Camera-assisted Dynamic Positioning of ROVs

Andreas Viggen Henriksen

Marine Technology
Submission date: June 2016
Supervisor: Roger Skjetne, IMT

Norwegian University of Science and Technology
Department of Marine Technology

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

MSC THESIS DESCRIPTION SHEET

Name of the candidate:	Andreas Viggen Henriksen
Field of study:	Marine control engineering
Thesis title (Norwegian):	Kamera-assistert dynamisk posisjonering av ROV
Thesis title (English):	Camera-assisted dynamic positioning of ROVs

Background
There has been an increased interest in research and development of technical solutions for underwater vehicles. We now possess the necessary knowledge and technology to perform complex underwater operations with high precision, such as seabed mapping, online underwater monitoring, subsea installations, and maintenance on pipes. There are many types of Unmanned Underwater Vehicles (UUVs) on the commercial market. Remotely Operated Vehicles (ROVs) and Autonomous Underwater Vehicles (AUVs) are most common. In most cases, underwater vehicles are expensive and not affordable to private consumers. Lately, a focus has been to develop low-cost solutions. In a student project by the candidate, he and Stian Skaalvik Sandøy assembled a fully actuated mini-ROV, named *ROV uDrone*, based on a "BlueROV Kit" from BlueRobotics. This kit came with a body frame, an electronics tube, and 6 thrusters. Other necessary electronics in the tube, power supply equipment, and relevant sensors for motion control were identified and acquired. The control system hardware and software were then designed based on the Robot Operating System (ROS), implemented, and successfully tested in MC-Lab. The ROV uDrone will be used for further development of camera-assisted dynamic positioning (CADP) control system algorithms in this master thesis.

Work description
1. Perform a background and literature review to provide information and relevant references on:
    - Low-cost, consumer-grade, inspection ROVs entering the market.
    - Use of computer vision, machine vision, etc. for positioning control of robots and vehicles.
    - Available and relevant open source graphical software packages.
    - Image processing techniques such as thresholding, object detection, feature detection and tracking, etc.

    Write a list with abbreviations and definitions of terms, explaining relevant concepts related to the literature study and project assignment.

2. Present the power system single line diagram, control system hardware layout, control system software topology, payload system, and signal communication topology for ROV uDrone.

3. Develop and implement an AutoHeading control function for the ROV uDrone, that based on a yaw measurement signal ensures tracking of a desired heading from a reference filter.

4. Explain in particular the onboard camera, the feed of camera images through the system (the RPi2 CSI) to the user, and how the camera images are available to the control system. Make a quantitative estimate of the lag in image data flow – from image acquisition to HMI, and from acquisition to the control loop. Discuss the effect of the respective lag in the control loop and the influence on the user perception.

5. Consider feature detection and tracking of simple geometries, and how to generate control error signals from movement of the feature in the camera images. Determine how this can be used to control the position and pose of the ROV, such as the longitudinal distance (object grows or shrinks), transversal (and vertical) motion (object translates), and yawing to keep the feature in view.

6. Develop CADP control algorithm(s) to control the ROV to a steady position and attitude using the feedback signals from the feature detection image processing algorithms and gyrocompass.
   - Test and present the efficiency of the CADP algorithm in a deterministic case.
   - Implement the CADP algorithm for the uDrone and test in MC-Lab.
   - Discuss pros and cons with the algorithm.

7. Consider a setup with at least 2 or more parallel laser lights on an underwater feature. Given any two laser reflection points and assuming the underwater feature is flat and normal to the laser beams:
   - Calculate the distance to the feature given the configuration of the laser lights and lens geometries.
   - Develop a "constant distance" automatic positioning mode for the ROV.
   - Implement and test in MC-Lab.

8. With more than two parallel laser lights, is it possible to control more degrees of? If so, develop algorithm, implement, and test in MC-Lab.

Guidelines
The scope of work may prove to be larger than initially anticipated. By the approval from the supervisor, described topics may be deleted or reduced in extent without consequences with regard to grading.

The candidate shall present personal contribution to the resolution of problems within the scope of work. Theories and conclusions should be based on mathematical derivations and logic reasoning identifying the various steps in the deduction.

The report shall be organized in a logical structure to give a clear exposition of background, results, assessments, and conclusions. The text should be brief and to the point, with a clear language. The report shall be written in English (preferably US) and contain the following elements: Title page, abstract, acknowledgements, thesis specification, list of symbols and acronyms, table of contents, introduction and background, problem formulations, scope, and delimitations, main body with derivations/developments and results, conclusions with recommendations for further work, references, and optional appendices. All figures, tables, and equations shall be numerated. The original contribution of the candidate and material taken from other sources shall be clearly identified. Work from other sources shall be properly acknowledged using quotations and a Harvard citation style (e.g. *natbib* Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct. Such practice is taken very seriously by the university and will have consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed upon.

The thesis shall be submitted with a printed and electronic copy to the main supervisor, each copy signed by the candidate. The final revised version of this thesis description must be included. The report must be submitted according to NTNU procedures. Computer code, pictures, videos, data series, and a PDF version of the report shall be included electronically with all submitted versions.

Start date: 15 January, 2016 **Due date:** As specified by the administration.

Supervisor: Roger Skjetne
Co-advisor(s): Mauro Candeloro

Trondheim, 31.05.2016

Roger Skjetne Digitally signed by Roger Skjetne
Date: 2016.05.31 18:51:23 +02'00'

Roger Skjetne
Supervisor

Preface

This document is concluding my master thesis within Marine Cybernetics as part of the study program Marine Technology at NTNU. The work in this theses has been carried out from January to June 2016. The project has been both challenging and exciting with many new concepts to get familiar with. It has been great to have such a practical project in order to really understand how the system behaves in reality, with all the different parts of a control system in an ROV.

Trondheim, 10. juni 2016

Andreas Viggen Henriksen

Abstract

When ROVs are operated with manual joysticks, the user will experience a lack of control because of currents and drag forces from the umbilical. This will induce undesirable oscillations and motions that are hard to compensate for manually. With the use of computer vision, it is possible to enhance control performance in unknown waters. This can be done by finding an object that stands still on the bottom, and have the ROV track a feature on that object. The ROV should then be able to lock visually on to this "target". Now the proposed Feature Tracking algorithm is telling the system where the object is along with its size. The algorithm can be used with different combinations of manual input, depending on what degrees of freedom the user wants to control himself.

The thesis also proposes a module called Range Finder, that finds the distance to the object or wall in front of the ROV. The distance estimate is obtained based on two parallel lasers that project two laser dots on the wall. The distance between these dots is then measured with use if computer vision, that counts the amount of pixels between them. This number will change depending on the distance from the wall to the ROV. A mapping is, therefore, found to give distance directly in meters instead of pixels. This module is also combined with different control modes that allow features such as automatic distance control in Wall Inspection Mode.

Both modules are tested extensively with pleasing results in Marine Cybernetics Laboratory at NTNU. The test platform is a low-cost ROV called uDrone which is one of the vehicles in MC-lab. The system is running Robotic Operating System (ROS) on a Ubuntu operating system, and OpenCV as a library for computer vision.

Sammendrag

Når fjernstyrte undervanns farkoster (ROV) styres med manuelle styrespaker, vil brukeren oppleve en mangel på kontroll grunnet strømninger og dragkrefter fra styringskabelen. Dette vil fremkalle uønskede svingninger og bevegelser som er vanskelig å kompensere for manuelt. Ved bruk av maskinsyn er det nå mulig å forbedre kjøreopplevelsen i ukjente farvann. Dette kan gjøres ved å finne et objekt som står stille på bunnen, og la ROVen spore en særegenhet på objektet. ROVen kan da bruke dette objektet som et visuelt anker. I denne oppgaven foreslås en *Feature Tracking* algoritme, som forteller systemet hvor objektet er i bildet sammen med objektets størrelse. Algoritmen kan brukes med ulike kombinasjoner av manuell styring, avhengig av hvilke frihetsgrader brukeren ønsker å kontrollere seg selv.

Avhandlingen foreslår også en modul med navn, *Range Finder*, som finner avstanden til objektet eller en vegg foran ROVen. Estimert avstand blir funnet basert på to parallelle lasere som projiserer to laserpunkter på veggen. Avstanden mellom disse punktene er så målt ved bruk av maskinsyn, som teller antall piksler mellom disse punktene i kamerabildet. Dette tallet vil endre seg avhengig av avstanden fra veggen til ROVen. Denne sammenhengen er derfor funnet for å gi avstandsestimatet direkte i meter i stedet for i piksler. Denne modulen er også kombinert med forskjellige kontrollmodi som muliggjør funksjoner som automatisk avstandsregulering i *Wall Inspection Mode*.

Begge modulene er testet grundig med tiltalende resultater i *Marine Cybernetics Laboratory* ved NTNU. Testplattformen er en lav-kost ROV kalt uDrone som er en av farkostene i MC-lab. Systemet kjører Robotic Operating System (ROS) på et Ubuntu operativsystem, og bruker OpenCV som bibliotek for maskinsyn.

Acknowledgment

I would like to thank the following people who have supported me throughout the work of my master thesis. First, I want to thank my main supervisor Roger Skjetne, and co-supervisor Mauro Candeloro for good discussions and guidance throughout the whole semester. I wish to thank Dr. Adrian Rosebrock whom I have followed and learned a lot from through his computer vision blog, Pyimageshearch.com. His literature has been a great resource for the development of this project. I would also like to thank Rune Hansen from BluEye Robotics for designing and 3D-printing a camera/laser bracket. At last, I would like to thank my fellow students in the MC-lab. We have learned a lot from each other when encountering software and hardware problems.

A.V.H.

Contents

List of Figures

List of Tables

Acronyms and Symbols

AUV	Autonomous Underwater Vehicle
CAD	Computer Aided Design
CADC	Camera-assisted Distance Control
CADP	Camera Assisted Dynamic Positioning
CMYK	Cyan Magenta Yellow Black
CSI	Camera Serial Interface
CV	Computer Vision
CPU	Central Processing Unit
DIY	Do It Yourself
DOF	Degree Of Freedom
DP	Dynamic Positioning
GPU	Graphics Processing Unit
HSV	Hue Saturation Value
I2C	Inter-Integrated Circuit
IMU	Internal Measurement Unit
MC-lab	Marine Cybernetics Laboratory at NTNU
NED	North-East-Down
OpenCV	Open Source Computer Vision
PID	Proportional Integral Derivative
PWM	Pulse-Width Modulation

RGB	Red Green Blue
RPi2	Raspberry Pi 2 single board computer
ROV	Remotely Operated Vehicle
ROS	Robotic Operating System
SIFT	Scale Invariant Feature Transform
SURF	Speeded-Up Robust Features
USB	Universal Serial Bus
η	Position of ROV relative to the basin col(x,y,z,ψ)
ν	Velocities of ROV relative to the basin col(u,v,w,r)
ψ	Heading of ROV relative to the basin
τ	Control force vector

Chapter 1

Introduction

"The future of drones is not only about flying high, but also going deep" - (Forbes 2016). As we know, there has been a big boom of aerial vehicles that has gotten cheaper, and we see more and more of them around us every day. Lately, many companies have started to develop underwater drones for the consumer market. This is now attractive because of the established market of drone buyers, who now wants to explore new waters.

“The ocean is the lifeblood of Earth, covering more than 70 percent of the planet’s surface, driving weather, regulating temperature, and ultimately supporting all living organisms.”
- (NOAA 2014)

Yet, less than 5 % of the oceans are discovered up to this date. This means there is a lot more to be seen, even in the shallow waters (0-100m). Underwater experiences have required hours of diving training and do require a lot of planning for each dive. Now with underwater drones, the ocean space will be more accessible for everyone! This will hopefully increase people’s awareness, love, and care for the ocean.

Development of low-cost underwater drones is now possible due to new, better, and cheaper hardware such as Raspberry Pi single-board computers, and Arduino microcontrollers. Also compact, powerful, thrusters from Blue Robotics Inc. has made it possible to create powerful mini-ROVs powered on a high-performance battery. Open Source software such as Robotic Operating System (ROS) and OpenCV, has enabled fast implementation and testing of new control systems. The recent research and progress of computer vision have also made it possible to implement new control functionality based on input from the ROV's video camera. Utilization of computer vision to enhance user experience in mini-ROVs will, therefore, be the main scope of this master thesis.

Background and Problem Formulation

To enable the growth of mini ROVs in the private market, a reduced complexity in controlling the vehicle will be necessary. State of the art requires some training and skills to move around freely on the bottom as the user wants to. Is it possible to set lock-in on an object of interest, say an anchor, while moving sideways around it, to ease inspection and video recording? Will it be possible to enhance the user experience by utilizing computer vision to assist the user with automatic control systems? How can this be combined with user input? This functionality will become vital for amateur operators, who wants eased control of the ROV while recording good quality video.

1.1 Low-cost ROVs Entering the Market

Several companies have decided to develop their own underwater robot for recreational use. The most promising ones will now be presented:

DeepFar MAX

DeepFar is an underwater drone developed in China; that was launched this year. It comes in two different models, one for consumers, and one for more professional use. The models are called White Shark MINI and White Shark MAX. The ROV can go down to 100 meters depth and has a battery capacity of 2 hrs. All thrusters combined provides 600 Watts of power. The vehicle is tethered and streams 2 MP video to the user in the MINI model, while it has full HD for the MAX model. The total weight is 12 kilos. (Forbes 2016, DeepFar 2016)

Figure 1.1: DeepFar MAX ROV (Source: deepinfar.com).

Deep Trekker

Next up is the DTG2 from Deep Trekker. This is a low-cost ROV that is aiming more for the small business market. The ROV is well suited for inspection of tanks and fish farms. The vehicle only has two thrusters, because it has an internal weight shifting system that allows the thrusters to be pointed in any pitch direction in the vertical plane. The starting kit can go down to 50 meters with optical fiber for video transmission. The weight is 8.5 kilos with a total battery capacity of 4-8 hours, depending on usage. Max speed is 1.25 m/s (DeepTrekker 2016).

Figure 1.2: Deep Trekker DTG2 (Source: deeptrekker.com).

OpenROV Trident

OpenROV is about to launch their newest ROV called Trident. With its 2.9 kilos, it is one of the lightest ROVs out there. However, it only has three times 12 W of thrust, but the combination is still powerful enough to lift the vehicle out of the water as shown in figure 1.3. The ROV is rated for 100 m depth, but the start-kit comes with 25 meters of cable (not fiber). Trident has a battery life of 3 hours, with promised maximum speed to 2.2 m/s. Shipping is starting in November 2016 (OpenROV 2016).

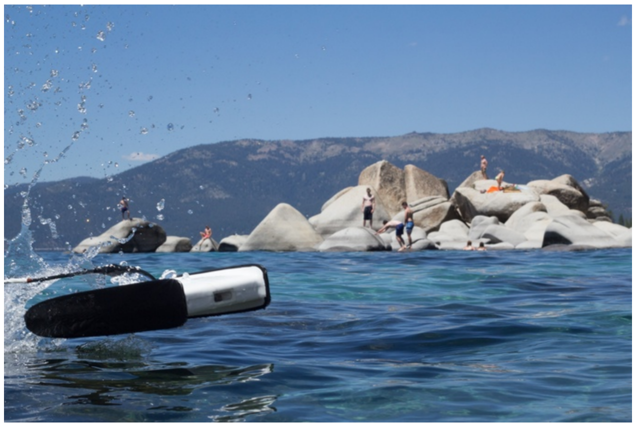


Figure 1.3: OpenROV Trident (Source: OpenROV.com).

FiFish

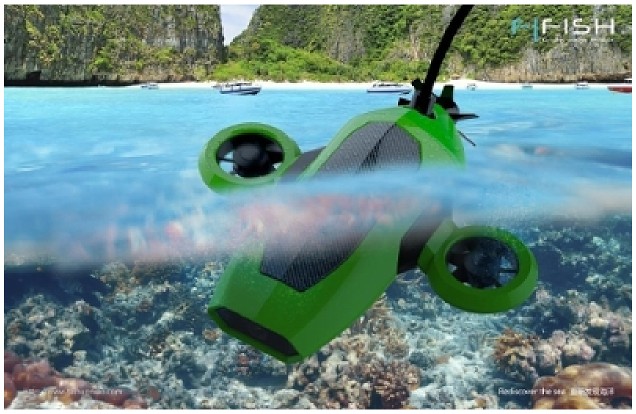


Figure 1.4: FiFish Atlantis (Source: prnewswire.com).

The FiFish was presented for the first time on a show in Las Vegas January 2016. They claim to be the first Smart ROV in the world for the mass consumer market. The ROV has a battery life of 2 hours, and can dive down to 100 meters. Their main goal is to get everybody to rediscover the ocean! The ROV with a carbon hull is shown in figure 1.4 (FiFish 2016). Both FiFish and DeepFar are developed by Chinese companies which can be taken as evidence for a globally growing low-cost ROV market.

1.2 Use of Computer Vision for Positioning Control of Robots

Karras & Kyriakopoulos (2008) and Karras et al. (2006) describes a position-based visual servo control scheme for an ROV based on what they call a Laser Vision System (LVS). The LVS is used to regulate the angle between the ROV and the wall. The LVS is also used for collision avoidance when there is a target on a wall that the ROV is set to approach. The target on the wall is tracked with computer vision algorithms using a method called "Snakes". A path planning strategy is based on Artificial Potential Field (APF) method; that is responsible for controlling the ROV to the desired position. The total system was extensively tested on a VideoRay mini ROV. The distance measurements vary with about 6 % from the actual distance.

In Karras et al. (2010), a semi-automatic control system is proposed to guide the human operator in holding the object of interest inside the cameras field of view. Estimation of the vehicle's states are achieved by an Unscented Kalman Filter on the LSV together with an Inertial Measurement Unit. The system is tested on an underacted 3 DOF ROV in a test tank. The automatic controller for depth is implemented as a PD controller, while the remaining two DOFs, surge, and yaw, are regulated by a backstepping controller with stability guarantees. The paper concludes that the under-actuated dynamics are input to state stable, hence robust and bounded by the user input. The object was also successfully held in the camera's field of view, while the ROV was maneuvering in a test pattern.

Heshmati-Alamdari et al. (2014) presents a novel Vision-based Nonlinear Model Predictive Control (NMPC) for underactuated ROVs. The self-triggered NMPC, rather than periodically triggered NMPC, shall reduce the required amount of data from the computer vision to reduce the required computational power and energy consumption. Also in this paper, the objective is to keep a target inside the vision system's field of view with use of feature tracking and pose estimation by the geometry of the tracked object.

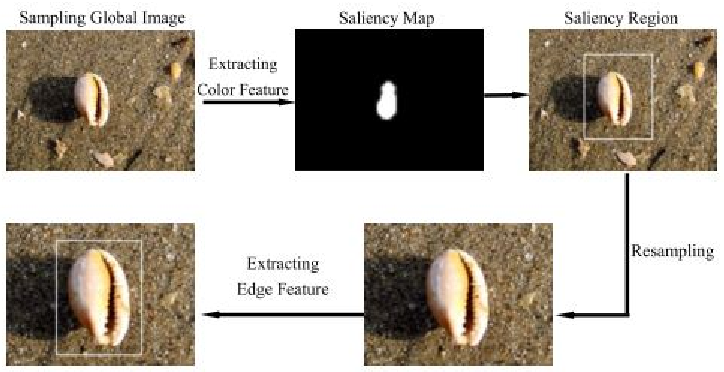


Figure 1.5: Flow chart for attention based tracking (Shi et al. 2010).

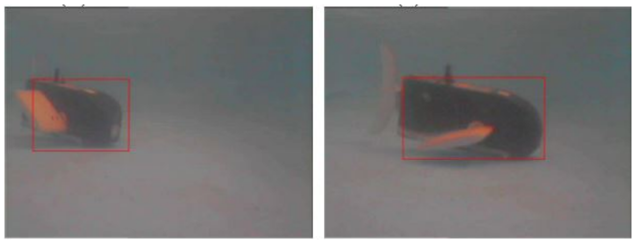


Figure 1.6: Robotic fish tracking each other (Hu et al. 2009).

Shi et al. (2010) presents a method that simulates humans vision attention when detecting objects. An efficient algorithm is proposed to crop areas of interest before running the filtering and detection functions at a higher resolution. The proposed method used 70 ms, versus 96 ms without the algorithm. The signal flow of the algorithm is presented in figure 1.5.

