
31          c1.ExposureTimeAbs=100000
32
33          frame1 = c1.getFrame()
34          frame1.announceFrame()
35
36          c1.startCapture()
37
38          # initiate variabels
39          framecount1 = 0
40          droppedframes1 = []
41
42          c1.runFeatureCommand("AcquisitionStart")
43
44        while 1:
45          # get status of camera 1
46          try:
47              frame1.queueFrameCapture()
48              success1 = True
49          except:
50              droppedframes1.append(framecount1)
51              success1 = False
52
53
54          frame1.waitFrameCapture(100)  # 1000
55          frame_data1 = frame1.getBufferByteData()
56
57          if success1:
58              img1 = getImg(frame_data1, frame1)
59
60              # use image for method
61
```

```
62          ############ CLOSE THE CAMEREA          ############
63          c1.runFeatureCommand("AcquisitionStop")
64
65          c1.endCapture()
66          c1.revokeAllFrames()
67          c1.closeCamera()
```

Listing 4.2: pymba example for accessing the camera

```
1 def getImg(frame_data, frame):
2     img = np.ndarray(buffer=frame_data,
3                      dtype=np.uint8,
4                      shape=(frame.height,frame.width,frame.
    pixel_bytes))
5
6     return img
```

Listing 4.3: python example for getIMG function

Explanation of important lines in the code from the uppermost listing:

- line 1: import pymba
- line 4: creating the vimba object
- line 5: get the system
- line 8: get all the camera id‘s there are
- line 9,10 : is just to print out all the camera id‘s for debugging purpuses.
- line 12: creates the camera object c1 using the first identified camera from line 8
- line 13: open the camera
- line 18: the maximum streaming capabilities are set to a max of 12400000 and therefore if the program uses two cameras one need to be below this number as the sum for both cameras. We set it to 35000000 as it also gives a a safety margin.
- line 25: sets the pixel format to BGR 8 bit image
- line 31: sets the exposure time to 100000 as it gave good results
- line 33: frame1 object is made from the camera object c1
- line 42: the c1 camera object a call that it should start to aquire frames
- line 47: this is where the frame gets captured by the camera
- line 55: gets the buffer data
- line 58: img1 is the frame captured by the camera in numpy array that is compatible with OpenCV

4.1.2 An overview

The LabVIEW program contains the vehicle control system and autonomous agent program and is connected to the computer vision program through an ethernet cable that sends UDP messages.

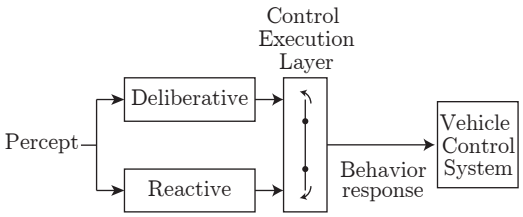


Figure 4.3: Architecture of autonomy program, courtesy of (Fossum, Trygve Olav and Ludvigsen, Martin and Nornes, Stein M and Rist-christensen, Ida and Brusletto, Lars, 2016)

In figure 4.3, an overview of the autonomy program architecture is visualized. The work in this thesis is concerned with giving a reactive signal and an exit direction to the system when an obstacle is detected.

In the field tests, the autonomy group first preformed tests to validate the *deliberative* behavior and how it compared to the (HIL) simulation to verify that the HIL simulation was accurate.

Finally, the autonomy group tested the "LabVIEW program" to see if it would behave as desired when the computer vision program sent a *reactive behavior*.

States

In Figure 4.4, the different states of the "path program" implemented by Ida Rist-Christensen as part of her project thesis is illustrated.

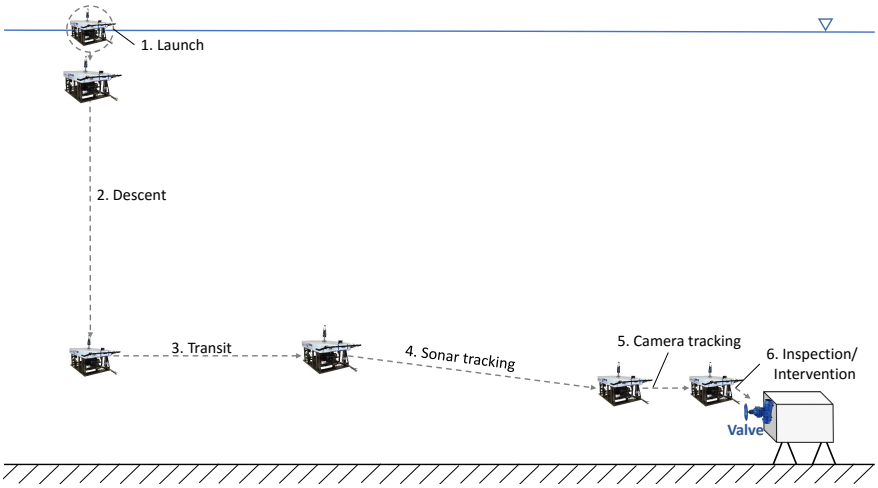


Figure 4.4: Plan of mission (Rist-christensen, 2016).

List of states seen in figure 4.4:

Table 4.1: Autonomy plan used in thesis (Rist-christensen, 2016).

Steps in plan	
1	Launch
2	Descent
3	Transit
4	Sonar detection and tracking
5	Camera detection and tracking
6	Inspection/intervention

In the experiments run in Trondheimsfjorden, the obstacle was meet in the third state "Transit". In this state, the "path program" is listening to UDP messages sent from the computer vision program. The reason it does not care in the "Descent" state is that it only moves along the z-axis and not in the x-y-plane. Therefore, the ROV only uses the DVL to detect if something is beneath the ROV.
For more information about the autonomy plan, the reader should consult Ida's thesis in (Rist-christensen, 2016).

In step 2 in table 4.1 the ROV will move vertically downwards, and it uses its DVL to measure the distance to the sea bottom, It also uses its pressure sensor to know its current depth.

Communication between the computer vision program and LabVIEW program

When the threshold for what the obstacle avoidance program interpret as an obstacle the following information would be sent with UDP over the ethernet cable.

- status
- center
- mean value

The *status* is either 0 for no obstacle or 1 for obstacle detected. The *center* is x and y position og the calculated center of the obstacle. The *mean value* is a measurement of how much big the given obstacle is.

- MESSAGE = Status + center + meanValue

The "Obstacle Avoidance" program sends the MESSAGE above to the LabVIEW: The LabVIEW utilizes the information to plan its path.

It sends the message usinng the following IP settings:

- UDP_{IP} = "192.168.1.73"
- UDP_{PORT} = 1130

The UDP_{IP} is the IP address to the computer LabVIEW program was run on.

In section 3.4 the details of how the status, center and the mean value is being calculated is presented.

4.1.3 Transponder Tower

The seafloor in Trondheimsfjorden where we conducted the experiment is mostly flat, with few obstacles. Therefore, the autonomy group decided to test the program using the transponder towers that are placed out in Trondheimsfjorden. This was thought to be a good "test" for the program. Please note, that the program is then only tested on static obstacles, and not moving obstacles.

Figure 4.5: Transponder tower ("AUR-Lab Deployment av LBL nett ved TBS", 2013)

In figure 4.5, the transponder tower is shown. That is a structure made of a tripod and an attached transponder. It is placed at a depth of 85.22 meters.

4.1.4 Organization

There was conducted several test during the day, in the table bellow a short summery of the field test and the time conducted is shown.

Table 4.2: Test cases run during the day

20 April 2016
1. 11:00-11:30 Deliberative autonomy (wrong transit coordinates) , fixed and the mission continued.
2. Deliberate autonomy - full mission
3. Test push corer
4. 17:30-18:00 Reactive: Collision Avoidance
5. Reactive: Collision Avoidance - first attempt (first attempt - missed target)
6. Reactive: Collision Avoidance - second attempt (backwards - u-turn avoidance)
7. Reactive: Collision Avoidance - third attempt (missed target)
8. Reactive: Collision Avoidance - fourth attempt (failure - should have used body fixed coordinates).

The program was tested on step 5,6,7 and 8. Unfortunately the ROV missed the obstacle two times(step 5 and 7). The Obstacle avoidance system reacted correctly both times it encountered the obstacle. The system sent messages to the LabVIEW program that the status was that there was an obstacle in front

of the ROV and the given direction the ROV should go. Unfortunately there was a bug in the LabVIEW program that was caused by using the the wrong body coordinate system that it choose the wrong latitude and longitude coordinates. Hence, this caused the new desired position to mismatch with the new given direction, this caused the ROV to set its new desired position behind itself in step 6 and behind the obstacle in step 8 (ROV was heading into the obstacle and therefore manual control was used to regain control of the vehicle.) The bug was later fixed in the LabVIEW program and worked well in subsequent simulations.

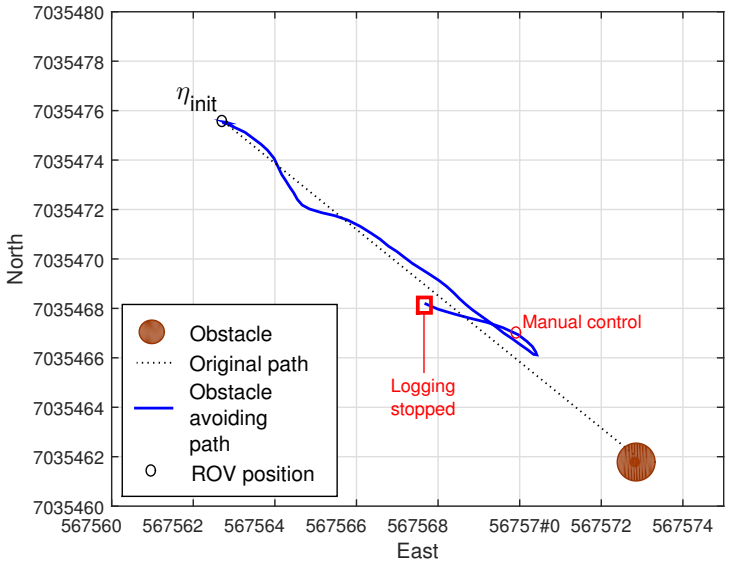


Figure 4.6: Plot of the U turn. Courtesy of (Rist-christensen, 2016)

Figure 4.6 is a plot of what happened at step 6. One can see that the ROV gets closer to the obstacle and turn away in a backwards direction. In steps it can be explained as:

- 1 follow path to goal waypoint
- 2 computer vision program changes status, telling the LabVIEW that there is an obstacle and the direction of escape
- 3 the LabVIEW program calculates new desired waypoint that effectively makes it go around obstacle
- 4 when status changes to no obstacle in front of the ROV, the LabVIEW changes desired waypoint to goal waypoint

- 5 follow path to goal waypoint

4.1.5 Obstacle avoidance path

Within the "autonomy plan", it is important for the ROV to be able to handle obstacles in its way. Ida Rist-Christensen path planning algorithm is pictured bellow in 4.7

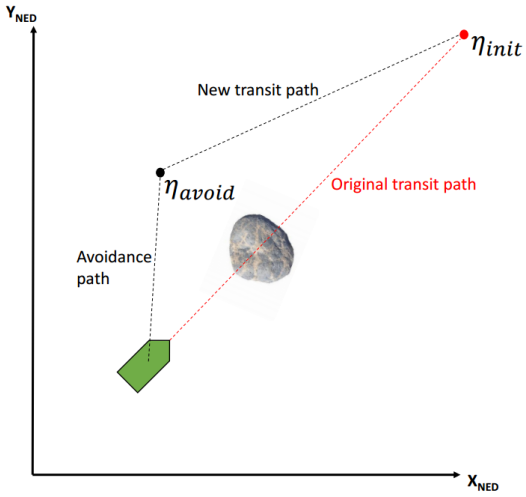


Figure 4.7: Obstacle avoidance implementation in LabVIEW. Courtecy of (Rist-christensen, 2016)

In figure 4.7, her algorithm sets a new longitude and latitude (η_{avoid} in the figure) to avoid an obstacle. This is a starting point for her algorithm, and she will develop a more sophisticated solution in her final thesis.

4.2 Simulator Environment

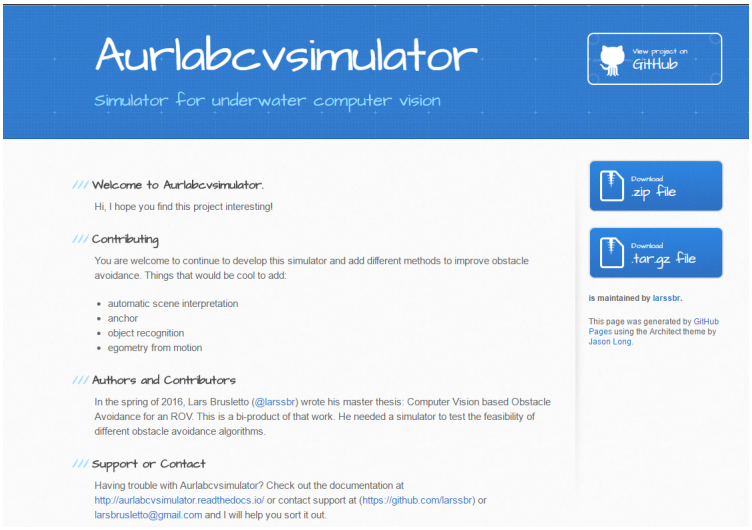


Figure 4.8: Webpage for the github project AurlabCVsimulator

In figure 4.8 one see the website for the AurlabCVsimulator located at:

http://larssbr.github.io/AURlabCVsimulator/

The repository for the source code is placed at:

https://github.com/larssbr/AURlabCVsimulator

Documentation for the project is placed at:

http://aurlabcvsimulator.readthedocs.io/

4.2.1 Simulator Overview

The name (AurlabCVsimulator) is made up of Aur-lab (Applied Underwater Robotics Laboratory at NTNU), CV (Computer Vision) and simulator. It is meant to be a starting point for designing reusable code, and as a project that contains useful folders of underwater imagery.

At this point, the AurlabCVsimulator is meant to be used for people with experience programming with Python. Inside the repository, the code is divided into object oriented classes, so it should be possible to modify and reuse the code. There is also provided "Jupiter notebooks" for fast prototyping of new methods.

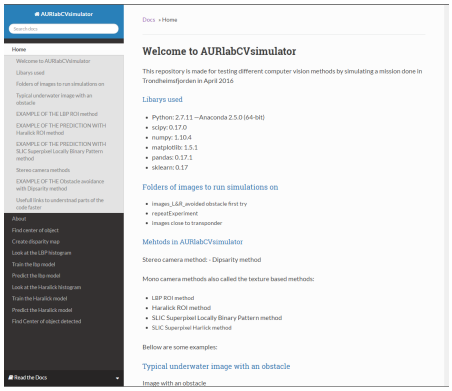


Figure 4.9: ReadTheDocs documentation for AurlabCVsimulator

In figure 4.9, the web page for the documentation of the AurlabCVsimulator is shown. Hopefully, as the code base grows, more and more documentation will be added.

In table 2.1 the packages used within the simulator is listed.

The AURlabCVsimulator has a method to load a set of stereo images. It contains the image data sets from the mission in Trondheimsfjorden. If desired, one can change the desired path to the folder of images It loads the images from a file in the repository or one can change the path to the desired folder.

It is not a simulator in the sense that one can create image sequences within the software and run it on synthetic computer-generated images.

4.3 Summary

In this chapter, the field test and simulation software has been presented. The main take aways is that the field test was run successfully, although the LabVIEW program choose the wrong desired path caused by a bug in that program. The data captured from the field test, has been reused as data for til simulator. The simulator is important since there is very few real world data sets one can try ones computer vision algorithms on. In the simulator it is possible to run sequences of stereo and mono images. The calibration parameters is already calculated and loaded into the program. This simulators goal is to reduce development time, and also save resources. In fact, it is quite expensive to go out with a ship and an ROV to test the software developed. It is advantageous to test the software in the simulator first.

Chapter 5

Results

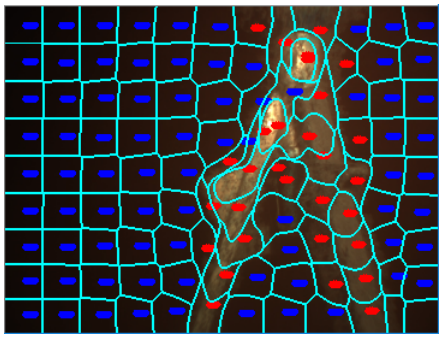
This chapter presents the results from the simulations, Trondheimsfjorden field test and other comparisons.

5.1 Comparing segmentation method with SLIC Superpixel segmentation

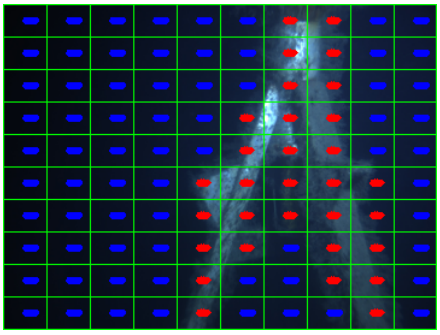
Since the SLIC Superpixel Segmentation algorithm is computation heavy, the author explored the possibility of discarding it. The results in performance and run-time can be seen below.

5.1.1 Prediction comparison for segmentation

As for comparing the performance, several experiments were investigated, and one example can be viewed in the figure bellow.



(a) *Prediction image made with SLIC superpixel segmentation and LBP descriptor*



(b) *LBP method prediction*

Figure 5.1: *Prediction image made with SLIC and LBP*

The two predictions in figure 5.1, there is very similar results, they both are able to detect the obstacle. The advantage of not using the SLIC superpixel segmentation is that it looks at the whole blocks within the image, and therefore naturally gets a "bigger" prediction, i.e it take a bit more area of the image. This is an advantage since the blocks easier gets connected and therefore the contours and center calculation has less area blocks to compute. The advantage of using the SLIC superpixel segmentation is that it forms it's segment around areas that are similar, but in the experiments this has not given any better results than the simpler segmentation.

5.1.2 Runtime comparison for segmentation

To compare the segmentation times, profiling of the code on the same method using two different segmentation algorithms are done. The profiling is visualized using the snakeViz python package, but since the figures are hard to read, tables are instead presented. Please note that the first row is the program and take 100 % of the time, the rows bellow are methods within it and are part of this time.

In the tables below, notice the high speed up when using a simpler method(not SLIC superpixel segmentation) than the one proposed in (Rodriguez-Teiles et al., 2014).

Table 5.1: SLIC Superpixel LBP method runtime

runtime
simulation_lbp.py(module) $1.15e+3$ sec (100%)
simulation_lbp.py(main) $1.15e+3$ sec
slicSuperpixel_lbp_method.py(__init__) $1.09e+3$ sec
slicSuperpixel.py(slic) 957 sec
skimage.segmentation_slic_slic_cython 932 sec

Table 5.2: LBP method runtime

runtime
simulation_lbp.py(module) 58.5 sec (100%)
simulation_lbp.py(main) 57.9 sec
ROI_lbp_method.py(__init__) 37.4 sec
ROI_lbp_method.py(predictMaskedImage) 35.4 sec
ROI_lbp_method.py(describe) 17.6 sec
cv2.imread 9.20 sec

In table 5.1the runtime of the Superpixel SLIC method is shown. It uses the SLIC superpixel segmentation algorithm implemented as in (Rodriguez-Teiles et al., 2014). While in table 5.2 one can see the LBP method proposed in this thesis and it‘s runtime.

The runtime of the skiimage SLIC segmentation in itself is 932 seconds and total of 1153 seconds compared to the total of 58.5 seconds it takes to run the LBP method.

That is an improvement of

$$\frac{1153}{58.5} = 19.7$$

This is almost an improvement of 20 times.

5.1.3 Summery of segmentation comparison

The prediction and runtime has been shown for two different segmentation method using the same descriptor for prediction. It has been shown that the SLIC superpixel segmentation is a huge bottleneck, being extremely slow while giving

similar results as a simpler segmentation method. Therefore, the slow method is discarded and the simpler method should be used in further development.

5.2 Run time comparisons between the three methods

To know which methods have the greatest potential, profiling of the code from each method is done. The profiling is visualized using the snakeViz python package, but since the figures are hard to read, tables are instead presented. Please note that the first row is the program and take 100 % of the time, the rows bellow are methods within it and are part of this time.