In (Narimani et al. 2009), computer vision is used to estimate the angle between the AUV and a pipeline on the seabed. This is used for autonomous following and inspection of the pipeline. Edge detection is used to find the angle of the pipeline. An adaptive sliding mode controller is used to regulate the heading to keep the pipe in the center of the image in vertical position. The system is simulated with recorded video from an ROV as input.

Computer vision is also used for an autonomous fish that follows another manually controlled robotic fish. Hu et al. (2009) are using an adaptive mean shift (Camshift) algorithm to lock on to the fish it is trying to follow. A fuzzy controller was designed for motion regulation of the fins. The fish following was successfully tested with experiments. The fish tracking is shown in figure 1.6

(Raabe et al. 2014) propose an indoor positioning system based on a Playstation 3 Eye cam-

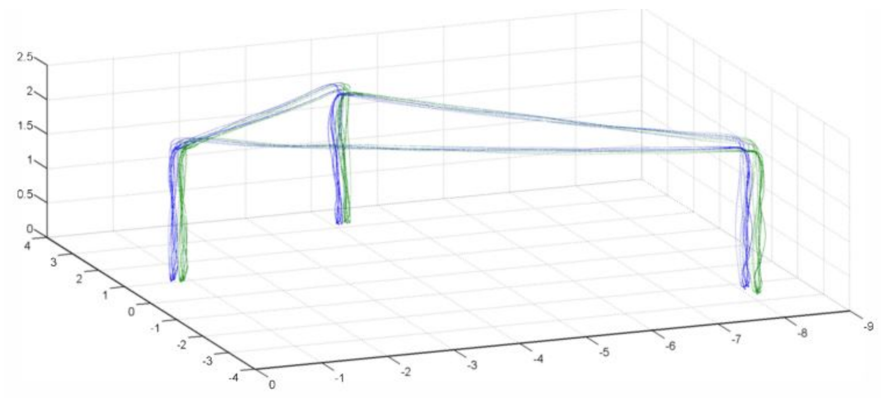


Figure 1.7: Results of indoor navigation based on computer vision (Raabe et al. 2014).

era mounted on a hexacopter. This is done without use of any external positioning systems such as GPS. It is only using three tracked targets on the floor, together with the internal IMU. The algorithm does not have to see all targets at the same time, because it remembers where the target was observed last time. This is done by using PTAM, which is a type of Simultaneous localization and mapping (SLAM) algorithm. The camera image is also divided into a grid to smaller areas with an independent threshold for filtering. This is to make tracking more robust and stable in environments with large color contrasts. The results are presented in figure 1.7, with "MoCap" position measurements in blue, and computer vision estimates in green.

1.2.1 Open Source Graphical Software Libraries

Many different software libraries are presented in Rosebrock (2014). The book has chosen Python as the primary programming language because it is open source and an easy language to get started with. To perform computer vision with Python, we need a good matrix library that handles heavy matrix calculations efficiently. This task is solved by the *NumPy* library for Python. *Matplotlib* is another library that is suitable for plotting signals in Python (similar to plotting in MatLab).

Then we have *SimpleCV*, which is computer vision made easy. Their goal is to create a library that enables the programmer to get started quickly with a lean introduction to the basic concepts of computer vision. It gets you started quickly, but at the cost of the raw, powerful techniques that sometimes are needed. Rosebrock also presents the less known library called *Mahotas*, which provides much of the same functionality as *SimpleCV* and *OpenCV*. *OpenCV* is by far the most used computer vision library with great speed and functionality. With its huge user community, it is definitely the best way to go if we want to be able to get help from others on the Internet. *OpenCV* was first presented in 1999, and first compatible with Python in 2009. The library itself is written in C++, but now Python bindings in the installation allow Python to work just as fast. So as long as we do not write heavy for-loops in Python, we will have the same speed potential in Python as in C++. *OpenCV* does have a steeper learning curve, but provides faster processing, more embedded functions, and has a great documentation page.

1.2.2 Image Processing Techniques

Thresholding

A basic tool in computer vision is to filter out pixels based on their color value. If we then set a certain limit, we can then set everything above that limit to 255 (white), and everything below to 0 (black). We can now make a masked image that only displays the pixels above the limit as a white figure. Thresholding is often used to separate out important areas of an image, like an object from its background (OpenCV 2013*c*).

Object detection

If the computer vision system is searching for an object that is known prior to the test, it is possible to compare key-points to find a matching object. There are multiple techniques for this task. Scale Invariant Feature Transform (SIFT) is one method of extracting key points and computing descriptors (Lowe 2004). Speeded-Up Robust Features (SURF) is a more intelligent key-point detector that also allows us to compare key-points in two different images. This method is used for camera calibration of stereo vision and object recognition (Bay et al. 2006).

Feature Detection and Tracking

A feature is simply information in the picture that is possible to extract via computer vision. A feature can be an edge, a color, shape, or a specific pattern, that is unique for an object of interest. It is like playing puzzles, where we have to determine the location of each piece by its color and unique feature relative to the pieces around it. Once the desired feature is filtered out

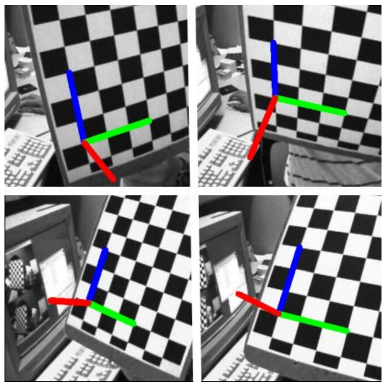


Figure 1.8: Pose estimation with `calib3d()` function in OpenCV (Source: Opencv.org).

of the image with thresholding, it is possible to track the features with *Meanshift* and *Camshift*. *Harris Corner Detector* and *Shi-Tomasi Corner Detector* are other methods for finding corners in images. Corners are often good features to track in air. In water, it will be more difficult since corners and contrasts usually are not as sharp due to growth and particles in the water. (OpenCV 2013*a*). A feature can also be tracked if the center point of each contour is stored from one image to the next. This also allows us to find the derivatives in the images to see how fast the feature is moving. Background subtraction can also be used to detect changes in a camera, that stands still, with a moving object in front. The algorithm simply subtracts one image from the previous one, which leaves the background with no change as 0, and random numbers on the pixels that have changed. These numbers can then be set to 255 by thresholding, and later tracked from image to image.

A technique for detecting the pose of the ROV relative to a known object is found in OpenCV (2013*b*). The tutorial proposes a method for finding the pose of a checkerboard in front of a calibrated camera. This method could be used for the ROV to estimate its pose relative to a known square. However, most of the time we do not have a perfect square at the sea bottom, so it will be difficult to use this method in the ocean. The method is illustrated in figure 1.8 and might be harder to implement under water. It is also unknown if this function will be able to run real-time to control the ROV.

1.3 What Remains to be Done?

The use of computer vision has been tested and verified for an ROV in Karras & Kyriakopoulos (2008), Karras et al. (2006), and Karras et al. (2010). However, it is not addressed how this can be used together with manual control for specific tasks. The proposed modules will also be implemented and tested on a low-cost ROV that has a shorter way to realization in the consumer market. The proposed Feature Tracking algorithm should able to track any object without any information about the object prior to the dive.

1.4 Objectives

The main objectives of this Master's project are

1. Develop camera based positioning control modes for mini-ROV uDrone.
2. Install laser lights and cameras needed for feature-tracking and laser-tracking.
3. Develop distance estimate based on two parallel lasers that are tracked in the video stream.
4. Implement an algorithm to filter video stream for feature tracking.
5. Perform thruster tests to enhance the performance of the thrustallocation.
6. Develop and implement an Auto-heading controller that ensures tracking of a reference.
7. Develop and implement Auto-depth controller with a suitable guidance system.
8. Test implemented control modes in Marine Cybernetics Laboratory (MC-lab) with Qualisys positioning system.

1.5 Limitations

In this thesis the ROV, hardware, and the cameras were chosen before the feature algorithm was developed; therefore, some computational limitations was experienced during development. The light conditions in MC-lab were limiting the range of colors that was feasible for feature tracking. The positioning system in MC-lab called Qualisys did also have its limitations on where it was possible to operate in the pool. On the other hand, the software framework called ROS has enabled fast development and rapid testing with great flexibility and robustness.

1.6 Abbreviations and Definitions

- **BGR** short for Blue Green Red, used in *OpenCV* instead of RGB (different order only).
- **Contour** is defined as a connected shape of white pixels in a binary image.
- **Drone** a term borrowed from the areal drone market. This report will use ROV, which is more precise.
- **Feature** is property of the object. Its color, surface pattern, shape, key point, etc.
- **Feature Tracking** is the proposed module to track a feature (color) on the object.
- **HSV** short for Hue Saturation Value, utilized for tuning of feature color filtering.
- **Object** is the tracked target of interest in the basin. Could be an anchor in the ocean.
- **Range Finder** is the proposed module for distance estimation based on lasers.
- **ROV** short for Remotely Operated Vehicle
- **Thresholding** is filtering all image matrix values within a certain range.

1.7 Approach

In order to find what control modes that are possible to use, a study on computer vision libraries will be performed to see what kind of information that is available from the filtering functions. To install the laser lights in a proper way, a mounting bracket for lasers and cameras shall be designed in CAD and 3D-printed to customize the perfect fit for the components. Being able to track the laser in the video stream requires a robust methodology to filter out only the lasers, with tunable parameters that can be tuned on-line in case of changing light conditions, or reflection of the material the laser hits. When lasers are properly tracked, it will be desirable to find the relationship between the number of pixels between the two laser dots and the distance from the ROV to the wall in front of it.

There are many ways to filter out objects in a video. To find the most feasible method, it will be necessary to find a method that easily can be applied to new features, with live-tuning of parameters. To start with, bright colors with large contrast to the rest of the environment are chosen for the features, even though this might not always be the case in the ocean. The feature is also going to be set up, so it is floating in the middle of the depth of the pool. This is, so it does not interfere with the reflection from the surface, and at the same time stay in place.

To enhance the control and performance of the control system, the ROV shall be mounted to the towing carriage with sensors to measure forces and torque at the RPM range of interest. From these data, it will be possible to make a proper mapping between PWM signals and forces. These sensors are not waterproof so a special ROV rig must be assembled and mounted properly to the ROV and towing carriage. The auto-heading, -depth, and -distance controllers must be able to run simultaneously with live tunable parameters. Simple PID controllers will be used to solve the different control tasks and tested in HIL mode on the system. All software will be written in C++ or Python to keep required computational power at a minimum. These programming languages do neither have a strict license policy; that can be of importance in the further development of this system.

Due to easy access to MC-lab; testing of subsystems will be performed as soon as they are implemented. This will give a better understanding of the possibilities that evolves during development. When all control modes are implemented and ready for final testing, a test scheme will be executed in the MC-lab with Qualisys to measure the actual movement as a performance indicator.

1.8 Structure of the Thesis

The rest of the report is organized as follows. Chapter 2 gives an introduction to the experimental platform used for testing the proposed control modes. Chapter 3 pretenses the thrustallocation force mapping for ROV uDrone. In Chapter 4 an introduction to computer vision will be presented followed by techniques that are used to solve the problem. Chapter 5 will present The camera-assisted positioning based on feature tracking with available control modes and how the problem was solved. Chapter 6 addresses how lasers can be used for distance estimation for an Auto-distance controller. Results from experiments in MC-Lab will be presented and discussed at the end of both chapter 5 and 6. At last, a summary and recommendations further work will be given in Chapter 7.

Chapter 2

Experimental Platform

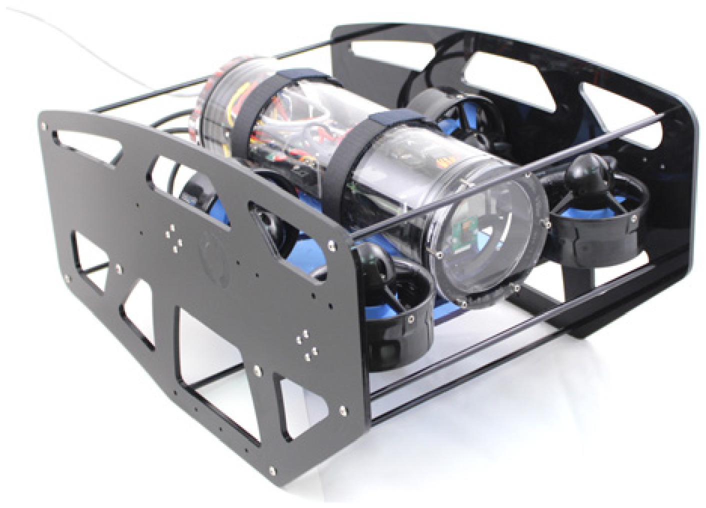


Figure 2.1: BlueROV kit from Blue Robotics Inc. (Source: BlueRobotics.com).

The experimental platform used for testing the proposed Feature Tracking and Range Finder modules is an ROV that was built by Stian Skaalvik Sandøy and myself during a previous student project. The name ROV uDrone was then given to the vehicle. Main relevant specifications will now be presented, while the full description is available in Henriksen & Sandøy (2015).

2.1 General Specifications for ROV uDrone

The ROV uDrone is based on a DIY kit from Blue Robotics called BlueROV shown in figure 2.1. This kit came with six thrusters, a watertight tube, and a body frame that holds everything. The necessary electronics inside the tube was identified and ordered before everything was assembled. The software system was chosen to be Robotic Operating System (ROS), and basic control functionality was implemented. The ROV was then tested extensively in MC-lab. The main specifications for the ROV are presented in table 2.1.

Table 2.1: Specifications for the BlueROV kit (Henriksen & Sandøy 2015).

Physical dimensions	
Length	483 mm
Width	330 mm
Weight (no electronics)	3.73 kg
Watertight Enclosure inner diameter	102 mm
Watertight Enclosure inner length	298 mm
Performance	
Total Forward Thrust	100 N
Total Vertical Thrust	150 N
Total Side Thrust	50 N
Maximum Depth Rating (tested)	100 m
Electrical	
Operating Voltage	12-16 volts

2.2 Hardware Topology

The electronics inside ROV uDrone consists mainly of a Raspberry Pi 2 (RPi2) and an Arduino Mega. RPi2 is a mini-computer running Ubuntu 14.04 with ROS Indigo installed. Arduino is a microcontroller that handles all input/output communication with the connected sensors and actuators. The thrusters are controlled by motor controllers that receive PWM signals from the Arduino. For depth measurements, a pressure sensor, MS5837, is integrated with the IMU device, and connected with I2C to the Arduino Mega. The cameras are connected with USB and

CSI directly to the RPi2. Communications between the ROV and the topside computer is done via Ethernet cable to a local network. Qualisys positioning system and tablets are also connected to the same network. Any computer running ROS in the network can now subscribe and/or advertise to any message to or from the ROV. This setup gives a lot of flexibility and possibilities when developing. The hardware communication flow is illustrated in figure 2.2.

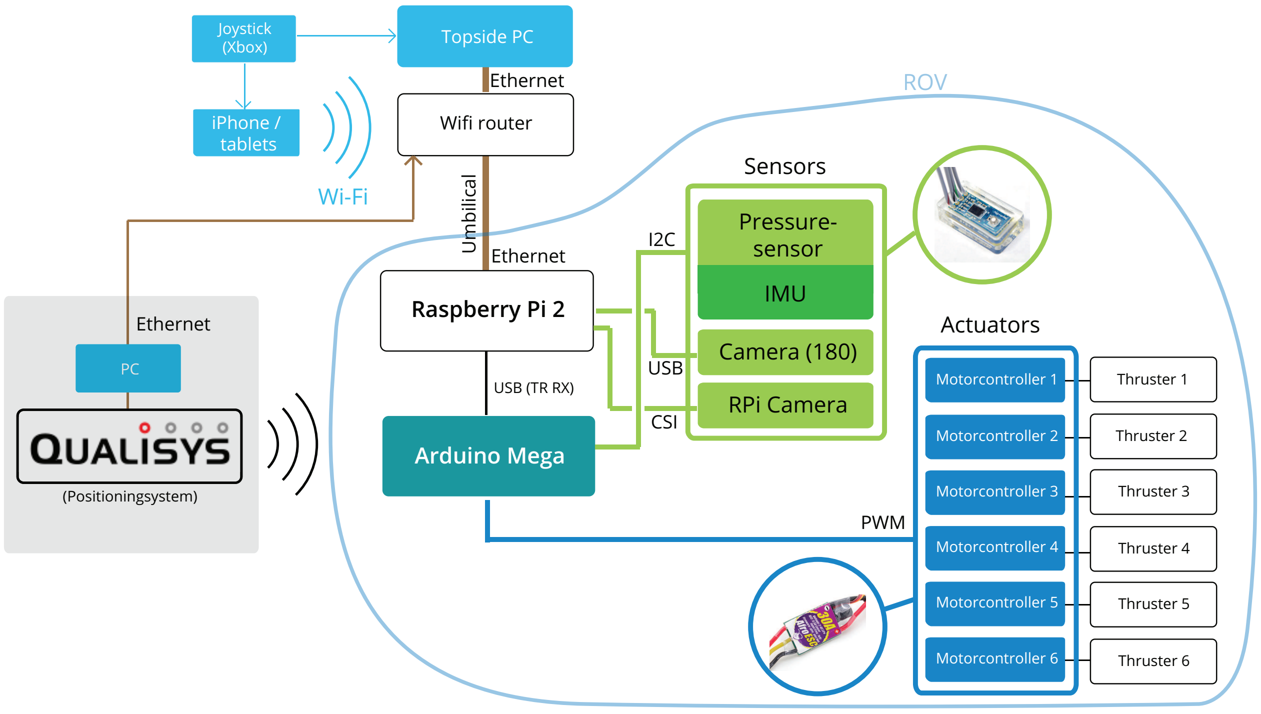


Figure 2.2: Hardware and communication configuration for the ROV uDrone (Source: Henriksen & Sandøy (2015)).

2.3 Power Flow in ROV uDrone

ROV uDrone is powered from in-house batteries with a total capacity of 10 000 mAh. This is enough for testing up to 4 hours, at low speeds. The battery is connected to a distribution board that distributes 14.8 V to the six thrusters, and 5 V to the remaining electronics. The IMU is powered from a 3.3 V port on the Arduino, while the cameras are powered directly from the RPi2. The power flow inside ROV uDrone is presented in figure 2.3.

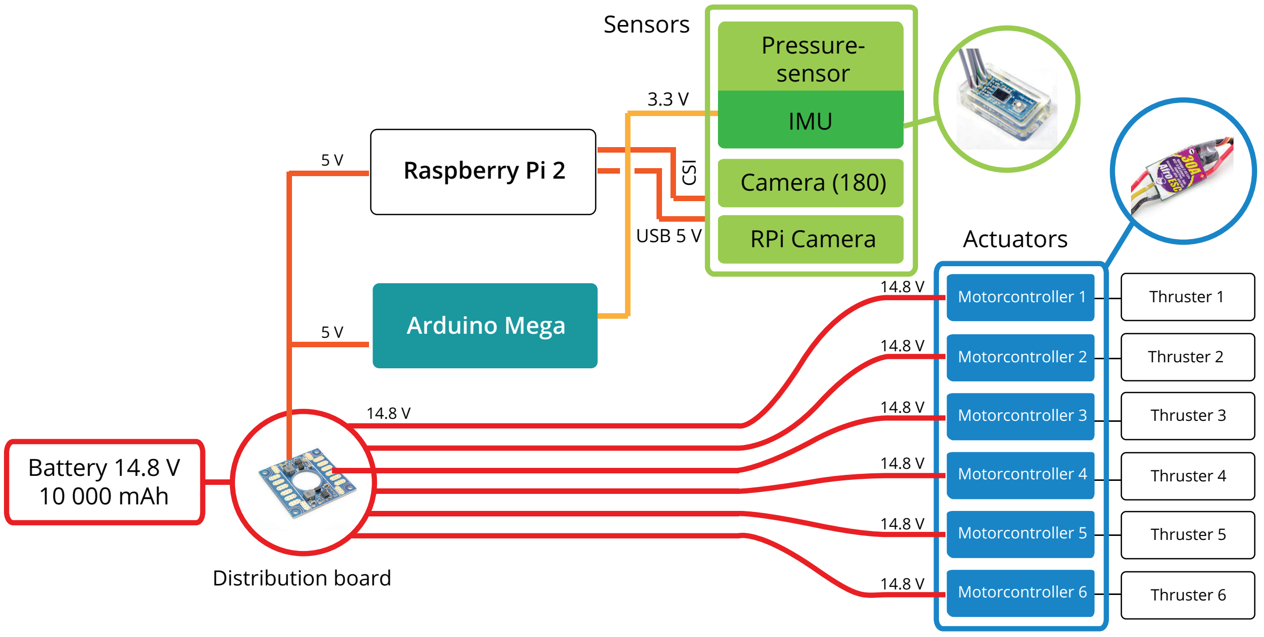


Figure 2.3: Power flow in the ROV uDrone (Source: Henriksen & Sandøy (2015)).