How does the LBP Method compare to the Disparity Method

Table 5.3: Disparity method runtime

runtime
simulation_disparity.py(module) 69.5 sec (100%)
simulation_haralick.py(main) 69.2 sec
simulation_disparity.py(process) 48.5 sec
simulation_disparity.py(disparityCalc) 74 sec
simulation_disparity.py(getDisparity) 54.1 sec
cv2.undistort 16.9 sec cv2.imread 9.18 sec

The disparity method uses 69.5 seconds. The majority of the runtime is linked to the stereo block matching algorithm and the undistortion of the stereo pair. Compared to the LBP prediction algorithm it is actually slower by $69.5 seconds - 58.5 seconds = 11 seconds$. And the speedup is:

$$\frac{69.5}{58.5} = 1.19$$

It is 1.19 times faster.

The speed of the disparity method can be increased by resizing the stereo pair before block matching.

The Haralick method runtime

Table 5.4: Haralick method runtime

runtime
simulation_haralick.py(module) 101 sec (100%)
simulation_haralick.py(main) 100 sec
ROI_haralick_method.py(__init__) 75.9 sec
ROI_haralick_method.py(predictMaskedImage) 74 sec
ROI_haralick_method.py(describe) 54.1 sec
cv2.imread 9.25 sec

From the table in 5.4 we can see that the runtime of the Haralick is way slower than the LBP. This is because it has more complex features to compute than the LBP method. The total time is 101 seconds. $101seconds - 58.5seconds = 42.5seconds$. and the speedup is:

$$\frac{101}{58.5} = 1.726$$

The LBP method is 1.726 times faster

5.2.1 Comparing the methods

To compare the methods, we have chosen to describe the images in a sequence by it‘s mean value. By comparing the mean values for each image by the different methods, we can analyze the quality of the methods.

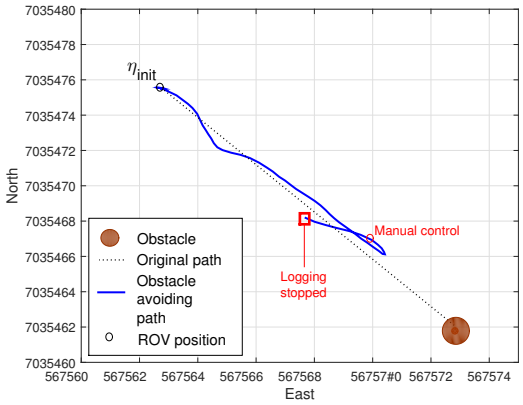


Figure 5.2: Plot of the U turn

The image sequence used is from the field test and reconstruct the situation where the ROV encounters the obstacle and reacts and turn away backward and can be seen in figure 5.2. It was chosen as a proper testing sequence since the ROV slowly encounters the obstacle and then turn back again. It is possible to compare the time the methods detect the obstacles and how the "mean value" change over the image sequence.

Figure 5.3: Images in chronological order

In figure 5.3 : a set of images in sequential order to visualize the images used in the simulation. Note: One can observe that the ROV gets closer and closer to the transponder tower, and then backs away in the opposite direction.

Calculating the mean value of the obstacle image:

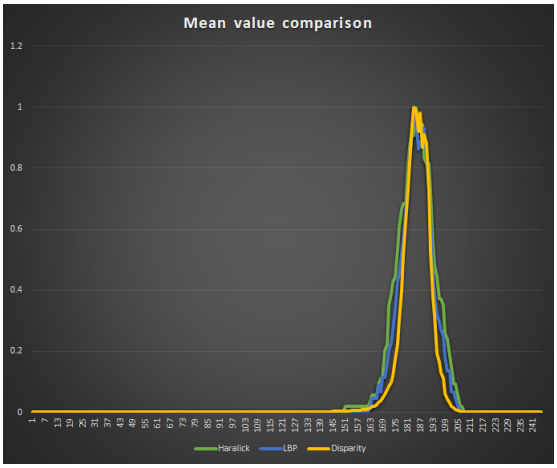


Figure 5.4: plot of meanvalue

The graph in figure 5.4 are from observing the obstacle from a sequence of 245 images. To get a clearer image, we shorten the x axis around the area it recognises the obstacle as seen in the following figure. Please note that the graphs are normalized, and that their maximum y-value is 1.

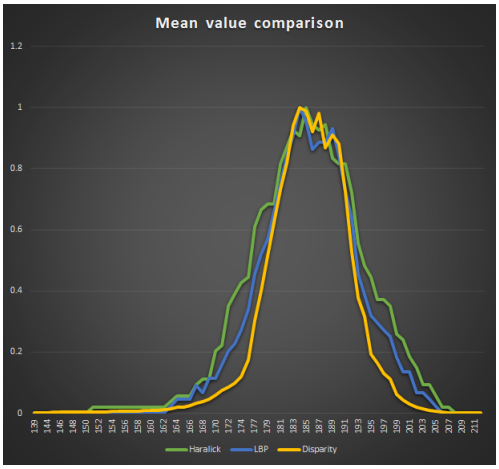


Figure 5.5: plot of meanvalue

As one can see in figure 5.5,the plots are very similar and observe the obstacle at the same time. They also have approximately the same rate of change.

Figure 5.5, led to the inspiration of a new way to use the “mean value” over time information as an auto distance or anchor function that is further explained in 7.2

5.3 Bad lighting comparison

It is interesting to see how the texture based methods perform on an image with difficult visibility. To do an evaluation, an image from a river with a group of fish has been chosen. It‘s hard even for the human eye to see what is “ocean” and “other”.

Figure 5.6: Test image

Figure 5.6 has been used as a testing image to evaluate how well the texture based methods are able to predict “ocean” and “other” in an image with a bright background and fishes. Note that ocean has been trained with blue water background, and could therefore be improved if trained with this bright watercolor.

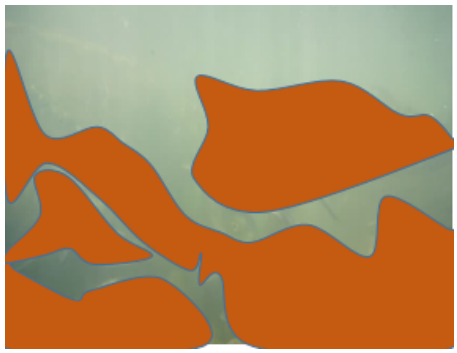
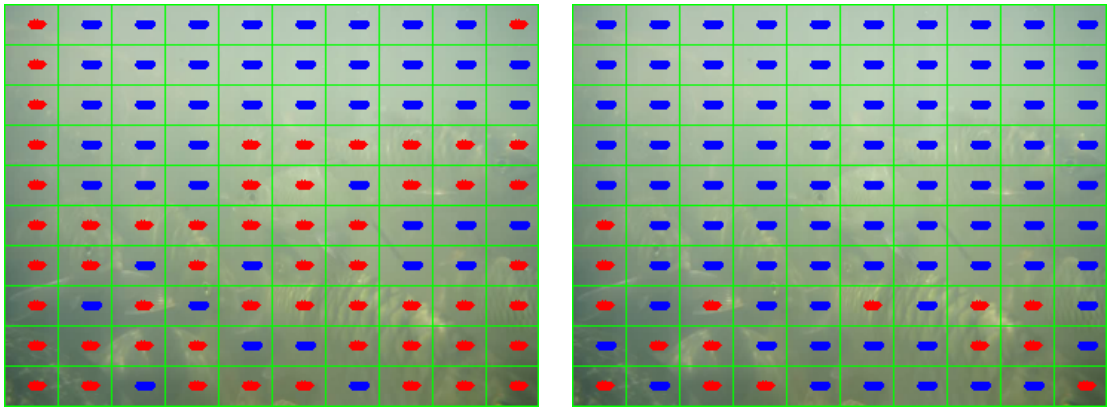


Figure 5.7: Human analysis

If figure 5.7 a human estimation of what is fish and what is ocean is given as a way to compare. The colored are is the area where there is “other”.



(a) *LBP method* (b) *Haralick method*

Figure 5.8: *Bad lighting comparison*

In figure 5.8 one can see a comparison between LBP method and Haralick method on a test image with bad lighting.

One can clearly see that the LBP method observes almost all obstacles as the human did, while the Haralick method works badly in this setting.

5.4 AurlabCVsimulator simulation with LabVIEW program

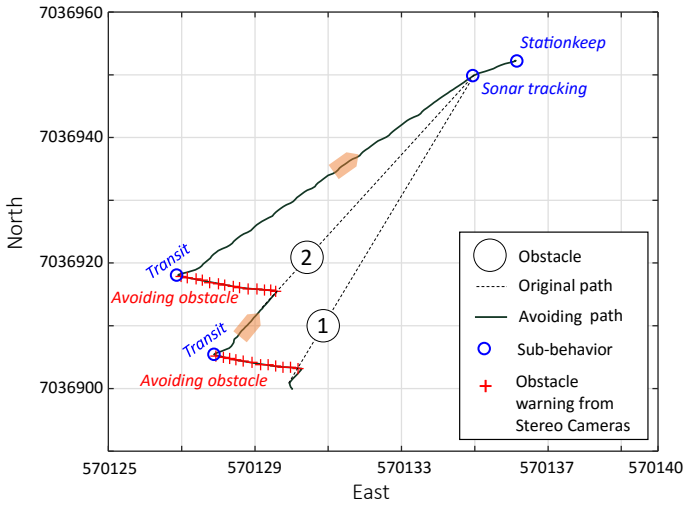


Figure 5.9: Simulation of Obstacle avoidance using the Disparity method. Courtesy of (Rist-christensen, 2016)

The figure 5.9 is a result from integrating the LabVIEW HIL simulator program and the AurlabCVsimulator running a joint simulation where the ROV encounters two obstacles simulated. The red crosses in the figure are from the *simulator* sending the message status "1" (represents an obstacle being detected) over UDP. In the message the center of the obstacle is also used in this plot.

5.5 Summary of results

The AurlabCVsimulator was used to test the all three methods described in chapter 3, and gave satisfying results.

In figure 5.9 the disparity method was used in the simulation.

Chapter 6

Analysis and Discussion

In this chapter, uncertainties, pros and cons of the methods used in this thesis are analysed and discussed.

6.1 Uncertainties and Obstacles

The capturing of the data used for the simulation can have uncertainties that result in incorrect data. There can always be uncertainties in results, equipments, and research work.

6.1.1 Uncertainties by removing error margin in the Disparity method

In the Disparity method the error margin is removed by excluding a part of the image from the right hand side. This means that some of the information about the ROV‘s view on the right hand side is lost. Therefore, it will perform worse when an obstacle is placed to the right hand side of the cameras view than to the left.

6.1.2 Synchronization uncertainties

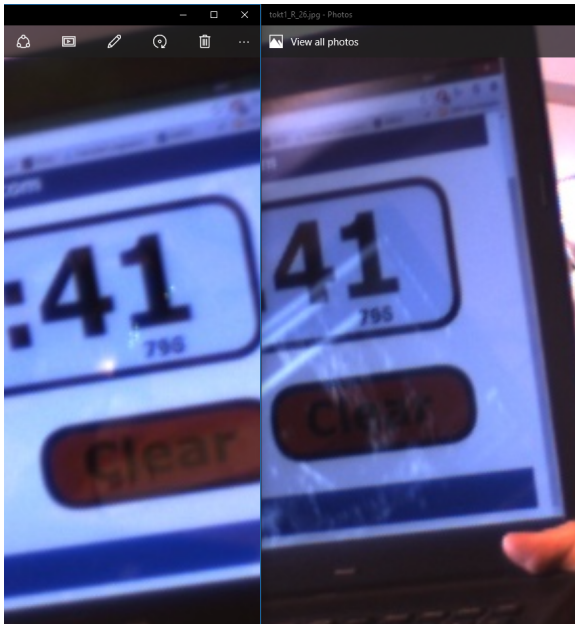


Figure 6.1: image of clock from stereo rig

The clock of the stereo camera can be out of sync making the stereo block matching algorithm less accurate, if the images matches are not taken at the same time. Figure 6.1 shows a test to see if the stereo rig is running synchronised. The test is to take an image of a clock and then compare the two images, if they capture the images at the same time it is acceptable.

6.1.3 Dust clouds

If the ROV is close enough to the seafloor it can make mud mix with the clear water, and this will create a change in the water clarity so it will effect the camera object avoidance system at that location for a short period of time.

6.1.4 Fish

A school of fish swimming in front of the camera, can affect the obstacle avoidance algorithm. Therefore, one can in a future implementation use optical flow (or another promising method), to recognize that there moving objects in front

of the camera, as, i.e., fish, and take that into the equation of obstacle avoidance detection.

6.1.5 Multiple simultaneous obstacles

The obstacle avoidance algorithm might end up getting trapped in a situation like this:

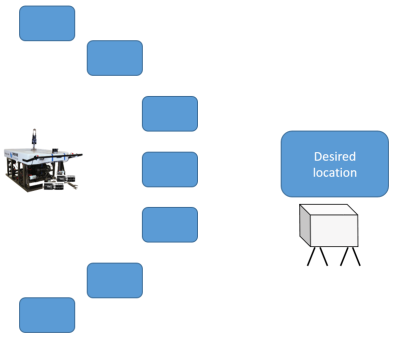


Figure 6.2: Example of a challenging situation

In figure 6.2, an example of a situation where the ROV gets stuck is shown.

This situation has the program not been tested for, and it is therefore unknown what would happen. In such a situation, the program should use the "mean value" to decide if there are too much obstacles and then back up until there it sees no obstacle and then try again with a new direction.

6.2 Pros and Cons of the methods

From the comparison of mean value in the results chapter, it has been proven that all the methods reacts around the same time for discovering an obstacle. The Haralick method is the most sensitive and reacts first, while the disparity and LBP method react at the same time.

The LBP method reacts better to obstacles in difficult light conditions as in the bad lighting comparison. It is therefore considered a more robust solution. Although the Haralick method responded faster in the image sequence, the LBP method was more robust and hence have potential for achieving better performance.

The runtime of the different methods is also important, even though the Haralick method reacts to the obstacle first in the image sequence, it is important to note that the LBP method is 1.726 times faster and would therefore be able to capture information at almost twice the frequency. The speed of the movement could therefore be increased, and would in that way be able to cover greater areas at a shorter time window. This is important for reducing time of offshore operations and could lead to cost savings.

The disparity method on the other hand could be tuned and improved similar to what they do in the paper (Barry & Tedrake, 2015), where they are able to calculate the disparity map needed for obstacle avoidance at 0.02 sec. But since disparity needs two cameras that have to be pre-calibrated and also lighting and water quality can have an effect on the calibration it is a very expensive method to use. It is expensive since the ROV needs a heavier payload, and it needs more time since one needs to calibrate the camera in the ocean because of refraction effects (how light breaks between the camera and the water).

Chapter 7

Conclusions and Further Work

This chapter presents the conclusions and proposed further work regarding this thesis.

The Obstacle avoidance modes developed are presented in the list below.

- Disparity method
- Locally Binary Pattern method
- Haralick method

7.1 Conclusion

Chapter 5 present the results and comparisons between the methods. It shows that they are all successful, but that the Haralick descriptor is a bit more sensitive, and therefore captures the obstacle a few images before the LBP method. The Disparity method provides a benchmark comparison for mono camera based obstacle avoidance methods.

The disparity method and the texture based methods were successfully implemented and tested. The disparity method was proven to work in sea trials at Trondheimsfjord.

Under simulated experiments, the methods have been compared, and all of them gave excellent results in obstacle avoidance.

The methods were developed to work in real time for underwater application. The most promising method is the LBP method and the Disparity method.

The LBP method has great potential also for other underwater applications as

it is rotation and illumination invariant. Therefore, it is robust under different rotation and light conditions, hence making it the method with potential for having the best performance.

This thesis has investigated several obstacle avoidance algorithms. The programs were proven to preform accurately in sea trials and in simulations.

The Computer Vision program have been able to extend the capabilities of the LabVIEW program as part of a new autonomous program for the ROV that is under development.

In conclusion the software developed in this thesis can perform obstacle avoidance with acceptable accuracy and is recommended for further investigation.

7.1.1 Performance of the chosen segmentation

In the method proposed by (Rodriguez-Teiles et al., 2014) they segment the whole image using the SLIC Superpixel algorithm as in (Achanta, R and Shaji, A and Smith, K and Lucchi, A and Fua, P and Süsstrunk, Sabine, 2011).

The trade-offs between the speed and the quality of the prediction have to be considered. Time and precision comparisons have been presented in chapter 5.

The LBP method and Haralick method achieved an improved speed and performance by using the simpler segmentation method.

7.2 Further Work

There are several further investigations that have potential for improving the range of applications for the proposed methods for the obstacle avoidance system.

The texture based methods have great potential for further development. It would be interesting to look at how one could use these methods to derive information for automatic scene interpretation. One could experiment with training the classifier for more objects as the model can classify more than two classes. For instance, it might be interesting to train the classifier for “reef”, “sand”, “subsea installation” and try to perform autonomous scene interpretation based on such classifiers.

The disparity image contains noise that makes it hard to reconstruct 3D models from the image pair; one could investigate further how to fix the noise problem.

If one can get disparity images with little noise, underwater computer vision SLAM module for the ROV might be developed.

One could create and test a DP anchor module based on the LBP method, as it is robust to light and rotation. Moreover, the user should be able to specify the object it wants to set as an anchor.

The LBP method could be made to run faster using (GPU) multithreading and (CPU) threading. Moreover, one could vectorize more of the "for" loops implemented in Python.

The LBP and Haralick method could be improved by using/gathering more training data for "ocean" and "other."

Mean Value

Over time, the images change and therefore the obstacle image also changes. By calculating the mean value of the binary image created by the method, it can be observed whether the obstacle size is increasing or decreasing. This is similar to the relative scale effect that the (Mori & Scherer, 2013) paper use, although they look at the relative size of SURF generated keypoints over time.

Disparity method improvement

The disparity method could be improved by doing a better calibration, but there are also other methods to improve the disparity. A design similar to the one introduced in (Barry & Tedrake, 2015), where the goal was to cause sparse disparity detections with few false positives is recommended for further investigation. This could greatly improve the speed of the algorithm. In fact, the algorithm in their method only used 0.02 seconds to process one image pair. By only comparing image blocks that go through a filter and rejecting blocks with a lack of edges, they were able to remove false positives, which might cause the ROV to turn away from a phantom obstacle.

The disparity method could remove repeating textures in the image (Barry & Tedrake, 2015) where a horizontal invariance filter was enabled to remove repeating textures that cannot be disambiguated with only two cameras.

Anchor system for Remotely Operated Vehicle

The idea is that by storing a queue of the last value of the center of an obstacle, one could use this information to make the ROV try to have that object in the center. In this thesis work, the Computer Vision program is dependent on the control system is written in LabVIEW. To implement the code in LabVIEW was outside the scope of the thesis, however, it is recommended for further investigation.

But one could imagine training the classifier for an image of a particular subsea structure one wants to "park" the ROV in front of, using visual servoing. It could take advantage of the "Mean Value" information as a relative distance measurement, and the "centerPosition" queue as an indication of movement sideways and in the up-down direction. Therefore, it would have sufficient information to stand still in DP(Dynamic Positioning) in front of the desired obstacle.

All the necessary information is already computed and sent over UDP; one needs only to implement and tune the controller.

7.2.1 Detect errors on pipelines

Figure 7.1: Pipeline example image. Courtesy of (*Subsea Pipeline Inspection, Repair and Maintenance - Theon - Oil and Gas Front End Consultancy & Petroleum Engineering Consultants — Theon - Oil and Gas Front End Consultancy & Petroleum Engineering Consultants*, 2016).

In order to maintain subsea pipes, one needs to inspect the pipes. If one could classify typical errors on pipelines that have to be maintained, one could train the classifier to detect these and then geo-reference or path-reference them. In that way, one could set up a mission for control of pipelines and automatically get feedback without having a human operator. The system could be applied to any underwater drone, and an AUV would be the best fit.

7.2.2 Application of camera based obstacle avoidance

It could be interesting to apply the methods on ROV's so that one does not need to drive the ROV to the desired position. But instead, could give a depth and a longitude and latitude and the ROV would drive to the specified coordinates.

7.2.3 Implement structure of motion using OpenSFM

OpenSFM is a Structure of Motion library written in Python on top of OpenCV. The library can be used to reconstruct the camera poses and 3D scene from multiple images. It is possible to integrate sensors such as accelerometers and more to improve the robustness and geographical alignment.