2.4 Payload Systems

Many auxiliary systems can be connected to the ROV and easily integrated to ROS, where most hardware drivers are already written. However, for this project, only two cameras and four lasers are implemented. The cameras, lasers, and assembly, will now be presented.

2.4.1 Fisheye Camera

For the video stream, an 180-degree fisheye lens web-camera was chosen due to its small size and high-quality video. The camera is connected and powered directly from the RPi2 via USB. Camera specifications are presented in table 2.2, and shown in figure 2.4. This camera is the one that is going to be used for the feature tracking module.

2.4.2 Raspberry Pi Camera

A second camera is also mounted inside the ROV uDrone. This camera has a narrower field of view and is directly connected to the onboard GPU on RPi2 through what is called Camera Serial Interface (CSI) by the Raspberry Pi Foundation. This connection allows faster video processing without draining all the capacity of the RPi2 CPU. However, at some point, the video stream has

Figure 2.4: Fish-eye camera inside ROV uDrone (Source: Henriksen & Sandøy (2015)).

Table 2.2: Specifications for the fish-eye video camera.

Specifications for USBFHDO1M
- Connector: USB 2.0 web camera
- Field of view: 180 degrees
- Resulution: 2592x1944
- FPS: 60 in 1280x720 resolution
- FPS: 30 in 1920x1080 resolution
- Sensor: CMOS 1080P

to be converted to a matrix that is readable for OpenCV. This process is computationally heavy for the CPU even though the streaming from the camera is done by the GPU. So the gain of using CSI might be gone when the video is imported to OpenCV. The RPi2 Camera will be used mainly for the Range Finder module that will be introduced later. The camera is shown in figure 2.5, and the specifications are given in table 2.3.

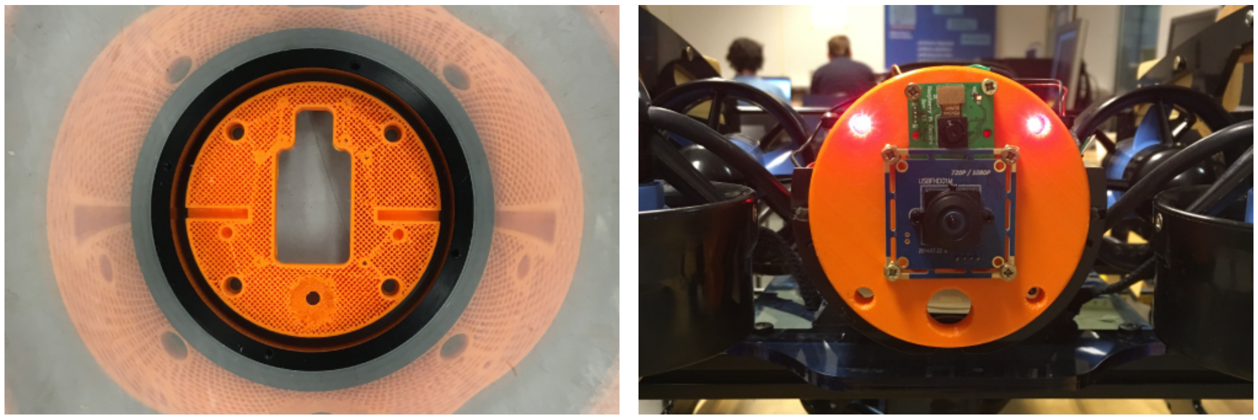
Table 2.3: Specifications for the RPi Camera (Source: Raspberrypi.org).

Specifications for Raspberry Pi Camera
- Connector: CSI
- Field of view: 54 degrees
- Resolution: 2592x1944
- FPS: 42 in 1296x972 resolution
- FPS: 30 in 1920x1080 resolution
- Sensor: 5-mega-pixel OmniVision OV5647

Figure 2.5: Raspberry Pi Camera inside ROV uDrone (Source: sparkfun.com).

2.4.3 Camera Mounting Bracket with Lasers

Since two different cameras were inserted at the same time, it was necessary to mount them properly. Four lasers were also needed to test the Range Finder and had to be installed parallel at a fixed angle and distance. It was critical to have fixed relationship between the camera and the lasers before performing the distance calibration. The solution was to create a 3D-model and print a bracket that everything could be mounted to. Once the cameras were mounted and tested, the lasers were glued at the correct angle by aiming at a target on the wall. The mounted bracket, cameras, and lasers, are shown in figure 2.6.



(a) (b)

Figure 2.6: The camera bracket in the tube (a), and (b) shows the cameras mounted.

2.5 Software Topology

The hardware is mainly divided into three layers. The first layer is the topside computer with user interface and monitoring. This layer is running on a stationary computer with Linux Ubuntu 14.04 LTS, ROS Indigo, and OpenCV 3.0 installed. The next layer is the single-board computer inside the ROV. In this layer, a RPi2 is running with same Ubuntu version, ROS, and OpenCV 2.4 installed. The lowest level is running Arduino programming language; that is very close to C++. This is where all the drivers for connected hardware are written. Most of these scripts are often given from the hardware vendor. The software is summarized in table 2.4.

Figure 2.7 presents the signal communication within the ROS system. The flow starts in the Joystick block where the user selects control mode. The yellow lines then take care of switching between the three control nodes, *Feature Tracking*, *Range Finder*, and *Direct Motion*. Usually, all control nodes are compiled on the RPi2 in the ROV. However, since the *Feature Tracking* node requires more computational power, it had to be moved to the topside computer. The *Feature Tracking* node can also run on the RPi2, but then at a much lower resolution. Once control mode is selected, only one of the control nodes will be active, and will start to publish force commands. This command is then converted to a thrust command by the *Thrust Allocation* node. *ROS serial* includes the Arduino to the ROS system so signals can be exchanged across the platforms. The thrust command is then converted to a PWM signal, which is the reference to the motor controller, that ensures the desired thrust from the thruster.

The video cameras are available to the system in separate ways. The fisheye camera is connected via USB and is read by the *Web Video Server* node. The raw video stream is then converted to an MJPEG stream that is published to port 8080 on the RPi2 (ROS 2015). The camera stream can then be accessed from any web browser on the same network with the following URL:

```
http://192.168.0.107:8080/stream?topic=/usb_cam/image_raw
```

where 192.168.0.107 is the IP of the RPi2. In OpenCV the camera stream can be read by:

```
image_matrix = cv2.VideoCapture("http://192.168.0.107:8080/...
        ...stream?topic=/usb_cam/image_raw&quality=20?x.mjpeg")
```

and displayed with the `imshow()` function:

```
cv2.imshow("Tracking", frame)
cv2.imshow("Binary", mask)
```

where *Tracking* is the image from camera, and *Binary* is the thresholded image. `frame` and `mask` are the corresponding image matrices

The RPi Camera is connected via RPi CSI and is available to OpenCV with a Python library called PiCamera. The code for accessing one image frame from this stream is slightly more complex. The code snippet to grab a frame is shown below:

```
camera = PiCamera()
camera.resolution = (320, 240)
camera.framerate = 30
camera.crop = (0.15, 0.15, 0.85, 0.85)
raw = PiRGBArray(camera, size=(320,240))
for frame in camera.capture_continuous(raw,format="bgr",use_video_port=True):
... #end
```

PiCamera library is first selected, before resolution and frame rate. A new matrix is then pre-allocated in RBG format with the same dimensions as the incoming camera feed. At last a `for` loop is then filling in the `raw` matrix with the current image. The remaining image processing is then performed inside this `for` loop. The `frame` matrix is now available for use in the *OpenCV* environment, in the same format (BGR) as in the previous example.

Once the video streams are imported to OpenCV in Feature Tracking or Range Finder, a new control force is computed and sent to the thrust allocation again. Range Finder is also using heading measurements from Qualisys to regulate on, shown in the top right corner. An auto-generated map for the communication in ROS is found in appendix A.

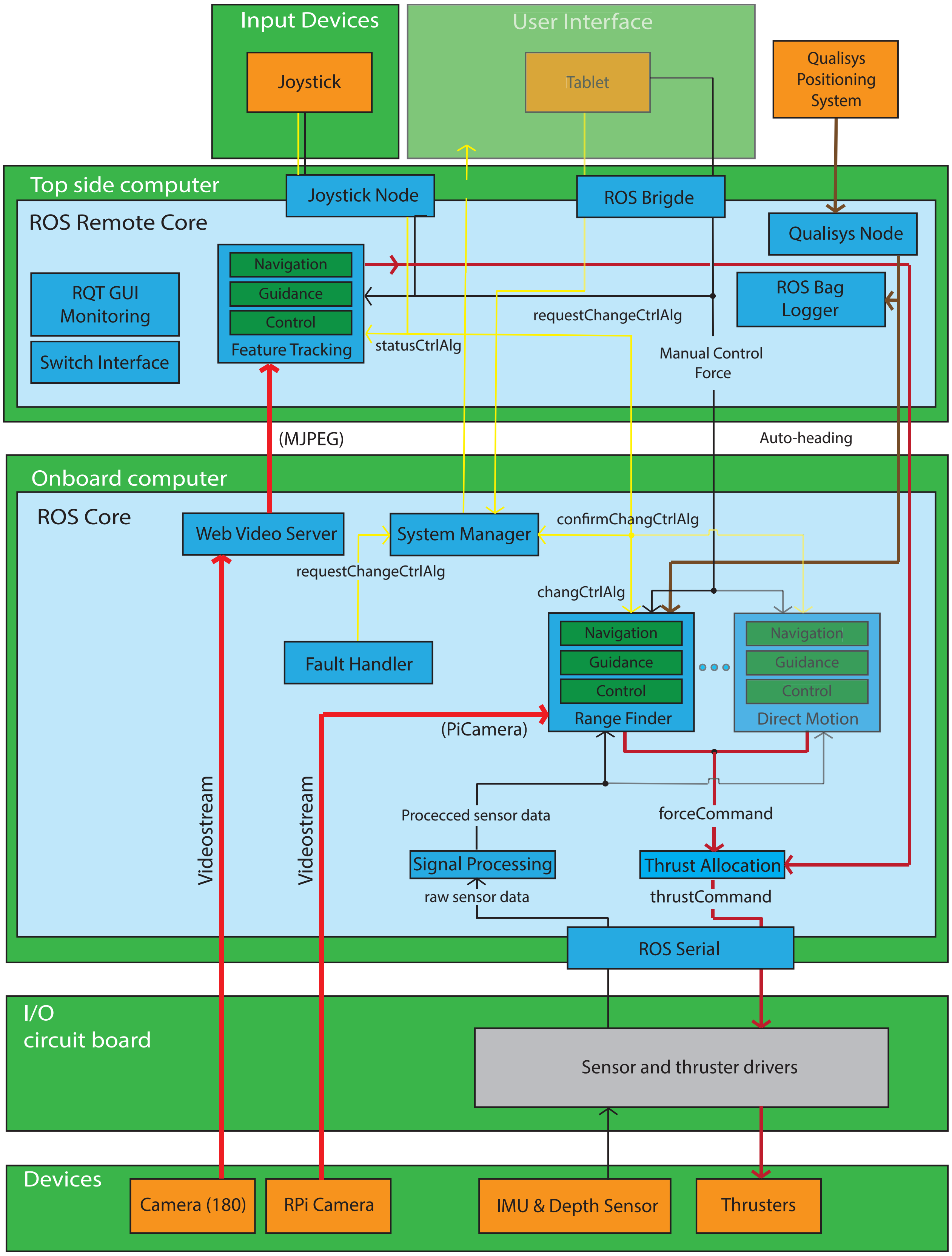


Figure 2.7: Software topology in ROV uDrone (Modified from: Henriksen & Sandøy (2015)).

Table 2.4: OS and framework on the hardware components.

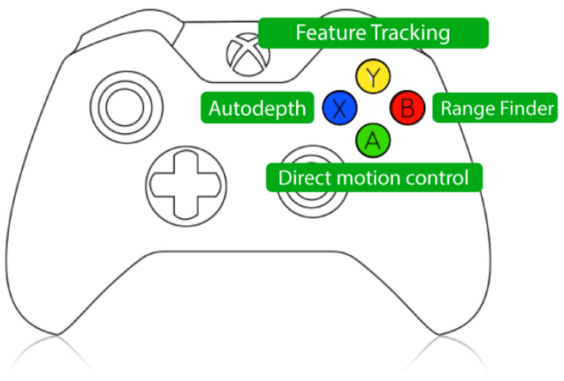
Topside computer	
OS	Ubuntu 14.04 LTS
Framework	ROS
Program Language	C++/Python
Computer Vision Library	Python in OpenCV
Raspberry Pi 2 on ROV	
OS	Ubuntu 14.04 LTS
Framework	ROS
Program Language	C++/Python
Computer Vision Library	Python in OpenCV
Arduino on ROV	
OS	Arduino
Program Language	C/C++

2.6 Control Modes Available

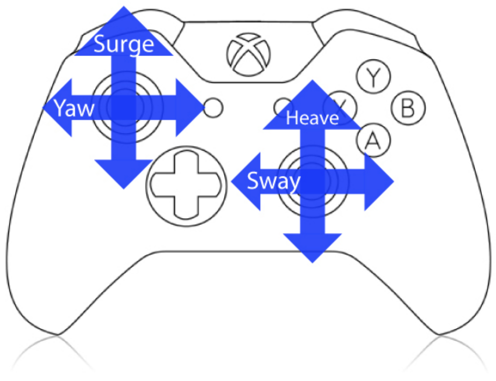
Several control modes are already implemented in ROV uDrone. These will be used in combination with the proposed modules Range Finder and Feature Tracking. The available control modes in ROV uDrone is presented in table 2.5. The proposed control modes, Feature Tracking and Range Finder, are accessible from the x-box controller as shown in figure 2.8. Manual control of each DOF is also presented in the same figure.

Table 2.5: Control Modes in ROV uDrone.

Control Mode	Description
Direct Thruster Control	The user is controlling each actuator manually
Direct Motion Control	The user is commanding a motion relative to the vehicle, i.e, surge, sway, heave, yaw motions.
Auto Depth Control	The user defines a desired depth, and can move in x-y-plane
Auto Heading Control	The user defines a desired heading relative to basin-frame
Full DP	The user commands a desired position in basin-frame coordinates and a desired depth. The controller takes input from Qualisys positioning system.



(a)



(b)

Figure 2.8: Control modes in (a), manual control in (b).

Chapter 3

ROV Model and Thrust Allocation

The reference frame for the ROV uDrone is chosen to follow the convention form of Fossen (2011), consisting of Basin frame and body reference frame, defined as shown in figure 3.1.

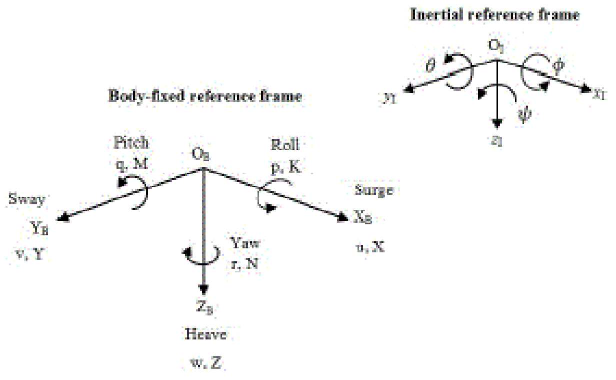


Figure 3.1: NED and Body frames with orientations and directions.

Even though the uDrone has the ability to control all 6 DOF's, the position vector has been simplified to 4 DOFs. This is because the roll- and pitch motions are assumed to be passively stable. The basin frame state vector then becomes: $\eta = \text{col}(x, y, z, \psi)$, with the corresponding body-fixed velocity vector $\nu = \text{col}(u, v, w, r)$. The ROV kinematic model is then

$$\dot{\eta} = J(\psi)\nu. \tag{3.1}$$

The kinetic control model is given as

$$\mathbf{M}\dot{\nu} + \mathbf{C}(\nu)\nu + \mathbf{D}(\nu)\nu + \mathbf{g}(\eta) = \tau + \mathbf{J}(\psi)^{-1} b(t), \tag{3.2}$$

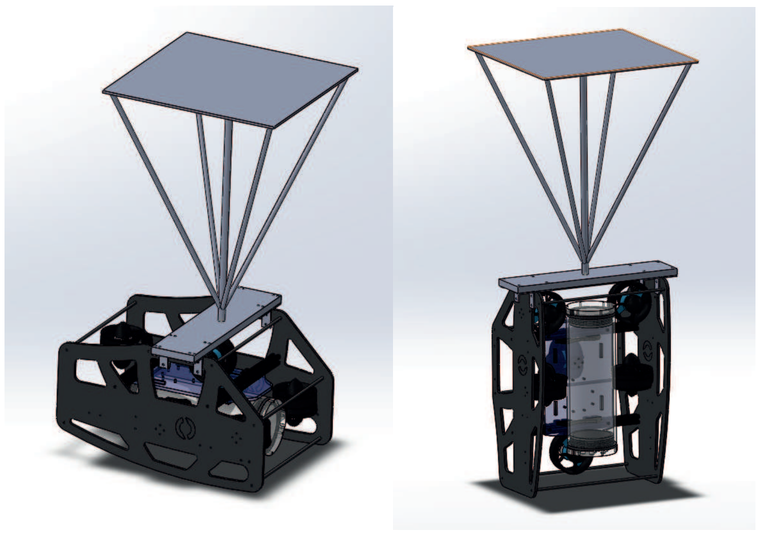
where, in particular, $\mathbf{M} = \mathbf{M}_{rb} + \mathbf{M}_a$ is the system inertia matrix, $\mathbf{g}(\eta)$ is the restoring vector, and $\mathbf{b}(\mathbf{t})$ is an external slowly varying bias vector. Equation 3.1 and 3.2 are complete processes models. However, in this project a focus has been to decouple motions as much as possible, allowing to control different DOFs based on different sensor input for each DOF. If we isolate each DOF independently, a state-space model for each DOF can be made. In equation 3.3 a model for the heave dynamics are presented.

$$\begin{aligned} \begin{bmatrix} \dot{z} \\ \dot{w} \end{bmatrix} &= \underbrace{\begin{bmatrix} 0 & 1 \\ 0 & -\frac{d}{m} \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} z \\ w \end{bmatrix} + \underbrace{\begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix}}_{\mathbf{B}} \tau_z \\ \mathbf{y} &= \underbrace{\begin{bmatrix} 1 & 0 \end{bmatrix}}_{\mathbf{C}} \begin{bmatrix} z \\ w \end{bmatrix} \end{aligned} \tag{3.3}$$

We see that z position is measured from the depth sensor, and the control force, τ_z, will manipulate the motion. The bias is neglected and compensated for later by the integrator in the controller. The same procedure can be done for surge, sway, and yaw, assuming the heading is close to zero.

3.1 Thruster Force Testing

When isolating each DOF as shown in equation 3.3, it is critical to have a good thrust allocation, so the controllers are not working against each other. To enhance the performance of the thrustallocation, some thruster tests were executed by mounting the ROV to the towing carriage. The ROV was mounted in two different configurations to measure force and torque in each DOF. The configurations are shown in figure 3.2a, assembled in figure 3.2b, and mounted in figure 3.3a. The two force sensors can be seen between the plates on figure 3.3b.


(a) Mounting configurations for thruster testing.

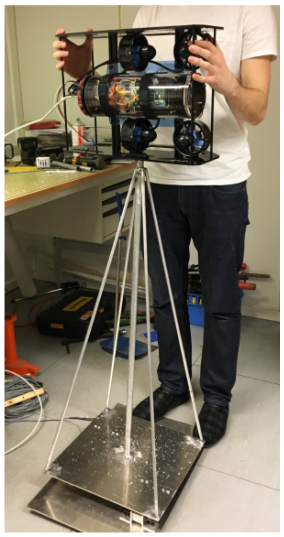
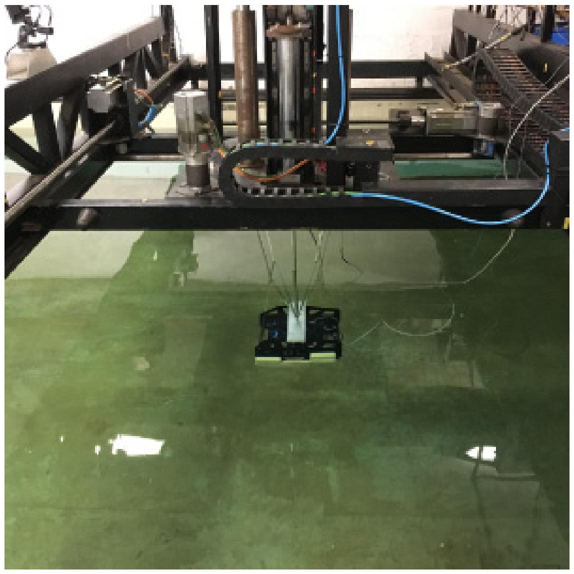

(b) Assembled test rig.

Figure 3.2: Testing configurations.


(a) ROV mounted under water.

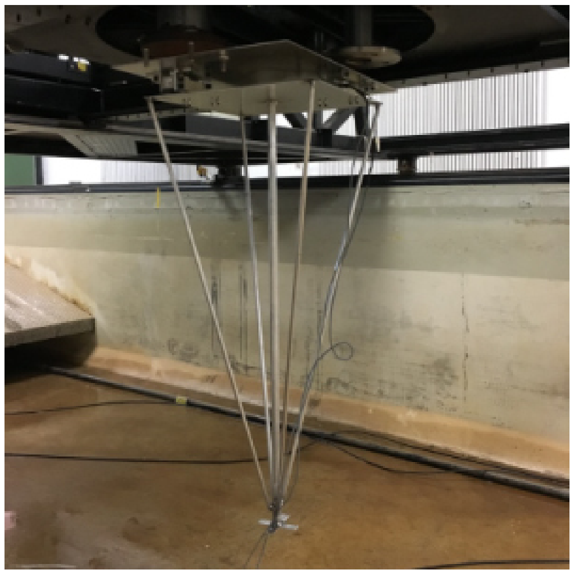

(b) Rigg mounted when pool was empty.

Figure 3.3: Testing Thrusters in MC-lab

A test script was made to run stepwise throughout the relevant PWM range. The result was a regression line that gives the relationship between PWM and Force for each thruster. The fitted curve for the surge thruster can be seen in figure 3.4. The full range of the PWM signal is from 1100 to 1900, where 1100 is full reverse, 1500 is zero, and 1900 is full forward. So the range of interest is set to +- 150 μs around 1500, which corresponds to 1350 - 1650 μs. In other words, 150/400 = 37.5% of full PWM signal, and 12/50 = 24% of full force [N]. From previous testing, it was discovered that 10 [N] is more than enough force for dynamic positioning in a pool.

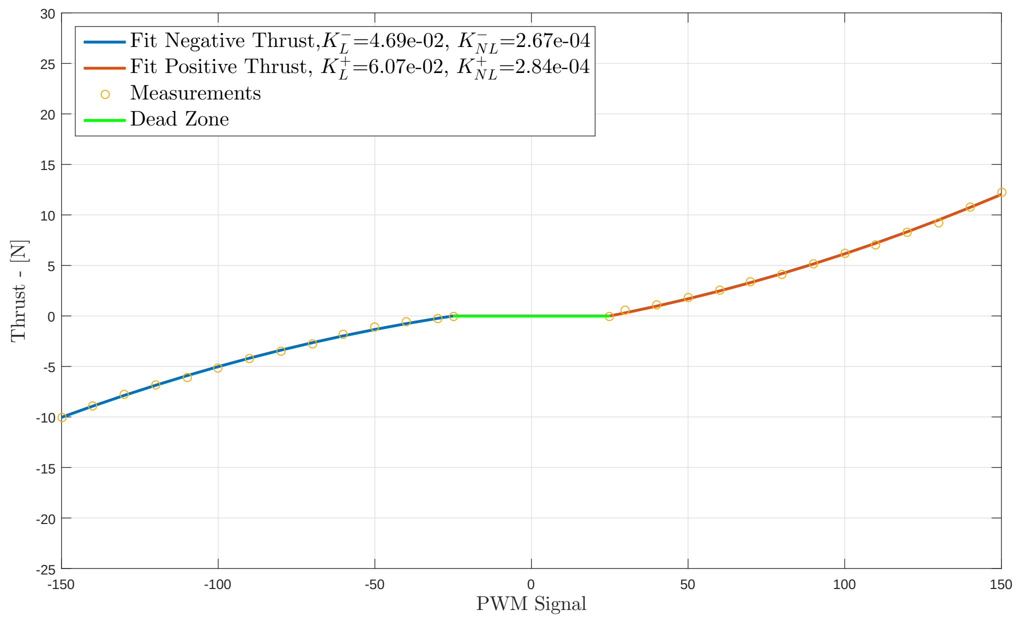


Figure 3.4: Mapping between PWM signal and thrust force.

3.2 Thrust Allocation

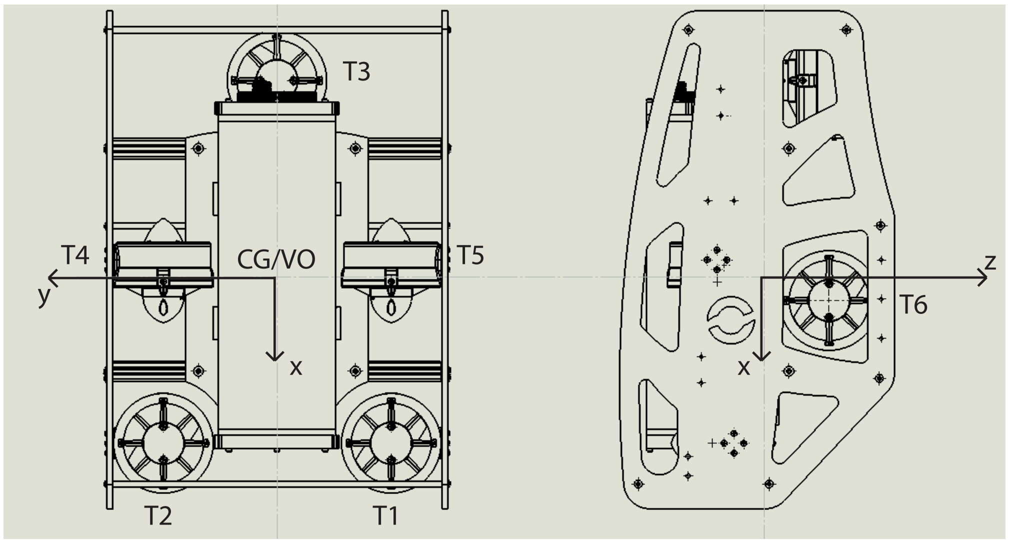


Figure 3.5: Thruster configuration for ROV uDrone.

The thrust allocation takes a 6 DOF force vector as input and commands a PWM signal to each of the six thrusters. A good thrust allocation makes it a lot easier when it comes to tuning of multiple controllers simultaneously. In (Henriksen & Sandøy (2015)) a thrust allocation function was implemented for the ROV uDrone with the thruster configuration as shown in table 3.1 and figure 3.5.

T_i	$l_{x_i}[mm]$	$l_{y_i}[mm]$	$l_{z_i}[mm]$	Orientation
T_1	152	110	-19	Heave
T_2	152	-110	-19	Heave
T_3	-200	0	73	Heave
T_4	-20	110	-19	Surge
T_5	-20	-110	-19	Surge
T_6	13	0	85	Sway

Table 3.1: Distance from each thruster to vehicle origin.

The thrust signal for each thruster is calculated by formula 3.4, where τ is the controller force, and $\mathbf{f}$ is the desired force from each thruster.

$$\tau = \underbrace{\begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ l_{y_1} & l_{y_2} & l_{y_3} & 0 & 0 & -l_{z_6} \\ -l_{x_1} & -l_{x_2} & -l_{x_3} & l_{z_4} & l_{z_5} & 0 \\ 0 & 0 & 0 & -l_{y_4} & -l_{y_5} & l_{x_6} \end{bmatrix}}_{\mathbf{B}} \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix}}_{\mathbf{f}} \tag{3.4}$$

The force for each thruster is then found by $\mathbf{f} = \mathbf{B}^{-1}\tau$. Finally, the mapping from [N] to [PWM] is done according to the test results shown in figure 3.4.

Chapter 4

Computer Vision

Good vision is vital for us humans to observe the world, as well as to orient ourselves about where we are relative to things around us. A focus lately has been to implement this ability into our robots, so they also can start to operate in new, unknown, environments and take decisions based on visual input. However, this requires some sophisticated skills, to be able to separate objects from each other, recognize and analyze the data autonomously. Researchers state that the human brain processes visual information in the semantic space mainly, which means we organize data based on meaningful data, such as shape, lines, and boundaries. (Zhang (2010)). This way of feature recognition is more difficult for machines, so other techniques have been dominating in computer vision in the past few years. Based on robust, but less informative features such as color and texture, machines can extract information from a scene that becomes available to the system. One example can be the location of the object in the video frame, or how an object is moving compared to a static background. Other examples are, machines ability to read license plates, Facebook's automatic face-recognition, or drones flying around a building to make a 3D model. So different methodology is used for different purposes, and there is no true answer to which method that is the best for your specific task.

Computer vision is simply manipulation and analysis of video images. Each image is a matrix where each cell represents a pixel. In each cell, three numbers describe the color for that pixel. So for an image with resolution 640 by 480 px we get 640 x 480 x 3 = 921 600 numbers. Computational time is therefore strongly affected by the resolution of the image stream. You can imagine doing CV on 4K resolution with 4096 x 2160 px, which gives a total of 26 542 080 numbers, 30 times per second. That will require an enormous amount of computational power. Optimized algorithms are therefore critical, and the images are usually cropped or downsized before the algorithms are applied. These numbers can then be processed and manipulated to perform specific tasks, such as masking out a feature, changing the contrast, or just blur the image. Masking is usually done by thresholding a color channel, while blurring is done by replacing each pixel with the average of the pixels around them. The matrix for an image is illustrated in figure 4.1

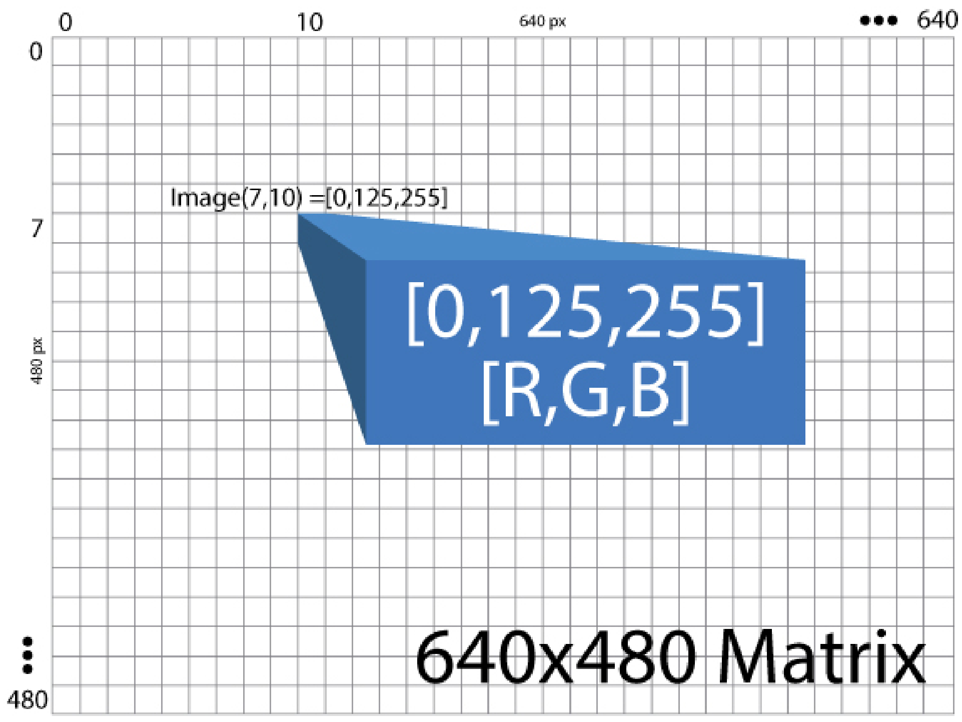


Figure 4.1: Pixels in a video frame, with RGB to describe color.

4.1 Choosing a Software Library for Computer Vision

In chapter one, multiple open source computer vision libraries were found. The most promising were "SimpleCV" and "OpenCV". "SimpleCV" provides a lean introduction to computer vision. The library is so simple that it is not very efficient and neither give access to more advanced raw image processing. However, it is a great tool to learn the concepts of computer vision, which then makes it easier to understand OpenCV and its possibilities. OpenCV, on the other hand, is a high-performance computer vision library that has been around since year 2000. OpenCV provides efficient, advanced, algorithms that are required in order to process video real-time. The library is also open-source, which means anyone can contribute to the library. Most likely, many people on the Internet are facing the same problems as you, which eases the debugging and development process.

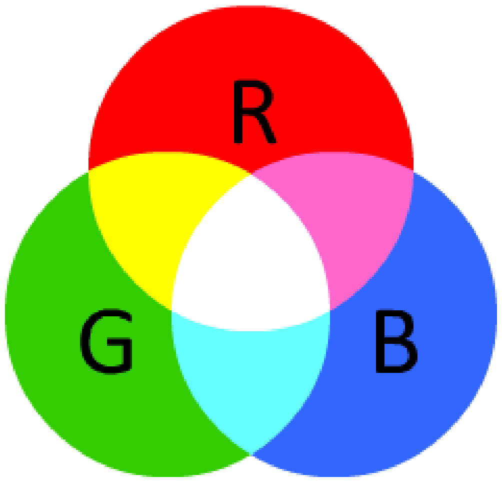
4.2 Detecting Objects Under Water

Different strategies are available for tracking objects. Some utilize features such as textures along with sharp edges and corners, while others track the motion of each pixel from image to image (Optical Flow). The problem with these methods is that on the sea bottom we always have particles in the water, and the brightness varies a lot throughout the image. This makes common methods such as Optical Flow and Canny Edge Detection less feasible. Filtering by the objects color was therefore chosen to increase robustness in different environments. This can be done with existing functions in OpenCV based on RGB or HSV color system. Next, we will have a closer look at the different color systems and determine the most practical system for color extraction.

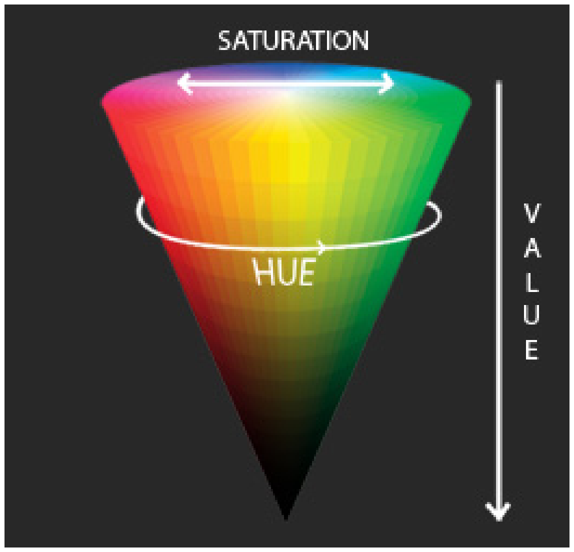
4.3 Color Systems

There are many ways of describing colors. RGB and HSV are made for computer- and TV-displays, while CMYK is made for printing posters and accurate real world representation of colors. The CMYK color system is defined as four numbers that represent how many percents of the colors cyan, magenta, yellow, and black should be used in the printer. The RGB color system consists of three colors red, green, and blue, is the most common color system for computers. By combining these colors, a huge variety of colors can be reproduced on a computer monitor. RGB is an additive model that adds green, red, and blue together. A number from 0 to 255 sets the amount of each color. The two color spaces are shown in figure 4.2.

Let us say you want to find the color values for a color on an object you are holding in front of you; then it is quite hard to tell how much we need of each color (RGB) to get that color. Then



(a) RBG color system.

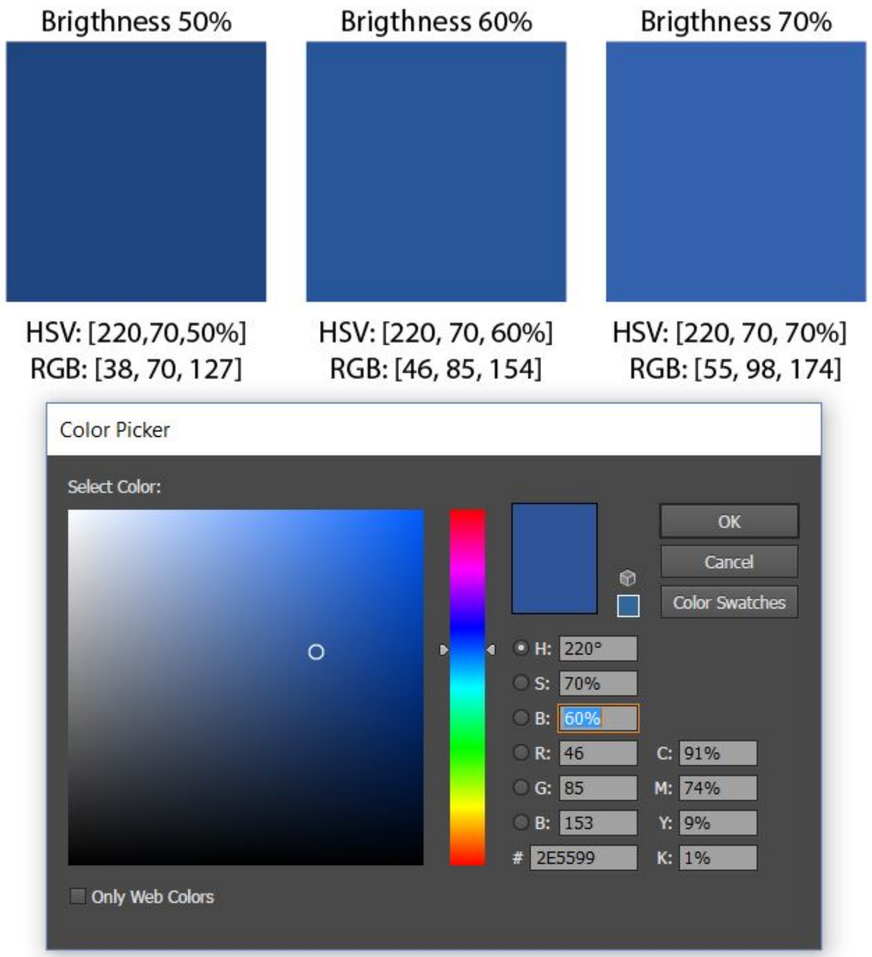


(b) HSV color system.

Figure 4.2: Two different color systems to represent colors in computers (Image source: kirupa.com).

there is the more intuitive color system called Hue Saturation Value (HSV), which is closer to how an artist mixes colors. Hue determines the main color; Saturation is the intensity of the color, and Value is the brightness of the color. Now find the right main color (hue), then find the correct saturation, and at last, brightness. This makes it much easier to determine the color values for the object manually.

An example is illustrated in figure 4.3 where the difference in the HSV color system is only the brightness value. This means its the same color, but with different light conditions. Doing the same task to find RGB values is almost impossible because RGB does not have the same intuitive meaning to us humans. For filtering of objects in an image, an upper and lower boundary for the desired color must be found. As seen in figure 4.3, this is much easier in the HSV color system. If we are tracking objects with a uniform color, it will appear with a different brightness of that color. Depending on the light setting we will always get some darker areas and some brighter ones on the tracked object. These thresholds are also changing from object to object and from environment to environment. It was, therefore, important to find a good method to set upper and lower thresholds to filter out any color of interest. This effort was significantly reduced when boundaries were set in HSV format rather than RGB, with live tunable parameters. The implementation will be presented in chapter 5.



H: 220°
S: 70%
B: 60%
R: 46
G: 85
B: 153
2E5599
C: 91%
M: 74%
Y: 9%
K: 1%

Figure 4.3: Difference in HSV and RGB. HSV makes more sense for humans.