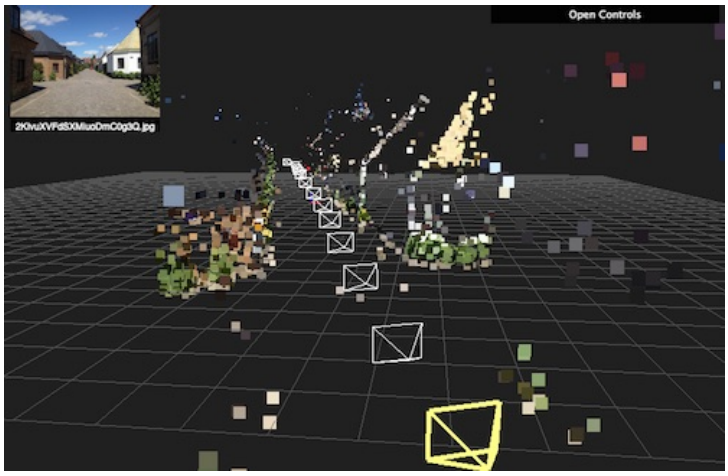


Figure 7.2: OpenSFM example image. Courtesy of (*mapillary/OpenSfM: Open Source Structure from Motion pipeline*, 2016).

In figure 7.2 an example of the javascript viewer provided within the software.

The software is available on GITHub at `https://github.com/mapillary/OpenSfM`.

References

Achanta, R and Shaji, A and Smith, K and Lucchi, A and Fua, P and Süsstrunk, Sabine. (2011). SLIC Superpixels Compared to State-of-the-Art Superpixel Methods. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *34*(11), 2274–2282. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6205760papers3://publication/doi/10.1109/TPAMI.2012.120 doi: 10.1109/tpami.2012.120

Agisoft photoscan. (2016). http://www.agisoft.com/. ((Accessed on 07/15/2016))

Ahonen, T., Hadid, A., & Pietikäinen, M. (2004). Face Recognition with Local Binary Patterns. *Computer Vision - ECCV 2004 SE - 36*, *3021*, 469–481. Retrieved from http://dx.doi.org/10.1007/978-3-540-24670-1{_}36$\delimiter"026E30F$nhttp://link.springer.com/10.1007/978-3-540-24670-1{_}36 doi: 10.1007/978-3-540-24670-1_36

AUR-Lab Deployment av LBL nett ved TBS. (2013). , 1–12.

Barry, A. J., & Tedrake, R. (2015). Pushbroom stereo for high-speed navigation in cluttered environments. In *Robotics and automation (icra), 2015 ieee international conference on* (pp. 3046–3052).

Brownlee, J. (2013). *Machine Learning Mastery with Python*. doi: 10.1017/CBO9781107415324.004

Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, *2*, 121–167. https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/svmtutorial.pdf.

Camera calibration toolbox for matlab. (n.d.). http://www.vision.caltech.edu/bouguetj/calib_doc/. ((Accessed on 06/15/2016))

Camera calibration toolbox for matlab. (2016a). http://www.vision.caltech.edu/bouguetj/calib_doc/. (Accessed on 06/27/2016)

Camera calibration toolbox for matlab. (2016b). http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example.html. ((Accessed on 07/15/2016))

Coelho, L. P. (2013a). Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software*, *1*(1), e3. Retrieved from http://openresearchsoftware.metajnl.com/article/view/jors.ac/5 doi: 10.5334/jors.ac

Coelho, L. P. (2013b, July). Mahotas: Open source software for scriptable computer vision. *Journal of Open Research Software*, *1*. doi: http://dx.doi.org/10.5334/jors.ac

Create gray-level co-occurrence matrix from image - matlab graycomatrix -

mathworks nordic. (n.d.). http://se.mathworks.com/help/images/ref/graycomatrix.html. ((Accessed on 07/14/2016))

Dissecting the camera matrix, part 2: The extrinsic matrix. (2016). http://ksimek.github.io/2012/08/22/extrinsic/. ((Accessed on 07/12/2016))

Fossum, Trygve Olav and Ludvigsen, Martin and Nornes, Stein M and Ristchristensen, Ida and Brusletto, Lars. (2016). AUTONOMOUS ROBOTIC INTERVENTION USING ROV:AN EXPERIMENTAL APPROACH.

Green's theorem - wikipedia, the free encyclopedia. (2016). https://en.wikipedia.org/wiki/Green%27s_theorem. (Accessed on 06/21/2016)

Haralick, R., & Shanmugam, K. (1973). Computer Classification of Reservoir Sandstones. *IEEE Transactions on Geoscience Electronics*, *11*(4), 171–177. Retrieved from http://ieeexplore.ieee.org/xpl/login.jsp?tp={&}arnumber=4071646{&}url=http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=4071646 doi: 10.1109/TGE.1973.294312

Haralick, R. M., Shanmugam, K., & Dinstein, I. H. (1973). Textural features for image classification. *Systems, Man and Cybernetics, IEEE Transactions on*(6), 610–621.

Hartley, R. I., & Zisserman, A. (2004). *Multiple view geometry in computer vision* (Second ed.). Cambridge University Press, ISBN: 0521540518.

Heikkilä, M., & Pietikäinen, M. (2006). A texture-based method for modeling the background and detecting moving objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *28*(4), 657–662. doi: 10.1109/TPAMI.2006.68

High sensitivity 1.4 Megapixel CCD camera with GigE Vision. (n.d.). , 1–4.

Introduction to support vector machines — opencv 2.4.13.0 documentation. (2016). http://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html. ((Accessed on 07/13/2016))

Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning* (pp. 137–142).

Kostavelis, I., Nalpantidis, L., & Gasteratos, A. (2009). Real-time algorithm for obstacle avoidance using a stereoscopic camera. In *Third panhellenic scientific student conference on informatics.*

Lens basics — understanding camera lenses. (2016). http://www.exposureguide.com/lens-basics.htm. ((Accessed on 07/10/2016))

Local Binary Pattern for texture classification — skimage v0.12dev docs. (2016). http://scikit-image.org/docs/dev/auto_examples/plot_local_binary_pattern.html. (Accessed on 06/18/2016)

Machine learning - wikipedia, the free encyclopedia. (2016). https://en.wikipedia.org/wiki/Machine_learning. (Accessed on 06/27/2016)

mapillary/opensfm: Open source structure from motion pipeline. (2016). https://github.com/mapillary/OpenSfM. ((Accessed on 07/13/2016))

Miyamoto, E., & Jr., T. M. (2011). FAST CALCULATION OF HARALICK TEXTURE FEATURES Human Computer Interaction Institute Department of Electrical and Computer Engineering Carnegie Mellon University. *Human Computer Interaction Institute Department of Electrical and Computer Engineering Carnegie Mellon University Pittsburgh PA*, *15213*, 1–6. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.9719{&}rep=rep1{&}type=pdf

morefigs/pymba: Python wrapper for the allied vision technologies (avt) vimba c api. (2016). https://github.com/morefigs/pymba. ((Accessed on 07/10/2016))

Mori, T., & Scherer, S. (2013). First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In *Robotics and automation (icra), 2013 ieee international conference on* (pp. 1750–1757).

Navab, N. (2009). *3d computer vision ii.* http://campar.in.tum.de/twiki/pub/Chair/TeachingWs09Cv2/3D_CV2_WS_2009_Reminder_Cameras.pdf. ((Accessed on 07/12/2016))

Ng, A. (n.d.). *Machine learning - stanford university — coursera.* https://www.coursera.org/learn/machine-learning. ((Accessed on 07/14/2016))

Ojala, T., Pietikäinen, M., & Mäenpää, T. (2002). Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *24*(7), 971–987.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, É. (2012). Scikit-learn: Machine Learning in Python. *... of Machine Learning ...*, *12*, 2825–2830. Retrieved from http://dl.acm.org/citation.cfm?id=2078195$\delimiter"026E30F$nhttp://arxiv.org/abs/1201.0490 doi: 10.1007/s13398-014-0173-7.2

Pietikäinen, P. M., & Heikkilä, P. J. (2011). Image and Video Description with Local Binary Pattern Variants. *Group*.

Profiling (computer programming) - wikipedia, the free encyclopedia. (2016). https://en.wikipedia.org/wiki/Profiling_(computer_programming). ((Accessed on 07/06/2016))

Rist-christensen, I. (2016). Autonomous robotic intervention using ROV.

Rodriguez-Teiles, F. G., Pérez-Alcocer, R., Maldonado-Ramirez, A., Abril Torres-Mendez, L., Dey, B. B., & Martinez-Garcia, E. A. (2014). Vision-based reactive autonomous navigation with obstacle avoidance: Towards a non-invasive and cautious exploration of marine habitat. In *Robotics and automation (icra), 2014 ieee international conference on* (pp. 3813–3818).

Rolling shutter - wikipedia, the free encyclopedia. (2016). https://en.wikipedia.org/wiki/Rolling_shutter. ((Accessed on 07/07/2016))

scikit-image: Image processing in python — scikit-image. (2016). http://scikit-image.org/. ((Accessed on 06/21/2016))

Second moment of area - wikipedia, the free encyclopedia. (2016). https://en.wikipedia.org/wiki/Second_moment_of_area. ((Accessed on 06/21/2016))

Semantic gap - wikipedia, the free encyclopedia. (2016). https://en.wikipedia.org/wiki/Semantic_gap. (Accessed on 06/27/2016)

Subsea pipeline inspection, repair and maintenance - theon - oil and gas front end consultancy & petroleum engineering consultants — theon - oil and gas front end consultancy & petroleum engineering consultants. (2016). http://www.theonltd.com/news/subsea-pipeline-inspection-repair-and-maintenance/. (Accessed on 06/27/2016)

Support vector machine - wikipedia, the free encyclopedia. (2016). https://en.wikipedia.org/wiki/Support_vector_machine. ((Accessed on 07/13/2016))

Visual odometry autonavx courseware — edx. (n.d.). https://courses.edx.org/courses/course-v1:TUMx+AUTONAVx+2T2015/courseware/b6f17cd5224f43d8a29aca3dcbc15902/9f52ee0a957047119ce4644f71100a4a/. ((Accessed on 06/15/2016))

Welcome to opencv documentation! — opencv 2.4.9.0 documentation. (n.d.). http://docs.opencv.org/2.4.9/. ((Accessed on 06/15/2016))

Wesley, A. (2000). *University physics. 10th edition.*

What is camera calibration? - matlab & simulink - mathworks nordic. (2016). http://se.mathworks.com/help/vision/ug/camera-calibration.html. ((Accessed on 07/10/2016))

Appendix A

Ocean 16 paper

AUTONOMOUS ROBOTIC INTERVENTION USING ROV: AN EXPERIMENTAL APPROACH.