Chapter 5

Camera-assisted Positioning Control Based on Feature Tracking (CAPC)

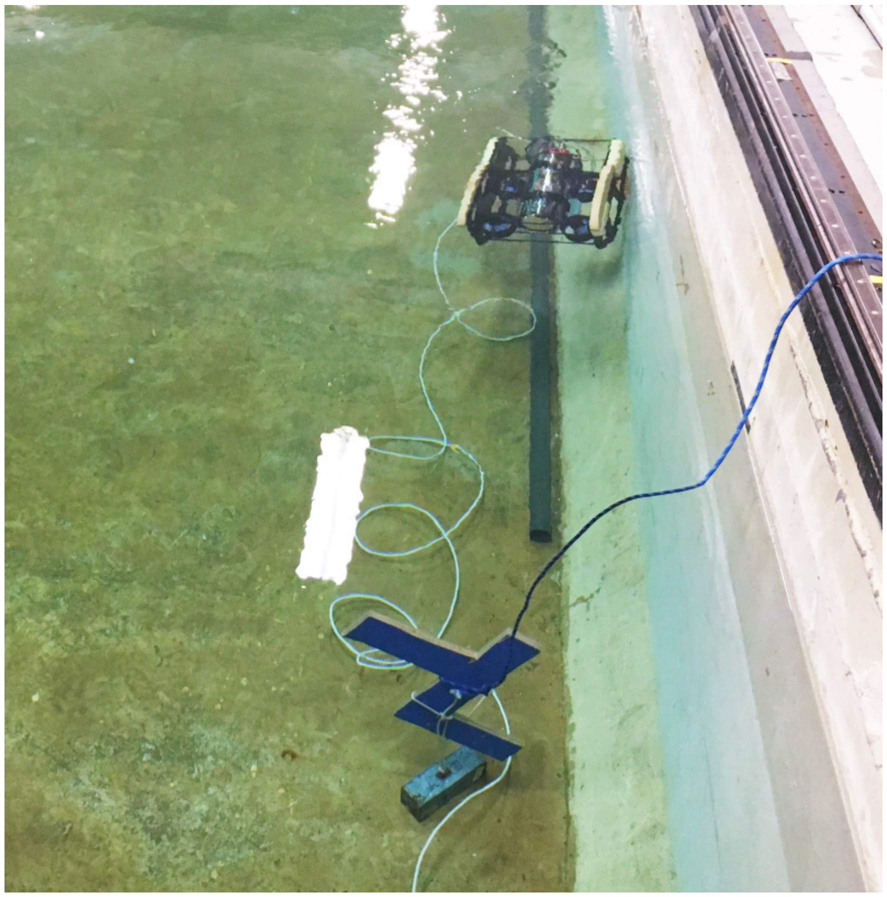


Figure 5.1: uDrone in MC-Lab tracking an object.

5.1 Problem Formulation

When the ROV is operated in direct motion mode, the user experiences a lack of control because of currents and drag forces from the umbilical. These forces change continuously depending on the environment. It also induces undesirable oscillations and motions that are hard to compensate for manually. So our goal is to enhance control performance in unknown waters while doing inspections. The idea is to find an object that stands still on the bottom and let the ROV track a feature on that object. In this case, the feature we are tracking is the color. The ROV should then be able to lock visually on to this "target". Now the algorithm shall tell the system where the object is along with its size. With feedback control on this information, is it then possible to ease the control effort for the user? The Feature-Tracking control modes should be compatible with a manual control input, depending on what degrees of freedom the user wants to move in. These modes are explained in section 5.3

5.2 Feature Tracking Coordinate System

The coordinate system inside the video frame is shown in figure 5.2. The origin is at the top-left corner, with positive x-direction towards the right, and positive y-direction downwards. For simplicity, the local coordinate frame is shifted to the center of the image before the signals get exported. Center point X (cX) and Y (cY) have units pixels, and the area has pixels squared.

5.3 Feature Tracking Control Modes

Some general sub-control modes have been developed for the Feature-Tracking algorithm and presented in figure 5.3. By decoupling the 4 DOFs, we can introduce automatic control in some directions while keeping others in manual control. Traditional automatic controllers as Auto-depth and Auto-heading can also be combined with the Computer Vision modes. This combination then gives benefits as full dynamic positioning without any external positioning system.

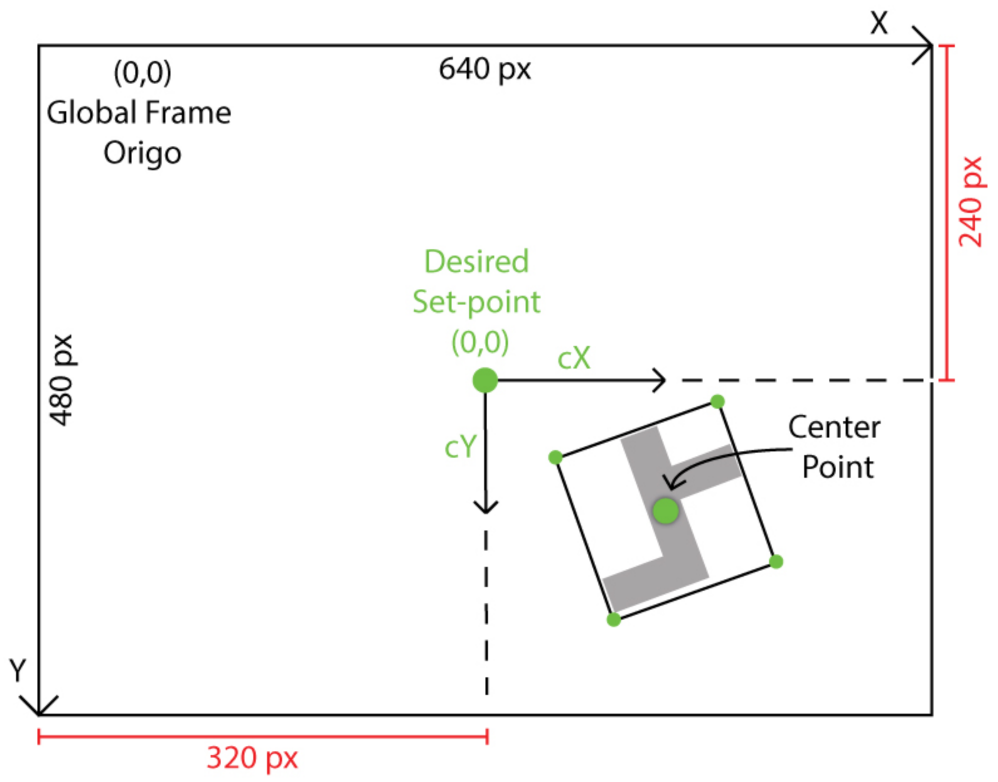


Figure 5.2: Coordinate frame for the Feature Tracking algorithm.

5.3.1 Heading Mode

The basic functionality of the Feature-Tracking algorithm is to keep the object in the camera by controlling the yaw motion. That is, if the object starts to drift out to the left, then the ROV should turn to the left to keep the object in the center of the video frame. The horizontal position of the object (cX) is the only signal used in this mode. The other DOFs, surge, sway, and heave, are controlled by the joystick. This mode provides good flexibility in maneuvering in front and around the object.

5.3.2 Distance Mode

Distance Mode also provide heading control like the mode above, but now we take the area of the feature into the controller, enabling us to control the surge motion. Hence, we can hold a certain distance to the object while we keep the object in the center horizontally. It is still possible to control the vehicle manually in heave and sway with the joystick.

5.3.3 Orbit Mode

Now we introduce the Auto-depth controller that takes pressure as input. The set-point is initially set to the depth where the controller was turned on. With integral action in the vertical position of the object (cY), we can sum up the error in vertical offset and add this to the desired depth for the auto-depth controller. This way we can combine the vertical offset given in pixels with the depth given in meters. Anti-windup is of course of interest, especially if the ROV hits the bottom, and still wants to continue downwards. The alternative would be to have direct feedback on the vertical position of the object (cY) in the video frame, but then we are very sensitive to pitch motions, which can induce big variations in vertical position, and again large control forces. On the other hand, the integralaction is averaging these variations and slowly changes the set-point for the desired depth.

This mode is called the Orbit Mode because it allows the user to move in a circle around the object to view it from different angles. Sway motion is the only input from the joystick, which is controlling the direction of the orbit. If we let go of the joystick, and current is present, the ROV will naturally drift down to the heading where it has the least resistance, hence facing the current. This is usually the heading with greatest stationkeeping capabilities for most ROVs. This mode is also feasible for ROVs that does not have actuators for sway motion.

5.3.4 Full DP Mode

In this mode, the heading is measured by the compass, IMU, or Qualisys. This allows the horizontal position (cX) of the object to control the sway motion, instead of yaw motion as in the previous modes. We now have full DP, and no input from the joystick is possible. This mode requires good heading and depth measurements, while using a traditional heading- and depth-controller. At the same time, we are using computer vision to control sway, distance, and auto-guidance for the desired depth. Now there is no DOF controllable from the joystick, and the ROV is keeping its position automatically.

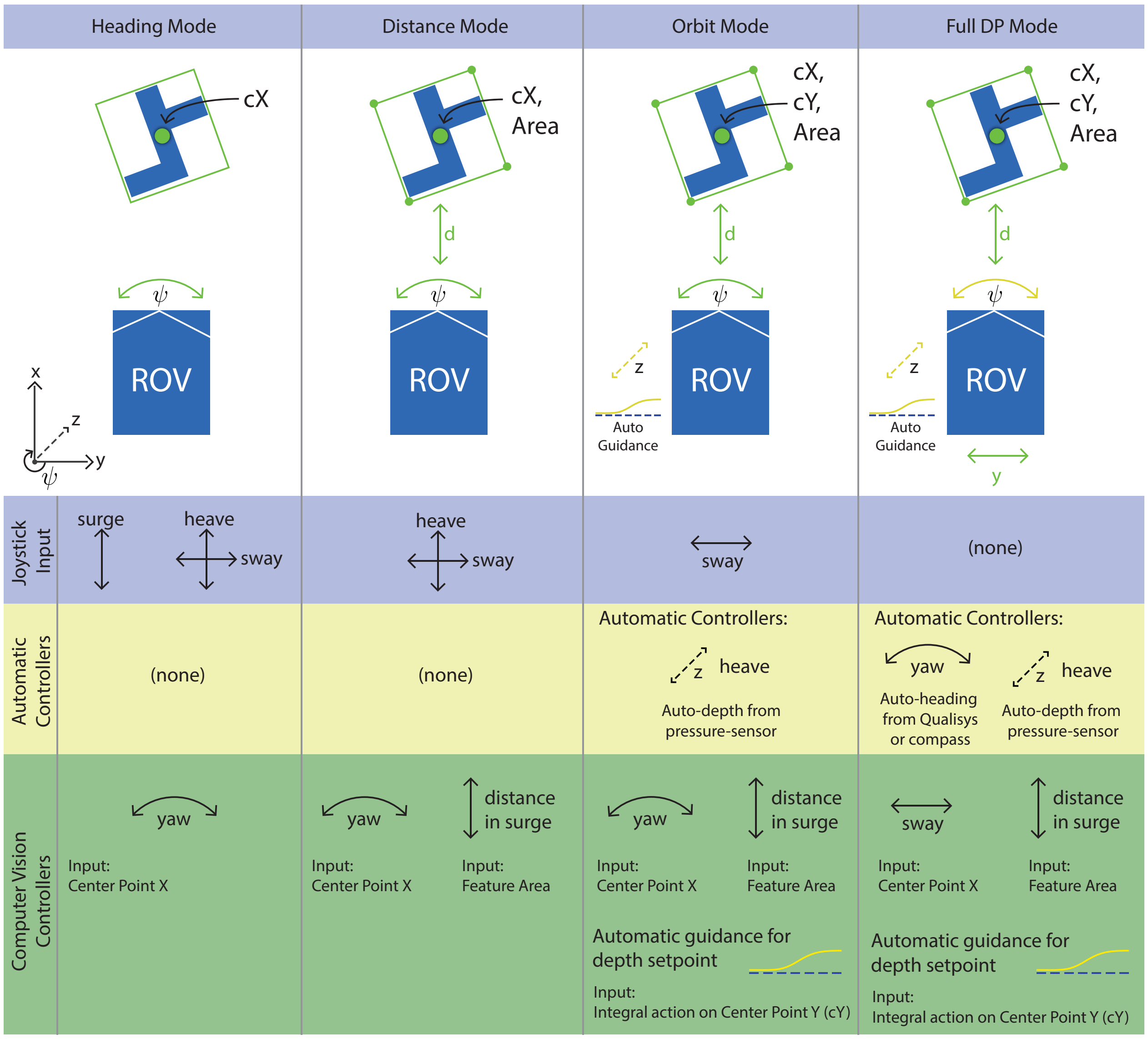


Figure 5.3: Illustration of the different control modes using Feature Tracking.

5.4 Control Design

5.4.1 Control Objective

The objective for the Camera Assisted Positioning Controllers is to get the object to the center of the video frame, while maintaining a constant area of the object. The area is decreasing if the ROV moves away from the feature, and increasing if it gets closer to the feature. Hence, it is now possible to maintain a distance without knowing the actual distance to the object. The distances and the coordinate system is illustrated in figure 5.2. Once we have the object in the center of the video camera, it is easier to get good video footage while easing the control effort at the same time.

5.4.2 PID Controllers

The control objective can be solved by regulating yaw-, heave-, and surge-motion of the ROV. The errors for cX, cY, and cntArea can be defined as:

$$e(t) = x_d(t) - x(t) \tag{5.1}$$

where the desired position (Cx_d) is set to 0 in order to keep the feature in the center. A PID controller is chosen, and defined as:

$$u_\psi(t) = K_P e(t) + K_d \dot{e}(t) + K_i \int_0^t e\,dt \tag{5.2}$$

for Auto-heading, and similar for the other controllers.

5.4.3 Filtering and Signal Processing

The object position values cX and cY were filtered through a low-pass filter to ease out noise before the controller. The filter induced a small delay of about 20 ms, but was not considered as a problem since it is less than 10 percent of the delay we already have in the video streaming from ROV to the topside computer. The low-pass filter is presented below with x as the measured input and y as the filtered output.

$$y(n) = (1-a)x(n) + ax(n-1) \tag{5.3}$$

$$a = \frac{T_s}{Tf + Ts} \tag{5.4}$$

The filtering ratio a was tuned to 0.125, with sampling frequency $T_s = 0.1$, and filter gain $T_f = 0.7$. This means we trust the new measurement with 87.5% and the previous measurement with 12.5%.

5.5 Implementation

5.5.1 Detecting Features in Video Stream

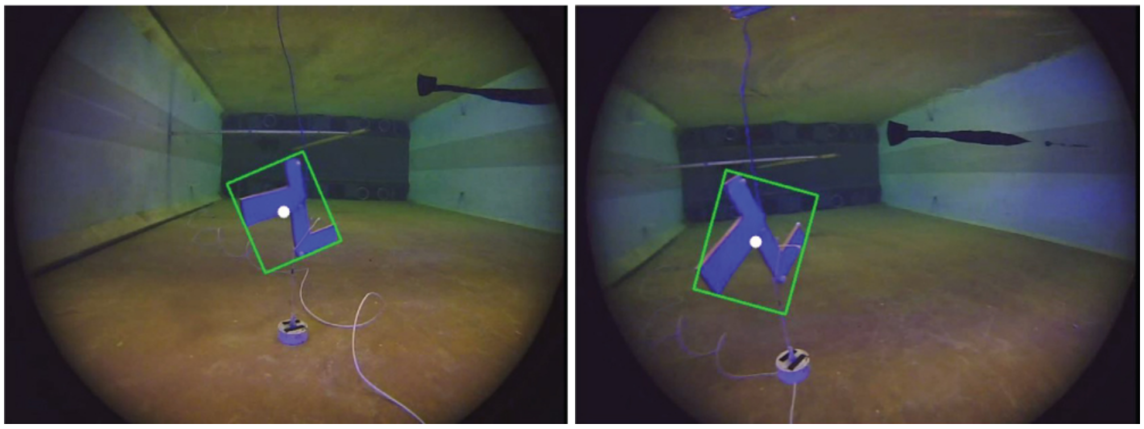


Figure 5.4: Tracked feature in MC-lab. The White dot and the green area are used as control inputs.

There are multiple approaches to find objects in the video frame. Background subtraction was considered not to be practical because the ROV will be moving most of the time, and the feature of interest is most likely standing still relative to the background. Optical flow was also considered, but due to particles in the water, this also gets tough. So a filtering strategy based on unique colors was then chosen in order to find a simple and robust feature. Finding the right limits for the color if interest can, however, be difficult sometimes. Especially at first when this was done in the RGB color system, then it was necessary to take a test image into photo editing program to extract the correct RGB values for upper and lower boundary color on the object of interest. This made it very difficult to change color to be tracked. In section 4.3 we talked about different color spaces, and came up with a solution of using HSV color system to filter the image instead. The Process of tracking the object will now be presented step by step. A flow chart of the algorithm is presented in 5.5.

5.5.2 The Feature Tracking Algorithm

With live tunable HSV parameters, these boundaries could be set very quickly. This was essential when testing in MC-lab since the light conditions were very different over and under the water.

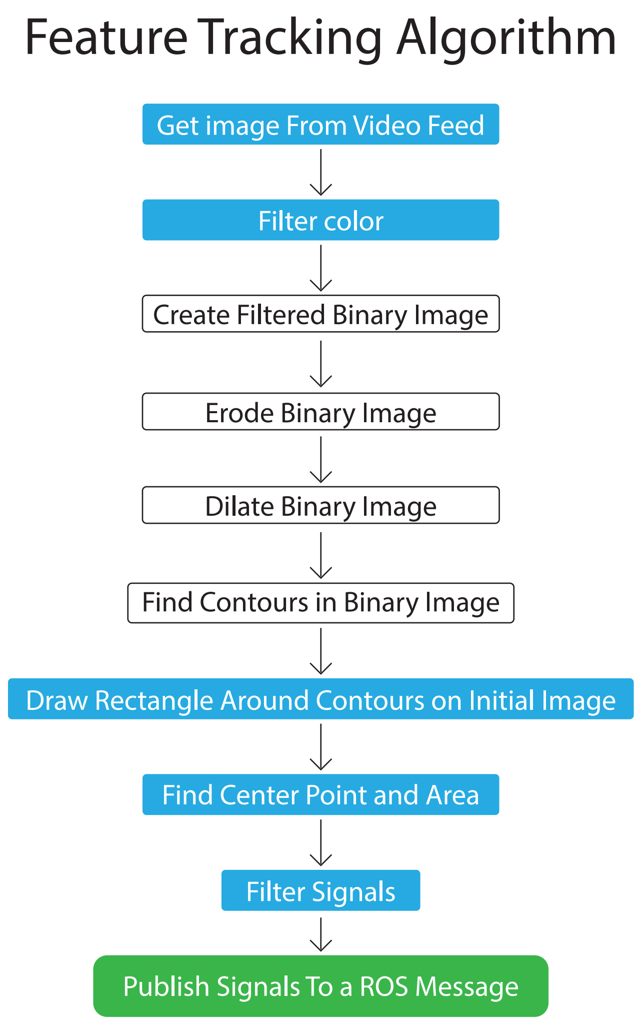
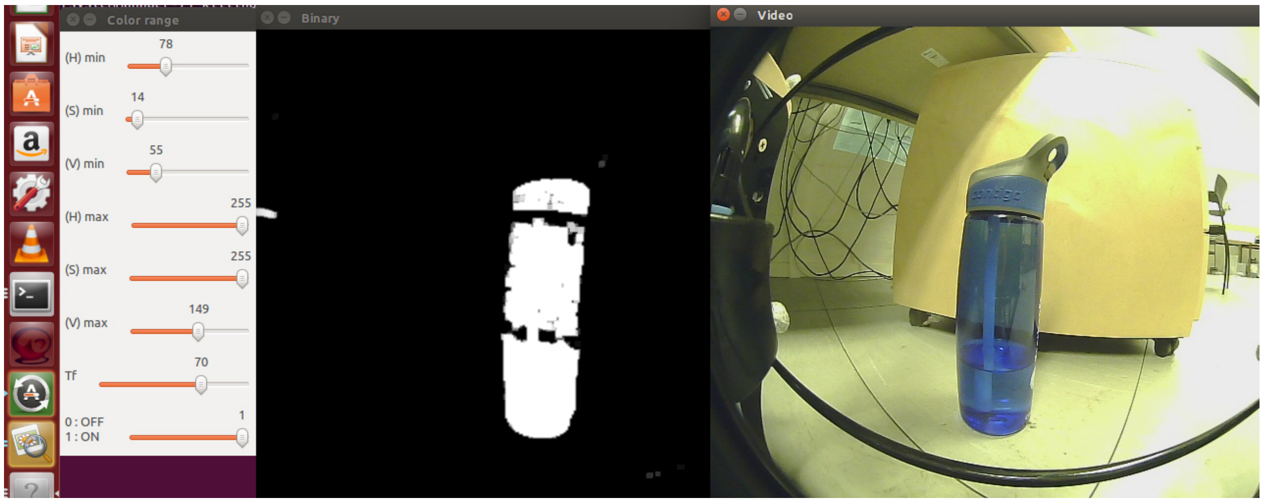

Feature Tracking Algorithm
Get image From Video Feed
Filter color
Create Filtered Binary Image
Erode Binary Image
Dilate Binary Image
Find Contours in Binary Image
Draw Rectangle Around Contours on Initial Image
Find Center Point and Area
Filter Signals
Publish Signals To a ROS Message

Figure 5.5: Flow chart of the Feature Tracking algorithm in OpenCV.

Also, when testing different objects, it was convenient to have good technique to filter them out quickly. This is off-course also of great importance if the system were to be implemented in a consumer product, where the objects if interest is not known prior to the dive.

When the right color limits are set, we make a binary image that is an image with only solid black and pure white colors. The feature is now shown as white, and the rest is filtered out as black. The filtered image can be seen to the left in figure 5.6. To make sure the white contour is treated as one body, we use a Gaussian Blur blur function to filter out noise grains. OpenCV's erode function is then used to smooth out the edges followed by the dilate function. One example is presented in figure 5.7. Dilate adds add a white border around the edge to fill in any small remaining holes. This was necessary in order to avoid our mask being separated into two



(H) min 78
(S) min 14
(V) min 55
(H) max 255
(S) max 255
(V) max 149
Tf 70
0 : OFF
1 : ON 1

Figure 5.6: Tracked feature at right, filtered binary image at left.

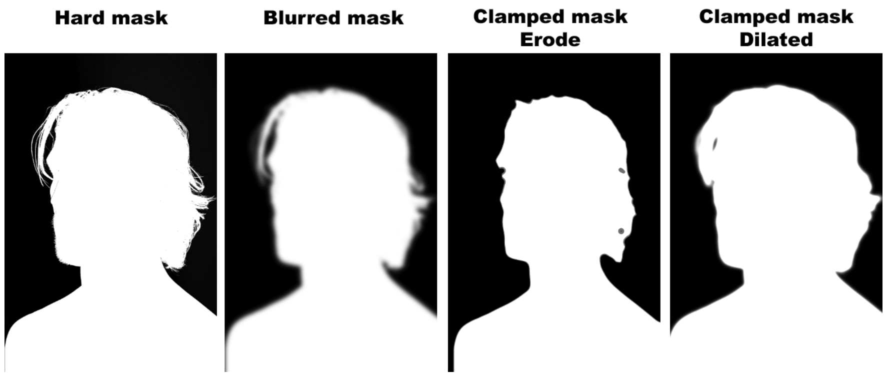


Figure 5.7: The effect of blur, erode, and dilate (Source: opticalenquiry.com).

contours, due to the ropes that were holding the object in place. In Figure 5.6 we can see how the masked contour has a small trace of the shadow behind, but the contour is trying to fill inn this area. The Find Contours function

```
cnts = cv2.findContours(mask.copy(),cv2.CHAIN_APPROX_SIMPLE)
```

in OpenCV provides a list of all contours in the image, sorted by its size. The second argument: "`CV_CHAIN_APPROX_SIMPLE`" means we want to store each contour as simple as possible, which is a rotatable rectangle, storing only the location of each corner. We pick the largest contour, and apply a drawing function, `cv2.boxPoints()`, that fills in the lines between each corner point. Now the rectangle is drawn on top of the unfiltered image. The center point of the rectangle is calculated by averaging the x and y values for the four points, while the area is found from the `cv2.contourArea(cnt)` function. The signals are now filtered by the low-pass filter presented in equation 5.3, before they get published to the ROS system.

The feature tracking algorithm was implemented with Python as a new control node in ROS, which made the computer vision signals available for all the other nodes in the ROV.

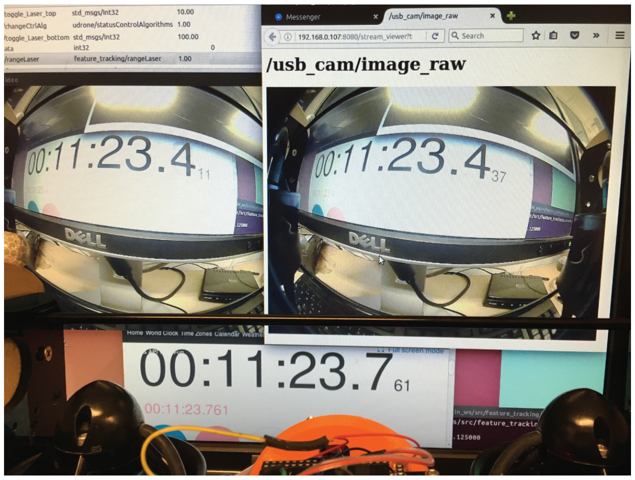
5.5.3 The Controllers Based on Computer Vision

The Computer Vision based Controllers were implemented in a separate node, and made available to the user through a button on the x-box controller. This node was written in C++ and was only using input from the feature tracking node, and the depth sensor depending on the operation mode, as shown above in figure 5.3.

The PID-controllers shown in equation 5.2 were then implemented discretely with constant loop frequency on 10 Hz. Manual control forces were also added into the control force vector if the mode was allowing it before the desired control force was published to the thrust allocation. Live tunable sliders for tuning of the different PID-controllers were also implemented and can be seen in figure 5.9

5.6 Delay in the Control Loop

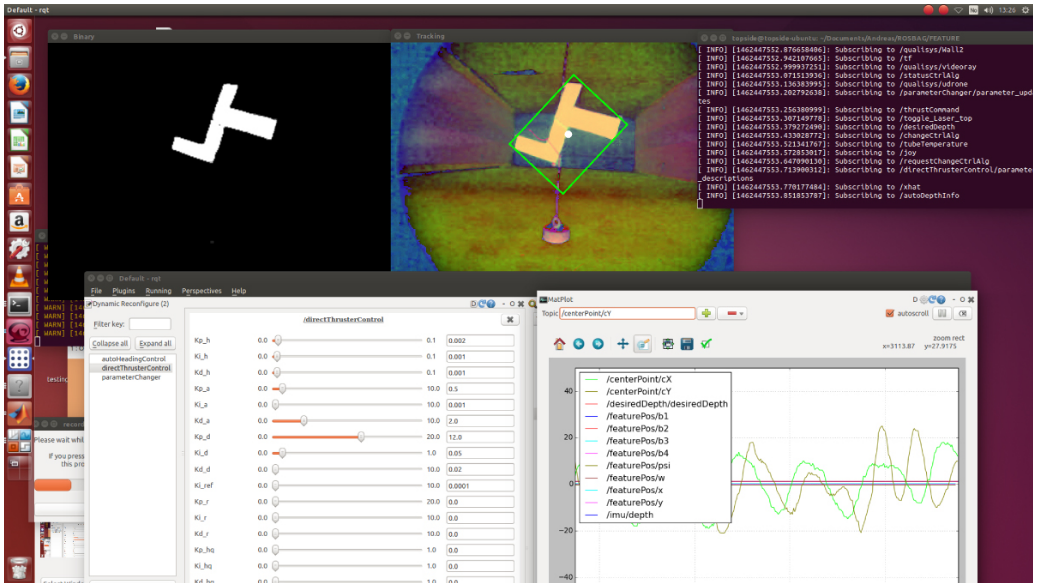
The feature tracking algorithm was tested both on the topside computer and on the RPi2 inside the ROV. Both options had pros and cons. Running topside gives a lot more computing power, but a small delay of 324 ms was then introduced due to live streaming over the local network, from ROV to web-browser on the topside PC. However, since the computing power is so much greater topside, it was beneficial to stream video up, do computer vision and then return the control force signals. The Feature Tracking algorithm in OpenCV had a delay of 26ms. Running topside also allows a higher resolution on the video which gives smoother input to the PID-controllers. The total delay from a visual change in the pool to the force command is given is then: 350ms. These values will vary a little depending on how much RPi2 is loaded. The delay test is shown in figure 5.8. On the other side, running locally on the RPi2 should give advantages such as low latency, but at the cost of lower resolution. This option was therefore chosen for the distance algorithm because this camera was directly interfaced with the RPi2 through CSI port, via the PiCamera library for Raspberry Pi and RPi Camera in Python.



```
toggle_Laser_top      std_msgs/Int32                        10.00
changeCtrlAlg         udrone/statusControlAlgorithms        1.00
toggle_Laser_bottom   std_msgs/Int32                        100.00
ata                   int32                                        0
/rangeLaser           feature_tracking/rangeLaser           1.00
```

Figure 5.8: Delay from ROV to the topside PC, OpenCV at left, URL stream at right, actual time at the bottom, for the fisheye camera.

5.7 Graphical User Interface



```
topside@topside-ubuntu: ~/Documents/Andreas/ROSBAG/FEATURE
[ INFO] [1462447552.876658406]: Subscribing to /qualisys/Wall2
[ INFO] [1462447552.942107665]: Subscribing to /tf
[ INFO] [1462447552.999937251]: Subscribing to /qualisys/videoray
[ INFO] [1462447553.071513936]: Subscribing to /statusCtrlAlg
[ INFO] [1462447553.136383995]: Subscribing to /qualisys/udrone
[ INFO] [1462447553.202792638]: Subscribing to /parameterChanger/parameter_upd
tes
[ INFO] [1462447553.256380999]: Subscribing to /thrustCommand
[ INFO] [1462447553.307149778]: Subscribing to /toggle_Laser_top
[ INFO] [1462447553.379272490]: Subscribing to /desiredDepth
[ INFO] [1462447553.433028772]: Subscribing to /changeCtrlAlg
[ INFO] [1462447553.521341767]: Subscribing to /tubeTemperature
[ INFO] [1462447553.572853017]: Subscribing to /joy
[ INFO] [1462447553.647090130]: Subscribing to /requestChangeCtrlAlg
[ INFO] [1462447553.713900312]: Subscribing to /directThrusterControl/paramete
_descriptions
[ INFO] [1462447553.770177484]: Subscribing to /xhat
[ INFO] [1462447553.851853787]: Subscribing to /autoDepthInfo
```

/directThrusterControl

Kp_h	0.0	0.1	0.002
Ki_h	0.0	0.1	0.001
Kd_h	0.0	0.1	0.001
Kp_a	0.0	10.0	0.5
Ki_a	0.0	10.0	0.001
Kd_a	0.0	10.0	2.0
Kp_d	0.0	20.0	12.0
Ki_d	0.0	1.0	0.05
Kd_d	0.0	10.0	0.02
Ki_ref	0.0	10.0	0.0001
Kp_r	0.0	20.0	0.0
Ki_r	0.0	10.0	0.0
Kd_r	0.0	10.0	0.0
Kp_hq	0.0	1.0	0.0
Ki_hq	0.0	1.0	0.0
Kd_hq	0.0	1.0	0.0

Figure 5.9: Sliders for tuning PID controllers and signal monitoring.

Both Range Finder and Feature Tracking do have live tuning of the filtering parameters. This is crucial until some automatic algorithm can filter out features for you. One way this could be done is, by dragging a square with the mouse on the video, or by the touch of your fingertip on an iPad. However, for the purpose of this project, that was not considered of great importance. Sliding bars were therefore implemented to be able to turn video preview on/off, and tuning of video and signal filtering as shown in figure 5.9.

5.8 Testing of CAPC Based on Feature Tracking

A great deal of testing was performed in MC-lab throughout the development phase of this project. One controller was tuned at the time before all were combined in the end. The Auto-heading controller based on the horizontal center point position (cX), was first tested and tuned to make sure the object would not leave the camera's field of view. Next, the Auto-distance based on feature area was turned on and tuned to keep a constant distance. Now it started to be interesting to see how the controllers were behaving together with manual input. At last, Auto-depth and Auto-reference for depth were tuned.

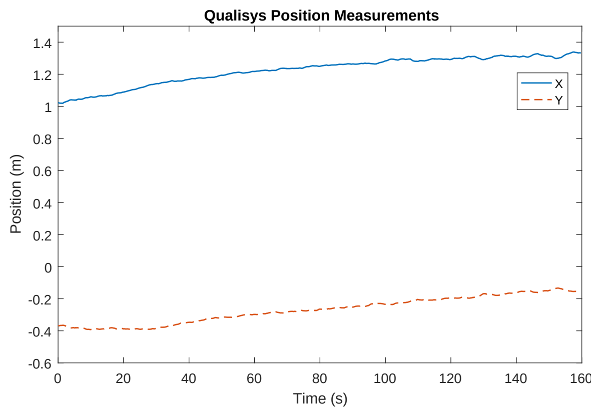
The results from two tests will now be presented. In the first test, we will have a look at the controllers performance at low velocities, and how the object converges to the center of the camera. Then in the second test we are looking into the Orbit Mode, where we allow manual input in sway direction to go around the object. The object used for these tests was a piece of wood that was painted blue. This object was chosen because it had a color that could easily be detected. The fact that the wood floats made it easy to get it standing upraised anywhere in the pool with a weight at the bottom.

Table 5.1: Test scheme for Feature Tracking controllers.

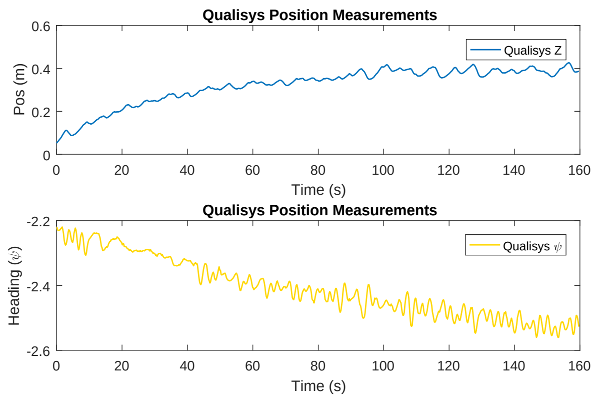
DOF:	Steady-State Positioning		Orbit Mode Test	
	Controllers:	Input Signals:	Controllers:	Input Signals:
Surge	Auto-distance	Feature Area	Auto-distance	Feature Area
Sway	0	_	Manual Control	Joy-stick
Heave	Auto-depth	Pressure Sensor	Auto-depth	Pressure Sensor
Heave-ref	Auto-reference	Center Point Y (cY)	Auto-reference	Center Point Y (cY)
Yaw	Auto-heading	Center Point X (cX)	Auto-heading	Center Point X (cX)

5.8.1 Test 1: Steady-State Positioning

The ROV is starting close to the surface with the object standing in the middle of the water column, in front of the ROV. When the controllers are activated, the ROV will start to regulate the ROV, so the object converges to the center of the video frame. As seen in Table 5.1, we have now activated Auto-heading, Auto-distance, Auto-depth and Auto-reference for depth. The Controllers are turned on, and we will now see what happens.



(a) X and Y position measurement.



(b) Z and ψ position measurement.

Figure 5.10: Position measurement of the ROV from Qualisys positioning system in MC-lab.

In figure 5.10 we see that the ROV is slowly moving in x- and y-direction, while starting to descend at the same time. Figure 5.11 shows a 2D representation of the same data. Here also the object we are tracking is plotted with its pose in the top left corner. The ROV is plotted in blue and moves in positive x- and y-direction as it lowers down into the basin. We can also see that the heading of the ROV is always pointing towards the center of the object. We will now have a closer look at how this is possible.

5.8.2 Auto-heading Controller

The Automatic Heading controller is regulating the horizontal center point of the feature into the center of the camera's field of view. The control function provides a control torque in yaw that is assumed not to influence any of the other degrees of freedom. However, when testing only the heading controller, it was clear that it started to drift closer and closer to the object. This indicates that a control torque in yaw also gives a small contribution in surge motion. In

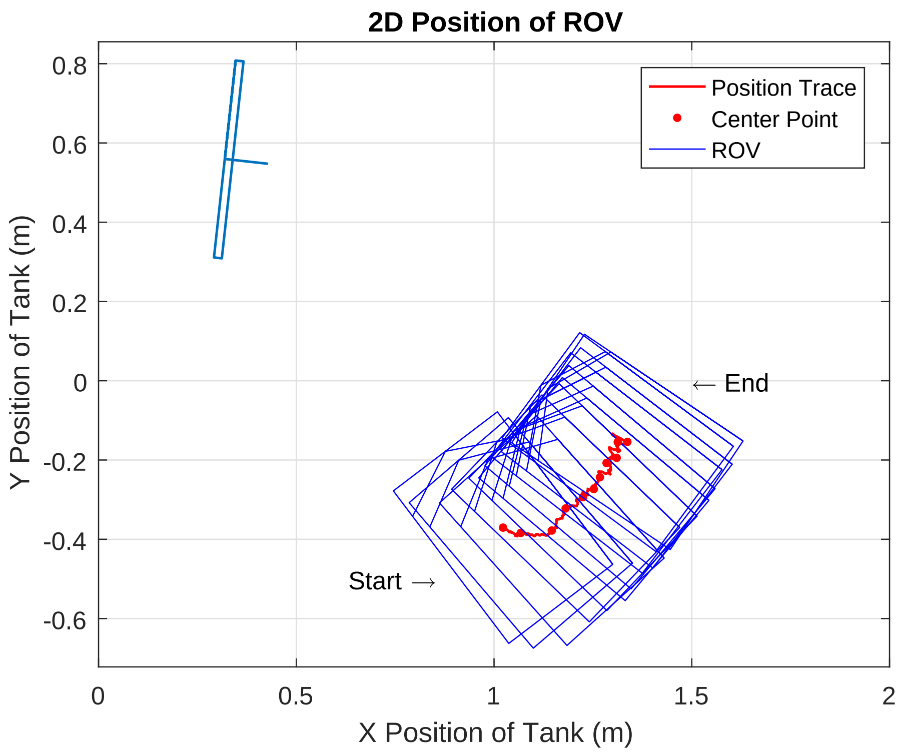


Figure 5.11: Position of the ROV seen from above in basin frame.

figure 5.12 the error for the object's horizontal position is presented. The steady-state error in cX is within +/- 20 pixels from the center, which is almost unnoticeable (5 %). We can also see how the control torque is oscillating back and forth. This is most likely due to the delay in the camera feed; that is causing too much force in one direction before the ROV starts to go the other way already. The peaks get lower if the proportional gain is reduced, but then the controller gets to slow when other controllers, manual control or environmental loads are added. The control torque is within +/- 0.1 Nm and is relatively low. The mean is very close to 0, which indicates that there are not much slowly varying biases. Only in certain situations we can get quite a lot of torque from the umbilical, but for the most part, this has been neglectable. It is also clear to see how the control torque increase as the ROV starts to move in depth and position after about 40 seconds.