Trygve O. Fossum, Martin Ludvigsen, Stein M. Nornes, Ida Rist-Christensen, Lars Brusletto
Department of Marine Technology,
Norwegian University of Science and Technology, NTNU,
NO-7491 Trondheim, Norway.
Email:{trygve.o.fossum,stein.nornes,martin.ludvigsen}@ntnu.no

***Abstract*—Using an experimental approach, this paper proposes a semi-autonomous agent architecture for a remotely operated vehicle (ROV). The system is inspired by Behavior- and Reactive-based architectures using stimulus response blocks to segment behavior. The capability and limitations of the system is demonstrated through a field experiment, where the goal is to approach and localize a structure of interest (SOI). The system is tested using Hardware-In the-Loop (HIL) simulations before deployment. The motivation for our approach is testing and verification of architecture feasibility in an environment similar to an operational situation. The results from the field campaigns demonstrate the ROV agent able to execute an inspection type mission, navigating to the SOI from surface, while avoiding obstacles.**

I. INTRODUCTION

In general, current industrial ROV operations are directly and manually controlled, with neither automatic control functions nor autonomy [22]. The operator must understand and process a wide range of information, while keeping safe distance to potential obstacles and seafloor terrain. Autonomous systems can alleviate and simplify the mission complexity, making the operation less dependent on operator skill, whilst providing increased precision, and reduced ship time. Increasing efficiency and regularity in ROV operations is important in a landscape where the industry is challenged to cut cost; consequently, automation should be considered. The combination of modern ROV motion control systems, see e.g. [11] and [8], and the advent of autonomous underwater vehicles (AUVs), has leveraged the development of autonomy in the ROV domain. An important factor to note in this regard is the development of station-keeping, which allows for independent control of ROV and manipulator arms. Also, advancements in telerobotics and remote control systems have had impact. Intervention capable AUVs are currently only at the research stage, and not fully developed for industrial application [12]. There are only a few vehicles equipped with manipulators and generally AUVs are purposed to perform survey missions; more on the topic of AUV intervention can be found in [14]. Expanding autonomous capabilities of ROVs can be considered with the notion that the system only need to incorporate a reduced degree of self-dependence. The platform enjoy the benefits of human supervision and continuous contact with the vehicle, which removes demand for full autonomy. The operator would monitor the operation and intervene only when unanticipated behavior is noticed. This conditions ROV autonomy – compared to AUV autonomy – to be semi-autonomous. Inspection, maintenance and repair (IMR) on subsea infrastructure is a natural application where this potential can be explored (i.e. locating a SOI).

The outline of the paper is given in the following. Section II describes the autonomous agent design, and the integration of computer vision. Selected results from field trials are presented in Section III. Finally, Section IV summarizes the concluding remarks.

A. Related work

The previous work on ROV automation is limited. Normally the attention is directed towards AUV research, but comparisons can be made where the system application is similar. A review of recent advances for autonomous marine robots and use of intelligent systems is given in [23]. An overview of traditional Behavior-Based Methods for underwater vehicle control is found in [5]. Most autonomous control [7] systems today are a variant of the Behavior-Based subsumption architecture, often attributed to [3]; having a large influence on the field. The work cited above focuses mainly on the AUV platform. However, the material is relevant for all marine robots. Periodic inspection, autonomous docking and valve turning, component maintenance, and repair are operations considered in literature applicable for robotic adaptation. Consequently, the research reported on manipulator control couples in with ROV autonomy. Automated valve turning is considered in [4] and [1], where a robotic arm aim to turn a valve in an underwater environment. The strategy is based on imitation and machine learning. A ROV prototype with high level autonomy capabilities is presented in [17]; the ROV is supported by an operational environment platform (OceanRINGS), installed in the ROV and the control cabin. Both ROV and environment platform are tested in field trials, see e.g. [16]. The NTNU based company *Eelume* have developed an intervention capable AUV snake robot, intended to fully automate the control of a subsea template, see [9].

B. Autonomy levels

An important factor to consider is the degree of autonomy adequate for practical use in ROV applications. It is anticipated

that future operation of ROVs will involve more autonomy, still relying on pilot support, but rather as a operational supervisor. There are several existing scales depicting levels of autonomy. One example is presented here from Chris et al. (2014) dividing autonomy for ROVs into five levels:

1. *Direct Control:* Control is executed within the actual vehicle by a pilot.
2. *Remote Control:* Having visible contact with the vehicle and remotely assigning control commands.
3. *Teleoperation:* Joystick control aided by visual information streamed from cameras. This is the most common type of control, with pilots sitting in a control-room.
4. *Logic Driven:* Semi-autonomous control using some level of logic driven programming. Waypoints and states are typically used in programmed execution.
5. *Logic Driven with goal orientation:* High level tasks and instructions are stated as goals and the ROV can perform these without human intervention.

The current level of autonomy used for ROV operations is largely restricted to level 1,2 and 3; though there exist examples and prototype systems having higher level capabilities. Most modern work-class ROVs have the capability to do auto-heading/depth/altitude, and incorporate dynamic positioning (DP) systems.

C. Autonomous architectures

Several types of Behavior-based architectures exist in marine robotics. These systems make up the framework containing control laws, error detections, recovering, path planning, task planning, and monitoring of events during mission execution. A common way to differentiate is between; Deliberative, Reactive, and Hybrid type systems.

1) **Deliberative** architectures generally include a world model, on which planning decisions are taken [5]. Planning can be done prior to mission, or adaptively in-situ. Highly advanced systems like T-REX (Teleo-Reactive EXecutive) [19] is an example of a Deliberative type system which can work adaptively in-situ. Maintaining an accurate world model is vital for favorable planning to proceed. Developing this model is a complex and laborious task, and mismatch will cause suboptimal behavior. However, suitable domains exist and several successful implementations and applications have been demonstrated, see e.g. [25] and [18]. Problems which need to be resolved considering the world model often are classified as NP-hard [23]. Time for internal computation (deliberation) can therefore be several minutes depending on implementation. Deliberative architectures follow the structure, *sense* $\rightarrow$ *plan* $\rightarrow$ *act*.

2) **Reactive** architectures are generally more simple to build and do not directly require a world model representation. The reaction time, compared to Deliberative systems, needs to be short, in order to react to non-predictable events. The complete behavior is divided into parallel sub-behavior blocks; each block is defined by reaction to a concrete stimulus and designated response. These independent blocks collect the incoming perceptions and calculate an output. The output of the different sub-behaviors are then collected and assessed inside a coordinating mechanism, which determines the resulting behavior [5]. One example would be to base the coordination on a hierarchical defined rule set; taking form as a "prioritizing switch", using *if-then-else* notation to handle behavior branching within the blocks. The resultant behavior can be a direct prioritization of one block or a combination. It is hence possible to obtain behavioral complexity inherited from such a summation. This is the principal idea behind Behavior-based approaches and apply to Deliberative as well as Reactive architectures. A central element for the Behavior-based architectures to be prosperous, is the correct composition of the coordinating mechanism. Thus, efforts must be spent designing this component carefully; focus on attaining behavioral predictability is important for ROV applications. The Reactive framework is comparably easy to implement and gives real-time response. Reactive architectures follow the structure, *sense* $\rightarrow$ *act*.

3) **Hybrid** solutions also exist. The Hybrid architecture integrates the Deliberative and the Reactive layer (in the hybrid context referred to as the Functional Reactive layer), assuring both satisfying mission conductance (high-level goal fulfillment) and quick adaptation to uncertainties (reflexive response). Similar to Reactive/Behavior-based architectures, Hybrid systems incorporate a coordinating mechanism responsible for regulating the task execution, generally addressed as the Control execution layer – responsible for enabling, disabling, and parametrization of the different behaviors. Several coordination schemes exist, e.g. fuzzy-implemented behaviors and hierarchical classifications of the behavior [10]. Hybrid systems have been successfully used in AUV applications. A more detailed description and comparison is given in [20]. Hybrid architectures follow the structure, *Deliberative* $\cup$ *Reactive* $\rightarrow$ *Coordinating mechanism* $\rightarrow$ *act*.

II. HYBRID AGENT ARCHITECTURE

Considering that a pilot will accompany and supervise the ROV; it is reasonable to argue for using a plain and intuitive architecture, keeping the behavior output deterministic and transparent. A Hybrid architecture type fits well to this purpose, since it can include properties for handling both fast and slow moving behavior. The system design characteristic was chosen to be based on Behavior-based principles; dividing behaviors into distinct sub-behaviors, having no explicit representation of knowledge, and having a

modular structure with each block dedicated to a particular aspect of the vehicle autonomy. This allows the system to be simplistic in structure and be open to modification. Further, the design method chosen is based on experimentally driven design, where the basic premise is to iteratively build skill components from running trials, and in turn add these to the system. The agent design have been verified and tested using a ROV HIL simulator. This is vital in order to assess the credibility and the behavioral model of the system before going into sea trials. The agent architecture presented follows the Hybrid framework, presented in section I-C, point (3), and consists of the Deliberative, Reactive, and Control execution layer. The behaviors are implemented and conditioned based on a typical ROV inspection and intervention mission; approaching, locating, and recognizing a SOI. The degree of autonomy for the agent, following the scale given in I-B, is comparable with level 5.

The **Deliberative layer** accounts for the slower moving behavior components, e.g. taking the ROV from surface to the SOI. This block is comprised of six underlying *sub-behaviors* which sequentially follow the mission anatomy, namely; *(1)Launch, (2)Descent, (3)Transit, (4)Sonar tracking, (5)Camera tracking, (6)Camera Inspection and Intervention*. Only one behavior is active at a time, implicating sequential steering. Each sub-behavior has a set of conditions which need to be fulfilled before progressing to the next state. The active sub-behavior block translate the sensor input (e.g. position, depth, altitude, camera, etc.) to high-level behavior (*sense* $\rightarrow$ *plan*). The functional mapping is discrete, yielding concrete actions – e.g. *setAltitude=10m*. Actions are then interpreted and executed ($\rightarrow$ *act*) by the vehicle control system, which handles the discrete to continuous decoding. The translation from sensor input to output is based on common planning formulations using *logic expressions* with time-dependent conditions, since several of the conditions are declared by being either *True* or *False* given a time limit – e.g. $X\ is\ True\ if\ (depth = 5m\ \&\&\ timer >= 1min)$, X representing a generic time-dependent condition.