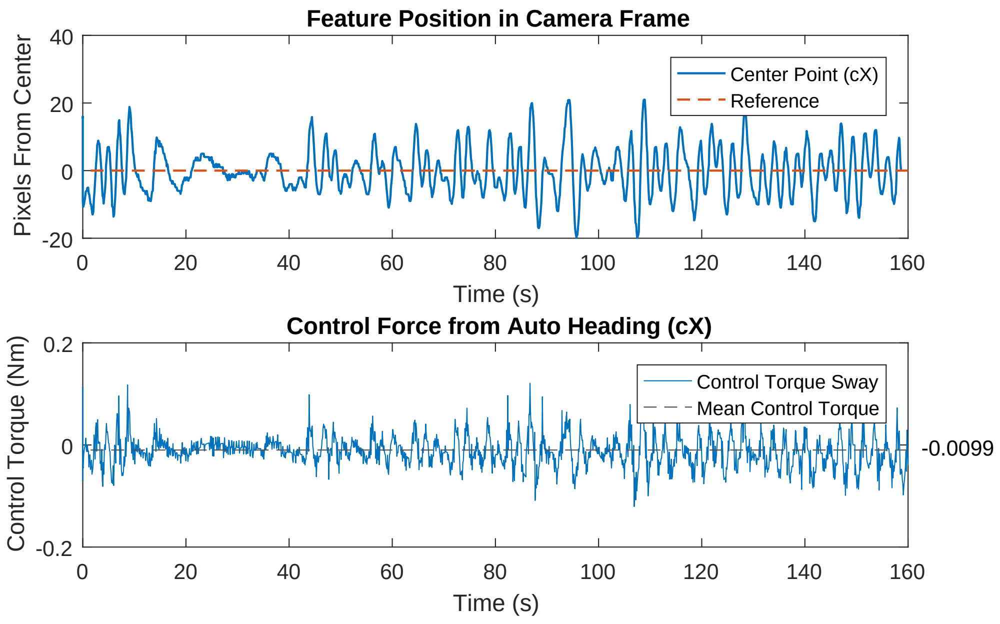


Figure 5.12: Error in the horizontal position of the object in the camera.

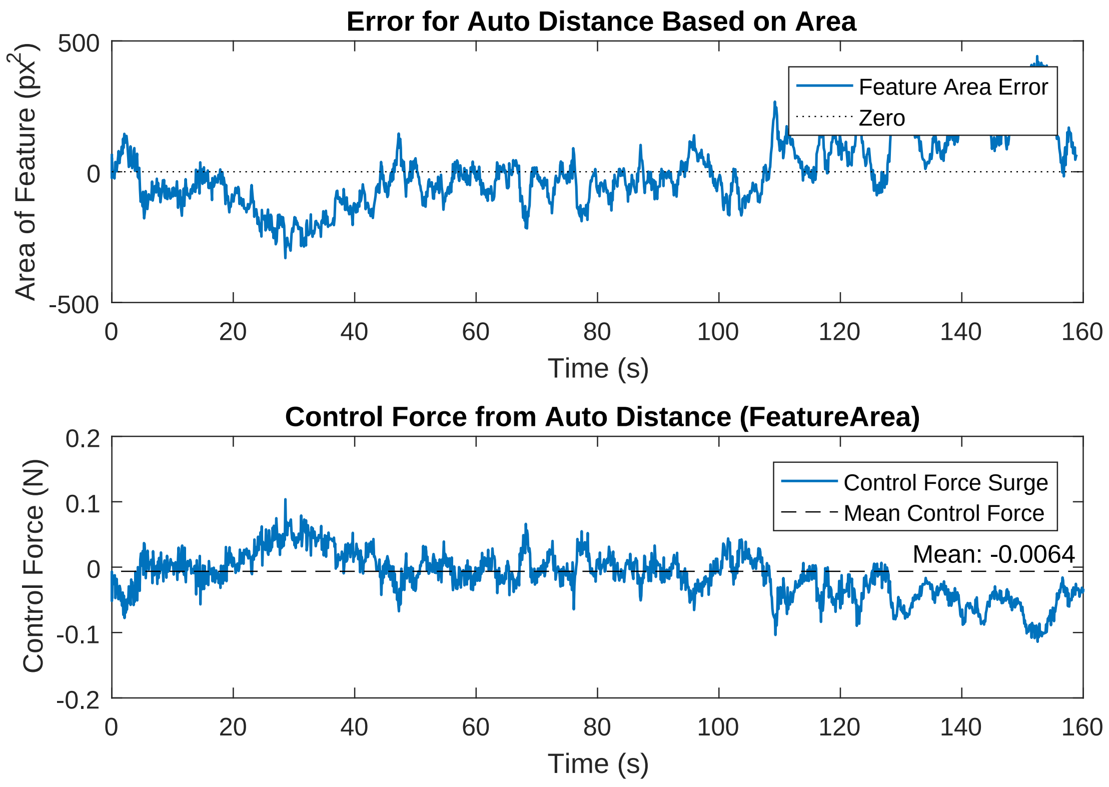


Figure 5.13: Error in the area of feature and control force in surge.

5.8.3 Auto-distance Controller

To make sure we stay at a constant distance away from the feature the Auto-distance controller was implemented. This control function is taking the area of the feature as input and returns a control force in surge. Since the area of the feature is typically 5000 pixels to 6000 pixels, we see that the error presented in figure 5.13 is quite large. However, compared to the input value we get;

$$400px^2/6000px^2 = 0.067 \approx 7\% \tag{5.5}$$

which is quite acceptable. In the same figure, we can also see that the control force is within +/- 0.1 N.

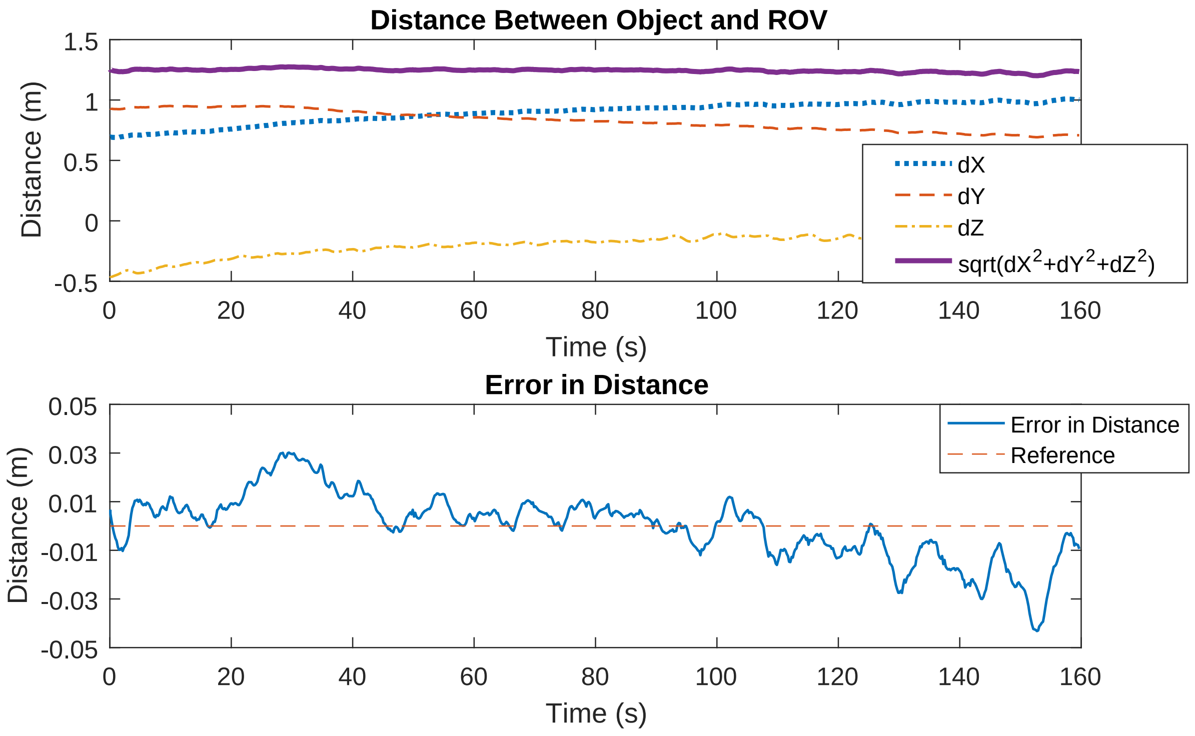


Figure 5.14: Constant distance between ROV and object at top, with error in distance below.

Now, how steady can the controller keep the actual distance to the object? With exact position measurements from the underwater positioning system, Qualisys, it is possible to calculate the "true" distance between the object and the ROV. Qualisys should provide measurements with +/- 1 mm accuracy. The distance in X, Y, and Z direction (dX, dY, and dZ) is found and combined by:

$$d = \sqrt{dX^2 + dY^2 + dZ^2} \tag{5.6}$$

where d is the total distance between the objects. The values from this test is presented in figure 5.14. Due to the motion in heave, we see that it actually has a constant distance for the whole time series, even though it seems like its varying quite a bit in figure 5.11. The deviation in the distance error is also surprisingly low with +/- 3 cm on a total distance of 125 cm (2.4 %).

5.8.4 Auto-depth Controller

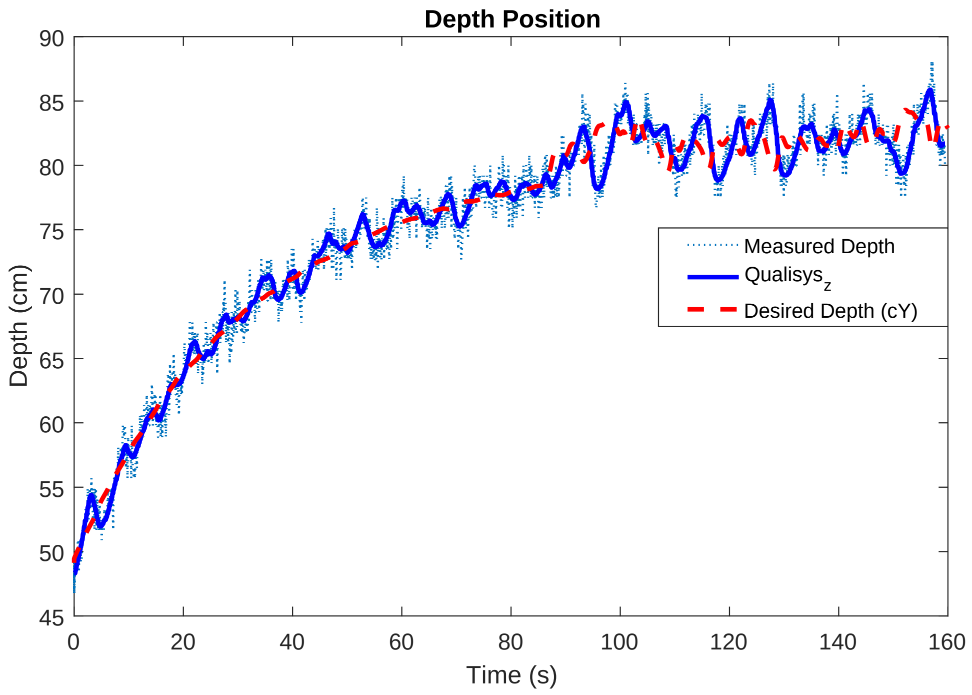


Figure 5.15: Auto-reference is pushing the ROV down to about 82 cm below the surface.

The heave motion is controlled by an Auto-depth control function. The function is taking water pressure converted to depth (cm) as input and returns a control force in heave. In addition, we have the auto-reference function that integrates the error of the objects vertical position (cY), and adds this to the desired set-point for the Auto-depth controller. As seen in figure 5.15, the reference is growing very fast in the first 40 seconds, before it grows slower and converges to about 82 cm below the surface. In figure 5.16 we see how the error in cY decreases and converges to 0 at around 100 seconds. The mean heave control force has a value of 0.75 N which corresponds to about 75 grams of buoyancy, that it has to compensate for.

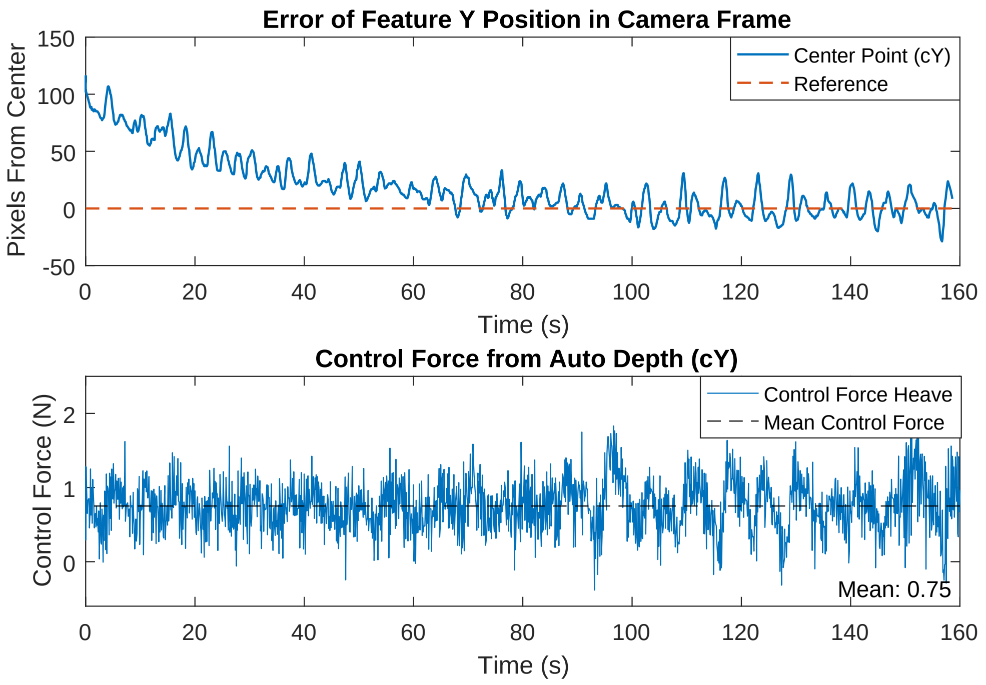


Figure 5.16: Vertical error in feature position in camera frame goes to zero.

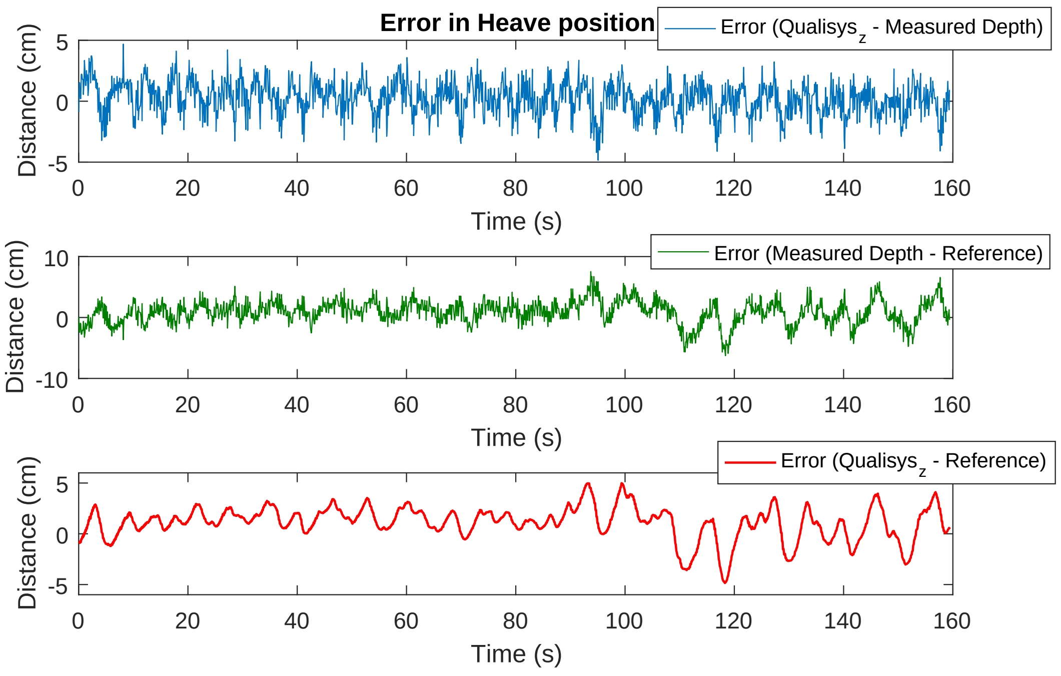


Figure 5.17: Comparison of errors in depth.

In figure 5.17 the difference between measured position from Qualisys, measured depth (pressure sensor), and reference. It is clear to see that the pressure sensor is the noisiest measurement, but it is still following the Qualisys quite nicely within +/- 5 cm. The controller is also following quite well in the 2nd-row plot. The last row shows the total difference between desired depth and "true" position from Qualisys. Note how the error is smaller the first 100 seconds. This is because this is where the ROV is moving downwards, but as soon as it finds the correct depth, it starts to oscillate. This is most probably because it is hard to provide small enough thrust from the thrusters. Still within +/- 4 cm from the reference (4.8 %).

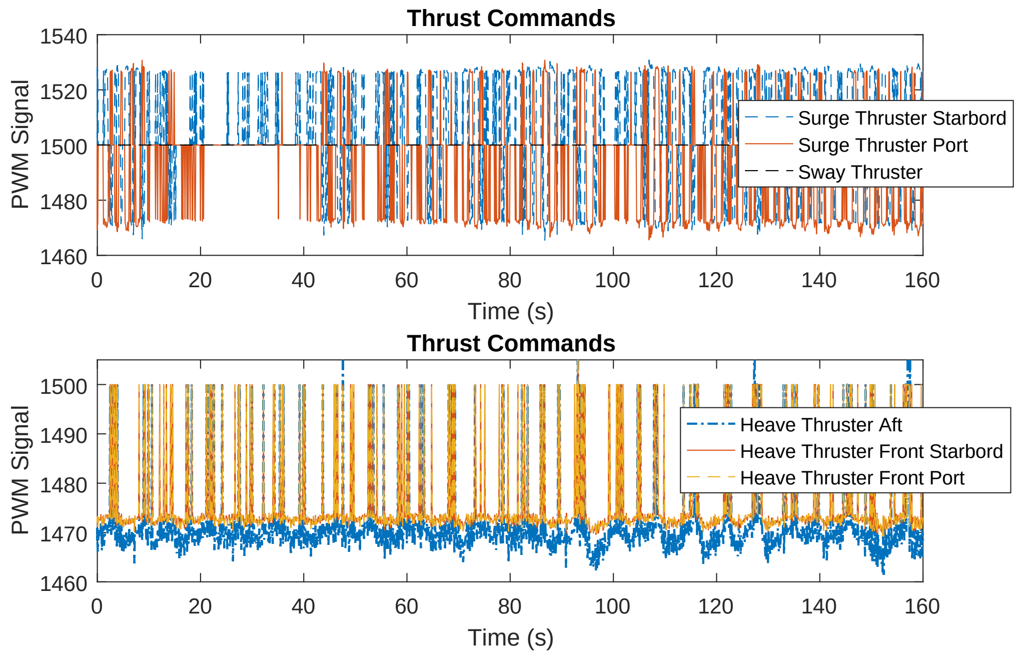


Figure 5.18: Thrust commands from the thrustallocation.

The output of the thrust allocation is plotted as thrust commands in figure 5.18. The white gap between 20 and 40 seconds, indicates that the ROV is driving forward. The PWM signal value gives zero thrust at 1500 and has a dead zone from 1475 to 1525. To get thrust at low control force values, we have to add 25 instantly to the signal. This shows up as spikes for low torque and oscillates back and forth when the control force is working around zero. For the heave thrusters in the next row, it is clear that the aft thruster is on continuously, below 1470. The other two thrusters have the exact same values as long as there are no control forces in sway. They are therefore plotted on top of each other in red and orange.

5.8.5 Object Positioning in Camera

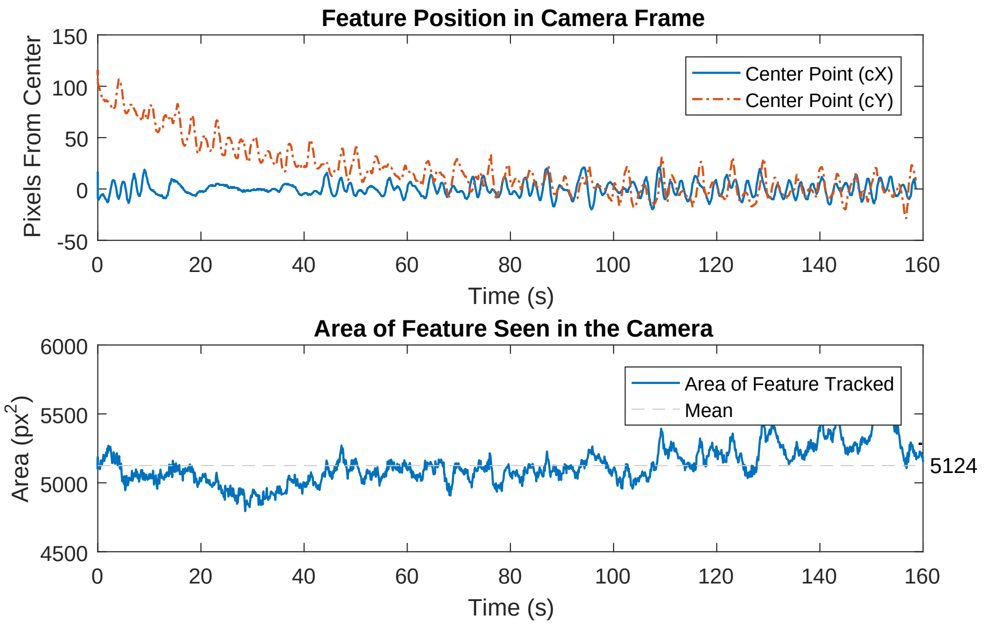


Figure 5.19: The object converges to the center of the camera, while keeping the area constant.