Having such a simple mapping introduces certain tradeoffs, but is practically suited for the experimental context brought into play by running trials and building the autonomy system incrementally. In the design phase finding program bugs, faults, and carry out modifications becomes uncomplicated, which is a considerable benefit. Only the active sub-behavior receive sensor information within the Deliberative layer, whilst in the Reactive layer all sub-behaviors are active and receive sensor data. The **Reactive layer** consist of the fast-moving behavior components; *Obstacle avoidance*, and *Stuck detection*. Reactive components require limited environmental knowledge and integrate with the Deliberative layer by altering the mission on the fly. Making the ROV able to modify the existing plan based on real-time sensory information. Similar to the Deliberative layer the Reactive functional mapping is discrete – E.g. in the case of obstacle detection, waypoints and direction is calculated using computer vision (see Section II-A), and is sent to the vehicle control system in the form: *newWaypoint*=N52°60.323′, E8°32.144′. Important to note is that the Reactive layer have priority in the coordination mechanism, i.e. the **Control execution layer**, meaning the Reactive sub-behaviors will be granted control if either obstacles or stuck symptoms are detected. The coordination follows a hierarchical ruling with the Reactive layer on top. Due to the sequential steering, the Deliberative layer will never request several behavior actions at the same time. The coordination will therefore only have to reconcile between the Deliberative and the Reactive layer. The output from latter is always sent through the Control execution layer, before dispatch to the vehicle control system. A diagram of the **Hybrid agent architecture** is presented in Figure 1.

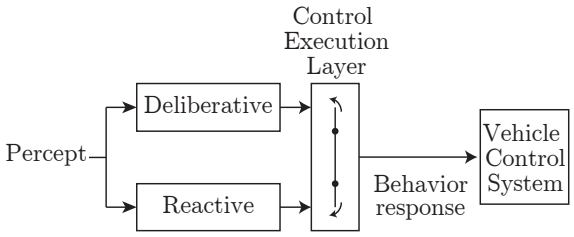
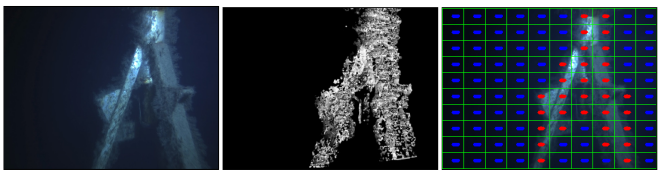


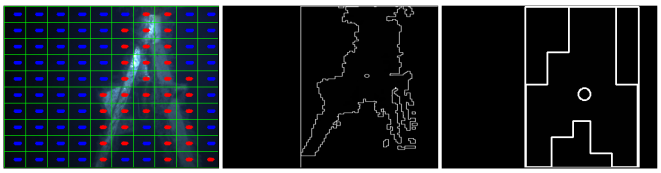
Fig. 1: Agent architecture.

A. Integration of computer vision

The Hybrid agent is complemented with computer vision for detection and avoidance of obstacles. Using visual information is not only necessary for resolving obstacle avoidance. Realization of autonomous IMR operations is dependent on computer vision based tools to deal with in-situ identification and precision control. The tools and methods used in this paper was developed to suit an underwater environment with a relative short visual range, inherit to the cloudy waters of the Trondheim Fjord. The main results are collected in Figure 2, followed by a summary of the theory and simulation results.



(a) Unaltered video frame of transponder tower. (b) Disparity image of obstacle. (c) LBP prediction.



(d) Haralick prediction (e) Disparity prediction. (f) LBP / Haralick prediction.

Fig. 2: Computer vision results.

The ROV was equipped with stereo cameras looking straight forward to simplify detection of obstacles. The video stream was received, and the computer vision output produced, on one single desktop computer. The computer communicated with the Hybrid agent, running on a similar computer, via the UDP protocol.

The first method of abstraction uses disparity maps. This approach creates a disparity map of the image pair in front of the ROV. A threshold filter, based on detection size, is utilized to avoid mis-identification of backscatter as obstacles. If the visual input precede the threshold values, the algorithm interpret this as an obstacle in the field of view, see Figure 2b. The algorithm then calculates the region of the obstacle and a corresponding bounding box with a center, referring to Figure 2e. The extracted visual information can now be used for safe path inference in an obstacle avoidance context. A weakness inherit to this approach is the dependence on the threshold filter, which can cause problems in low light conditions. Consequently, two additional methods, for obtaining a more robust obstacle bounding box prediction, are developed to complement the preceding approach using; Local Binary Patterns (LBP) (as described in [15]), and Haralick texture descriptors (as described in [13]). The main advantage of these particular texture descriptors is that ocean in itself has almost no texture and is therefore easy to distinguish from obstacles. Simulations show better accuracy in varying illumination – which is practical for underwater purposes. Uniform Local Binary Patterns is known to be illumination invariant, which make the approach appealing to use, providing robust prediction of regions of obstacles. Further augmentation is done to speed up results from [21], replacing the (slice-by-slice) SLIC Superpixel Segmentation algorithm; the modified methods simply segment the image into workable grid blocks. Simulations indicate improvement close to 20 times faster using this approach. The LBP and Haralick methods are also advantageous compared to the Disparity approach, removing the need to calibrate a stereo camera pair, and allowing predictions with only one camera active. The output from the LBP and Haralick method is drawn in Figure 2c, 2d. Comparable with Figure 2e, the bounding box produced by the LBP prediction is shown in Figure 2f; similar output is obtained with the Haralick approach. Having a more robust prediction in low light conditions, the LBP and Haralick methods are preferred over the Disparity approach.

The integration of the methods described was able, based on the bounding box and center prediction, to provide explicit detection and direction of avoidance. Direction is calculated based on the location of the obstacle in the field of view. Data from field trials in the Trondheim fjord made it feasible to rerun the field test for HIL testing and tuning of the obstacle avoidance. Figure 3 demonstrate HIL simulations using the Disparity prediction to avoid two obstacles in succession. The red crosses represent an ”obstacle detected” message. The message is sent until the obstacle no longer appear in the cameras. The ROV avoids the obstacle for a certain amount of time or until the obstacle is out of the camera range. Satisfactory results was also obtained in simulation using the LBP and the Haralick methods.

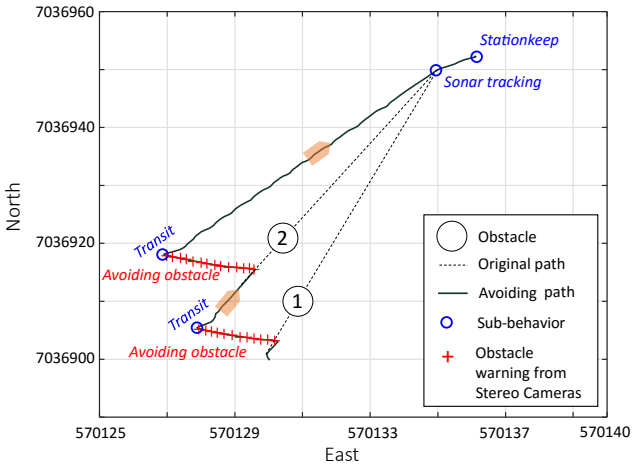
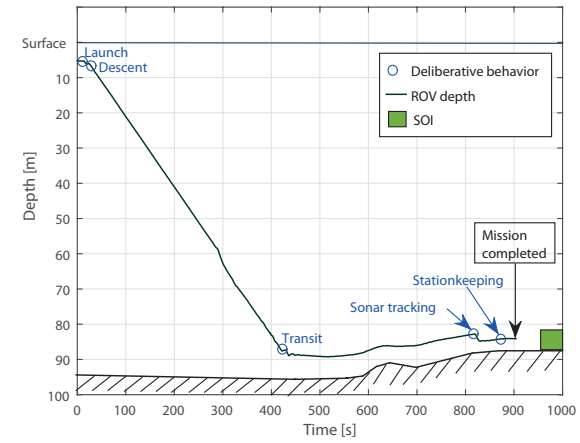


Fig. 3: HIL simulation of obstacle avoidance using computer vision.

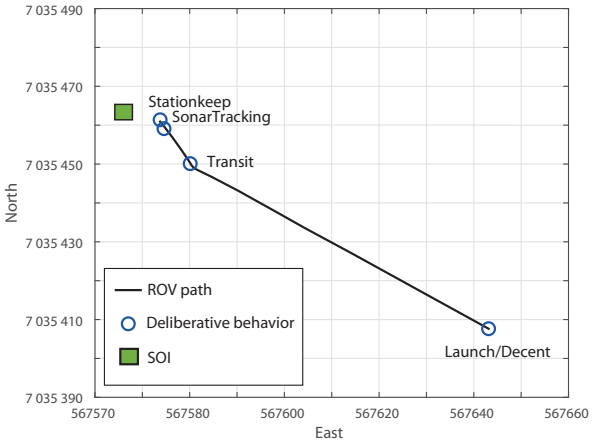
III. EXPERIMENTAL RESULTS

The Hybrid architecture was tested in the Trondheim Fjord April 2016 using a ROV SUB-Fighter 30k from Sperre AS, see [24]. The research vessel RV Gunnerus provided the premises for remote control and computer resources. The mission setup reflected an operational situation where the ROV was to approach and localize a SOI, in the form of a transponder tower submerged at 85m, see Figure 2a. The underlying sub-behaviors active in the experiment were; *(1)Launch, (2)Descent, (3)Transit, (4) Sonar tracking*, with pre-generated sonar data, and approximate position for the SOI provided for simplification. The ROV supervisor could intervene at any time, overriding the agent. The default and fail-safe sub-behavior was *Stationkeep*, keeping the ROV still at the current depth and location. The remaining sub-behaviors (5,6) were dormant. The autonomous agent and the vehicle control system were connected and operated separately from the computer vision computer. The experiment was organized in two parts; First part concentrated on validation of the Deliberative behavior and comparison with the pre-mission HIL simulations. The second part was testing with both the Deliberate and Reactive behavior active, reflected with a obstacle avoidance scenario. The ROV was manually taken to 5m water depth before activating the Hybrid agent. Figure 4a shows the Deliberate behavior taking the ROV from the surface to the SOI, switching successfully between the sub-behaviors (marked with blue text), in accordance with HIL simulations carried out on land. After attaining confirmation from the acoustic sensors, altitude control takes over from depth control, on request from the Hybrid agent – switching from Decent

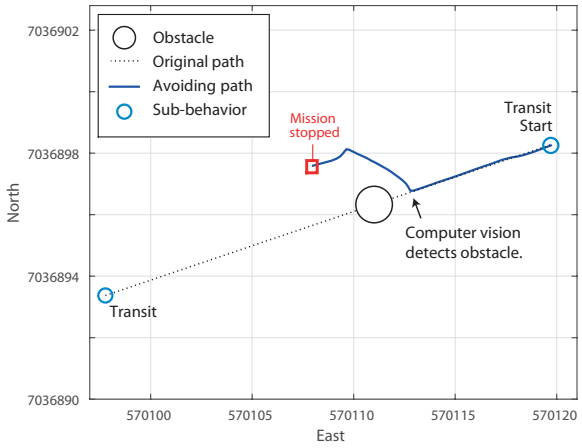
to the Transit behavior. The approach then continue with the provided location of the SOI, until a distance condition activates the pre-generated sonar data and the Sonar tracking behavior. The mission is concluded by the agent when arriving within 5m radius of the tower, going into the Stationkeep behavior. A overlooking plot is shown in Figure 4b.



(a) Depth-Time plot of the vehicle.



(b) North-East plot of the vehicle.



(c) Results from testing obstacle avoidance.

Fig. 4: Results from field trials.

For the second part of the experiment, both the Reactive and Deliberative layer were activated in order to test the Control execution layer, as well as the camera-triggered obstacle avoidance described in section II-A. Figure 4c shows this scenario with the planned destination waypoint blocked by an obstacle. The reconciliation between the layers is successful and the Reactive behavior – i.e. obstacle avoidance – takes control and steer clear, before handing control back to the Deliberative layer. The mission is stopped after successful avoidance has been verified.

IV. CONCLUSIONS AND FUTURE WORK

We have used a Hybrid agent architecture to show the autonomy potential for intervention type ROV missions. In addition, the system have demonstrated handling of non-deterministic events, i.e. obstacle avoidance, using computer vision for identification and navigation. There remains much work to be done; the lessons learned from field trials is to be included for next iteration and system version. In particular, all the sub-behaviors (1-6) are to be included, and further work on improving logic flow within the architecture is important. One can further imagine implementation of more elaborate computer vision methods, e.g. camera tracking, for attaining accurate navigation relative to the SOI.

It is anticipated that future operation of ROVs will involve more autonomy. Arguing for a semi-autonomous approach, with the pilot in a supervising role, a Behavior-based architecture is considered adequate for practical use in ROV applications. Having a plain and intuitive structure makes finding program bugs, faults, and carry out modifications uncomplicated, which is a considerable benefit in building ROV autonomy systems following a bottom-up approach. Certain tradeoffs are associated with Behavior based robotics, see e.g. [2, 23], where particular concerns relate to overall action selection and adaptation for the vehicle in off-nominal conditions. Additionally, the use of propositional logic for deliberation can cause problems for problem domains which are hard to represent. For routine and recurring ROV missions, this shortcoming is accepted, but can become a concern for more complex intervention activity, e.g valve operation, and manipulator tasks. System augmentation will in this case be necessary to obtain these capabilities, implicating development towards full autonomous control without human supervision. If this convergence is practical in the ROV domain, will not be advocated in this paper, but is a question which to be mindful of when planning the composition of autonomous agents.

ACKNOWLEDGEMENT

The work presented in this article is conducted through the NTNU Applied Underwater Robotics Lab (AURLab), providing equipment and resources for field trials. The funding is supported by the Research Council of Norway (RCN) under the project number, 223254.

References

[1] Seyed Reza Ahmadzadeh, Petar Kormushev, and Darwin G. Caldwell. "Autonomous robotic valve turning: A hierarchical learning approach". In: *Proceedings - IEEE International Conference on Robotics and Automation*. 2013, pp. 4629–4634. ISBN: 9781467356411. DOI: 10.1109/ICRA.2013.6631235.

[2] Michael Benjamin, Matthew Grund, and Paul Newman. "Multi-objective optimization of sensor quality with efficient marine vehicle task execution". In: *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE. 2006, pp. 3226–3232.

[3] Rodney A. Brooks. "A Robust Layered Control System For A Mobile Robot". In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23. ISSN: 08824967. DOI: 10.1109/JRA.1986.1087032. arXiv: 1010.0034.

[4] A Carrera et al. "Towards Autonomous Robotic Valve Turning". In: *Cybernetics and Information Technologies* 12.3 (2012), pp. 17–26.

[5] M Carreras et al. "An Overview on Behaviour-Based Methods for AUV Control". In: *Behaviour* (1999), pp. 141–146.

[6] Robert D. Christ and Robert L. Wernli. *"The ROV Manual" (Second Edition), Chapter 4: Vehicle Control and Simulation,* Elsevier, 2014, pp. 93–106. ISBN: 9780080982885. DOI: 10.1016/B978-0-08-098288-5.00003-8.

[7] McGann, Conor et al. "T-rex: A model-based architecture for auv control". In: *ICAPS 2007 - USA*. Vol. 2007. 2007.

[8] Fredrik Dukan, Martin Ludvigsen, and Asgeir J. Sørensen. "Dynamic positioning system for a small size ROV with experimental results". In: *OCEANS 2011 IEEE - Spain*. 2011. ISBN: 9781457700866. DOI: 10.1109/Oceans-Spain.2011.6003399.

[9] Eelume. Eelume - Intervention capable snake robot. http://eelume.com/. 2016.

[10] Jonathan Evans et al. "Design and evaluation of a reactive and deliberative collision avoidance and escape architecture for autonomous robots". In: *Autonomous Robots* 24.3 (2008), pp. 247–266. ISSN: 09295593. DOI: 10.1007/s10514-007-9053-8.

[11] Daniel de A Fernandes et al. "Output feedback motion control system for observation class ROVs based on a high-gain state observer: Theoretical and experimental results". In: *Control Engineering Practice* 39 (2015), pp. 90–102. ISSN: 09670661. DOI: 10.1016/j.conengprac.2014.12.005.

[12] Einar Gustafson et al. "HUGIN 1000 Arctic Class AUV". In: *Arctic Technology conference*. Vol. 2. 2011, pp. 1–8. ISBN: 9781613991725. DOI: 10.4043/22116-MS.

[13] Robert M Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. "Textural features for image classification". In: *Systems, Man and Cybernetics, IEEE Transactions on* 6 (1973), pp. 610–621.

[14] Giacomo Marani, Song K. Choi, and Junku Yuh. "Underwater autonomous manipulation for intervention missions {AUVs}". In: *Ocean Engineering* 36.1 (2009). Autonomous Underwater Vehicles, pp. 15–23. ISSN: 0029-8018. DOI: dx.doi.org/10.1016/j.oceaneng.2008.08.007.

[15] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 24.7 (2002), pp. 971–987.

[16] E. Omerdic and D. Toal. "OceanRINGS: System concept amp; applications". In: *Control Automation (MED), 2012 20th Mediterranean Conference on*. July 2012, pp. 1391–1396. DOI: 10.1109/MED.2012.6265833.

[17] Edin Omerdic. *ROV LATIS: next generation smart underwater vehicle*. Control, Robotics and Sensors. Institution of Engineering and Technology, 2012, pp. 9–44. DOI: 10.1049/PBCE077E_ch2.

[18] Frédéric Py, Kanna Rajan, and Conor McGann. "A Systematic Agent Framework for Situated Autonomous Systems". In: *Scientist* 128.44 (2010), pp. 583–590. ISSN: 00027863. DOI: 10.1021/ja065334o.

[19] Kanna Rajan and F Py. "T-REX: partitioned inference for AUV mission control". In: *Further advances in unmanned marine vehicles. The ...* (2012), pp. 171–199.

[20] P Ridao et al. "On AUV control architecture". In: *Intelligent Robots and Systems, 2000. (IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on* 2 (2000), 855–860 vol.2. DOI: 10.1109/IROS.2000.893126.

[21] F Geovani Rodriguez-Teiles et al. "Vision-based reactive autonomous navigation with obstacle avoidance: Towards a non-invasive and cautious exploration of marine habitat". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 3813–3818.

[22] Ingrid Schjølberg and Ingrid Bouwer Utne. "Towards autonomy in ROV operations". In: *IFAC Papers Online* (2015), pp. 183–188. ISSN: 24058963. DOI: 10.1016/j.ifacol.2015.06.030.

[23] Mae L. Seto. *Marine robot autonomy*. Springer-Verlag New York, 2013, p. 382. ISBN: 978-1-4614-5658-2. DOI: 10.1007/978-1-4614-5659-9.

[24] Sperre. SUB-FIGHTER 30K. http://sperre-as.com/portfolio/sub-fighter-30k/ [Accessed: 28.04.2016]. 2016.

[25] Ioannis Vlahavas and Dimitris Vrakas. *Intelligent Techniques for Planning*. English. IGI Global, Jan. 1. ISBN: DOI: 10.4018/978-1-59140-450-7.

Appendix B

Attachments

This appendix lists the attachments. The attachment is a zip file that includes the Msc poster while the software is online and available at the given hyperlinks in B.2.

B.1 Msc poster

Mandatory scientific poster that presented the work done so far, at May 24. Please note that at the time of the delivery, the SLIC superpixel segmentation method was used, and was later changed to the faster and simpler segmentation presented in chapter 3.

B.2 Links to software and documentation

The software is divided in the project AurlabCVsimulator and the ROV_objectAvoidance_StereoVision The ROV_objectAvoidance_StereoVision project is the code developed for the field test at trondheimsfjorden, it also contains other code used for development.

B.2.1 AurlabCVsimulator

The website for the AurlabCVsimulator located at:

http://larssbr.github.io/AURlabCVsimulator/

The repository for the source code is placed at:

https://github.com/larssbr/AURlabCVsimulator

Documentation for the project is placed at:

http://aurlabcvsimulator.readthedocs.io/

B.2.2 ROV_objectAvoidance_StereoVision

The repository for the source code used in the field test is placed at:

https://github.com/larssbr/ROV_objectAvoidance_StereoVision