Looking back on the control objective, we now see in figure 5.19 that both center point X and Y converges to zero, and the area stays constant at the value it was when the control function was activated. Figure 5.20 shows how the feature is moving in the camera frame. It starts towards the bottom of the camera frame and ends up in the center. The squares are plotted based on how the area signal was changing throughout the time sequence. The length of the square sides is found by taking the square root of the area signal. The blue line shows the continuous trace of the objects movement in the camera, while the frames are taken at equal time intervals.

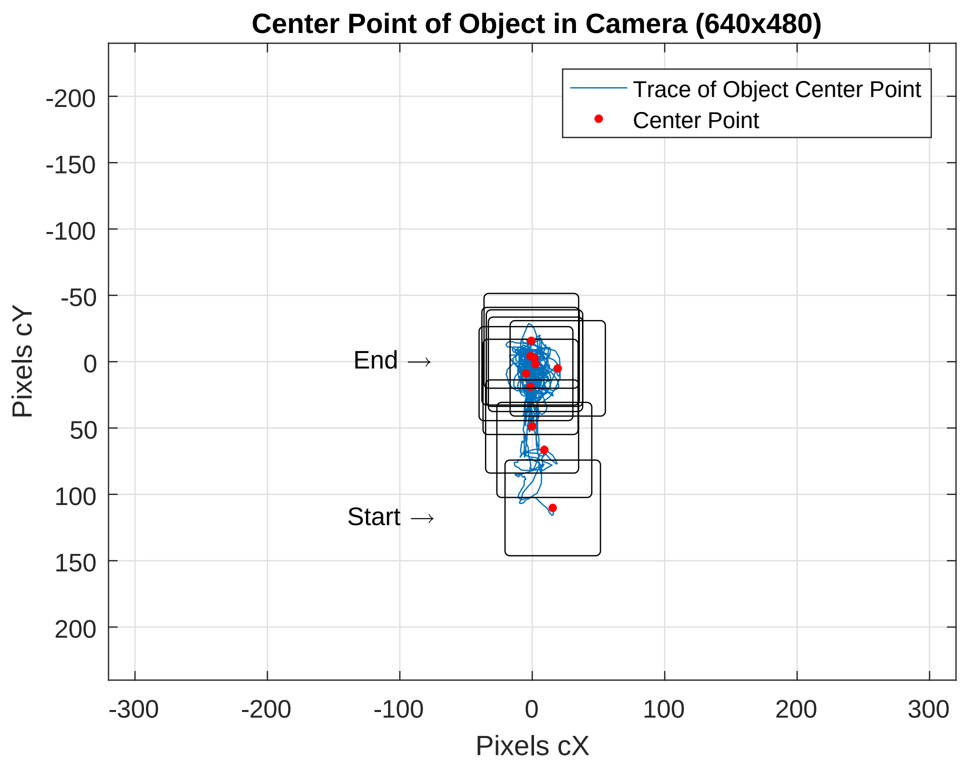
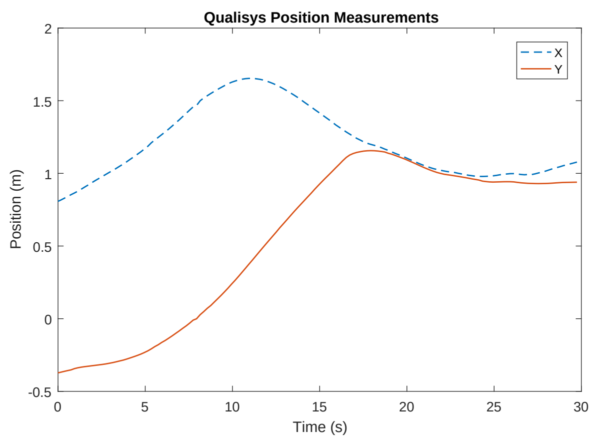


Figure 5.20: The movement of the feature in the camera frame during the time series.

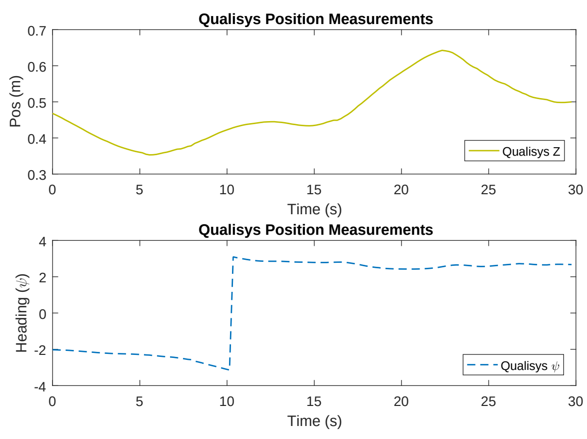
5.8.6 Test 2: Orbit Mode

We are now going to induce manual control force in sway to test the controllers at larger disturbances. The manual control force is limited to 10 N. That is about 10 times more than what the controllers has performed so far. We start out by presenting the position and movement in the basin, shown in figure 5.21. X and Y position seem to vary smoothly, but the depth is changing significantly. However, it converges quite fast down to the desired depth as soon as the manual control force is eased. In figure 5.22 we can see how the ROV is moving in an orbit around the feature. The maximum velocity is at about 17 seconds, right before the sway thruster is turned off. Towards the end of the time series, we see that the ROV approaches the feature. This is because the area of the feature decrease as the viewing angle gets closer to the side of the feature. Also, notice how the heading is always pointing towards the feature. This feels like a vast improvement for the user in controlling the vehicle, and it is also nice to have when recording video.

In figure 5.23, it is clear that we have the object moving more around in the camera frame in



(a) X and Y position measurement.



(b) Z and ψ position measurement.

Figure 5.21: Position measurement of the ROV from Qualisys positioning system in MC-lab.

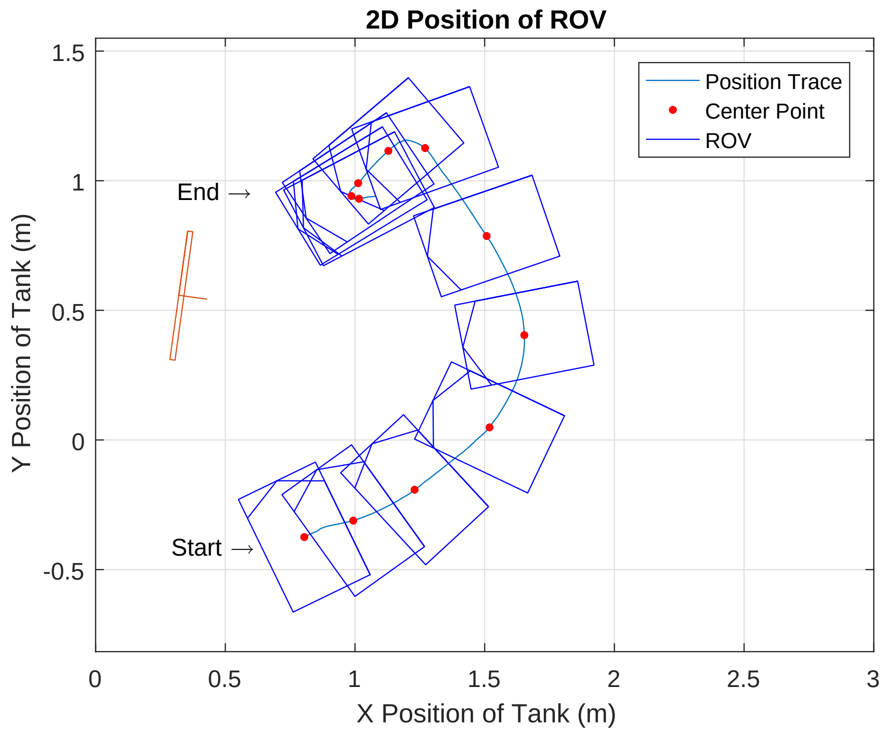


Figure 5.22: 2D position of the ROV in Orbit Mode from a bird perspective.

this mode. However, there is still a long way before we lose the feature out of the field of view. The resolution is set to 640 x 480 pixels which means we have about 2/3 (100 px/320 px) left of the screen on both sides before the object is out of sight. From 25 to 30 seconds we see that the signals are jumping a bit, this is because the feature contour gets split into 2 areas. Since we are always tracking the largest one, it starts to jump between tracking the whole and only the half contour. This is visible in the graphs where the area suddenly drops to a lower value, while the center points jump at the same time.

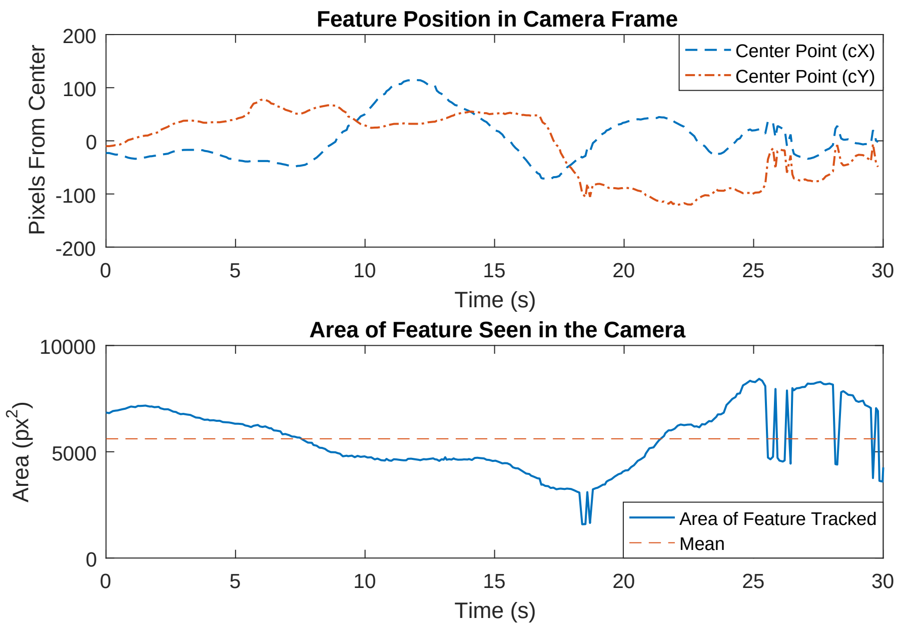


Figure 5.23: Position and area of the feature seen in the camera.

Figure 5.24 shows the thrust commands from the thrust allocation. We now see that the sway thruster is much higher than the others. Due to the thruster configuration, we have to compensate for the roll motion due to the sway thruster. This is why the front heave thrusters also have significant inputs in opposite direction. The heading controller also provides a quite large yaw torque with the surge thrusters in opposite direction to maintain the desired heading.

Figure 5.25 we see how the object is moving throughout the time series. It is moving significantly more than in the previous test, but it is still within the center area of the camera frame.

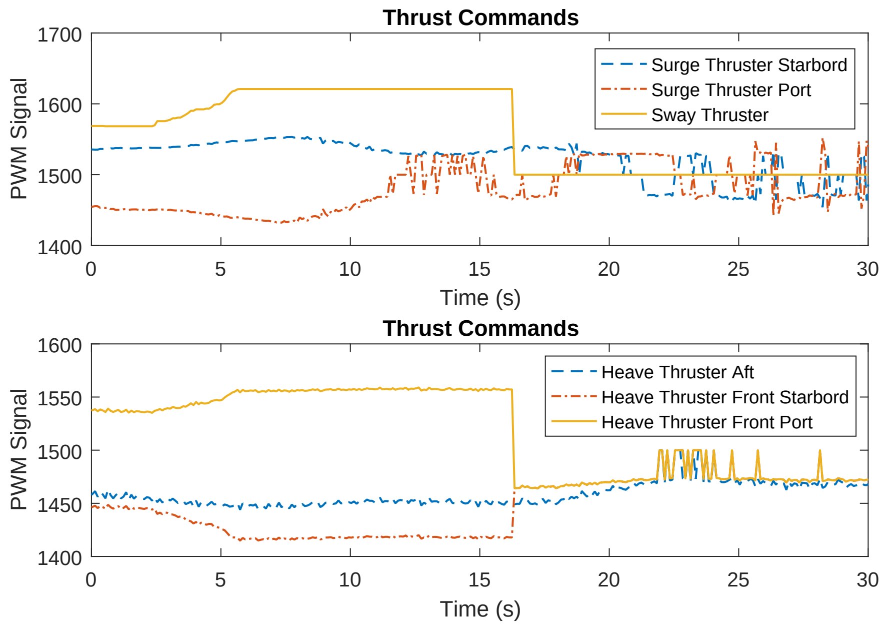


Figure 5.24: Sway thrust commands are dominating in Orbit mode.

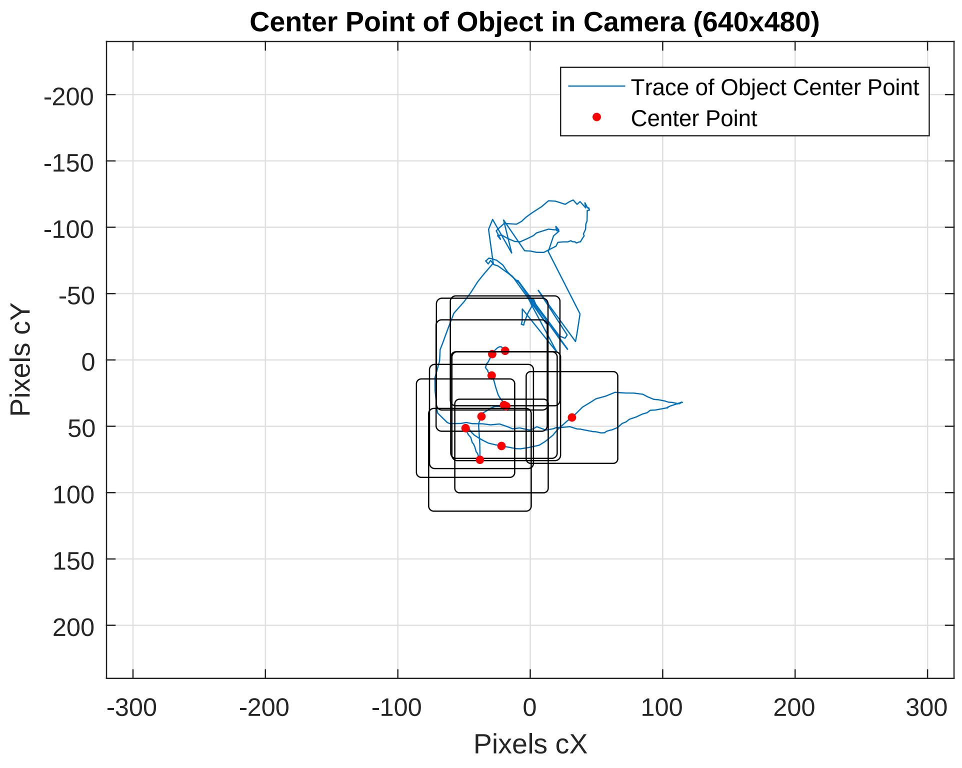


Figure 5.25: Movement of the feature inside the camera in Orbit Mode.

5.9 Discussion

The objective for the Camera-Assisted Positioning System was to control the ROV with automatic controllers, based on computer vision together with manual joystick control. The testing in MC-lab showed that the control problem was solved with high reliability. All controllers were easily tuned to give satisfactory results. These gains could, of course, be optimized to enhance response time and performance even further. However, the aim of this project was to develop a proof of concept on how computer vision can be integrated into low-cost ROVs.

Latency in the video transmission was not as big of a problem as first expected, but it could have probably removed the small oscillations in the Auto-heading control function if the latency was reduced. For that to be possible, a greater CPU should be considered for the ROV. That allows all computer vision to be running locally on the ROV. However, it was very convenient to have the image processing running topside for testing and debugging. As explained earlier, running image processing topside also allowed greater resolution on the image matrices, hence smoother output signals from the control function. If we were to run Feature Tracking on the RPi2, it could be significant to run at 320 x 240 pixels; this will reduce the CPU intensity, and maybe still have enough points for a smooth output. The steady state error would then work on 10 steps instead of 20 (ref. figure 5.19). The question is still, will it actually be faster than running on the topside computer?

The depth position in Orbit Mode got pushed far off set-point when the sway thruster was turned on. This could be solved by tuning the controller more aggressive, or decrease the allowed sway force in this mode. The thrust allocation can also be tuned for better precision for low velocities. Other factors are that when the camera is tilted in roll the center position Y of the feature will not be correct. This means that the auto-reference for depth will start to grow, and is responsible for the peak from 17 to 30 sec in heave position, seen in figure 5.21. The roll angle should, therefore, be considered before outputting the cY signal to the Auto-reference function. An alternative is to set cY to 0 when sway control force is larger than 5 N. That way the depth adjustments will only happen when the ROV is hovering on the spot.

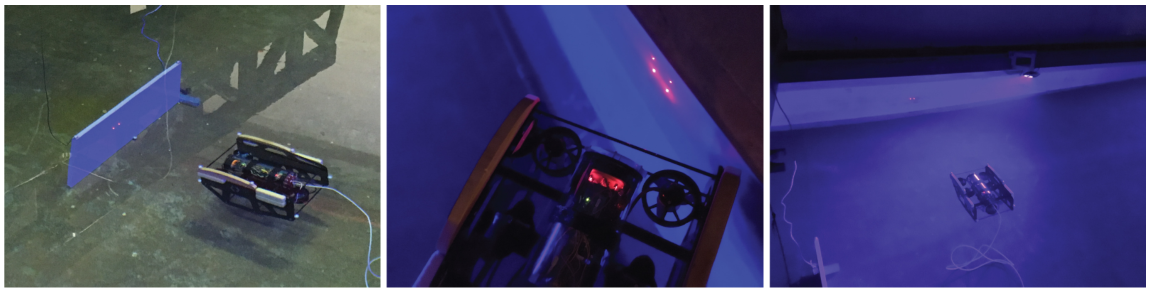
A weakness of the Feature Tracking algorithm is that it is only tracking features based on its color. This means it will easily get distracted if the same color shows up on a different object (i.e.

blue light from Qualisys cameras). This problem could be reduced by filtering out the contours that have a sudden jump in position and area from the previous one. Also features as shape could also be stored with more points than a rectangle; then it would be possible to compare the new contour with a desired shape. Shapes as squares and circles would be easy to find, but usually, there is not that kind of shapes in the ocean, at least not naturally. A more user-friendly way of selecting new features on the sea bottom is advised to be developed before this module can be implemented in a consumer product.

Overall, the performance of the feature tracking algorithm has been very satisfactory. The ROV uDrone has been functioning very well as a robust testing platform together with Qualisys positioning system in MC-lab. The splitting of DOFs was also successfully working with partial manual and partly automatic control.

Chapter 6

Camera-assisted Distance Control Based on Laser Lights



(a) Test setup for Range Finder with Qualisys. (b) Development of Range Finder. (c) Testing in low light conditions.

Figure 6.1: Range Finder in MC-lab.

6.1 Problem Formulation

Two parallel lasers pointing straight forward from the camera hits a wall in front of the ROV. The apparent distance between the reflected laser dots will then change when the ROV gets closer to the wall in front of it. This distance can be measured by computer vision and translated to a distance in cm from the ROV to the wall. The control objective is to maintain a constant distance to the wall while being able to control sway motions manually. The ROV should also be able to keep a desired depth and heading, depending on the operating mode. Hereafter we will call this laser module for "Range Finder". Four laser lights are mounted inside the ROV and create laser dots as shown in figure 6.1. The lasers can also be switched off at any time through the software.

6.2 Control Modes for Range Finder

Range Finder can also be utilized in different combinations with manual control as shown in figure 6.2

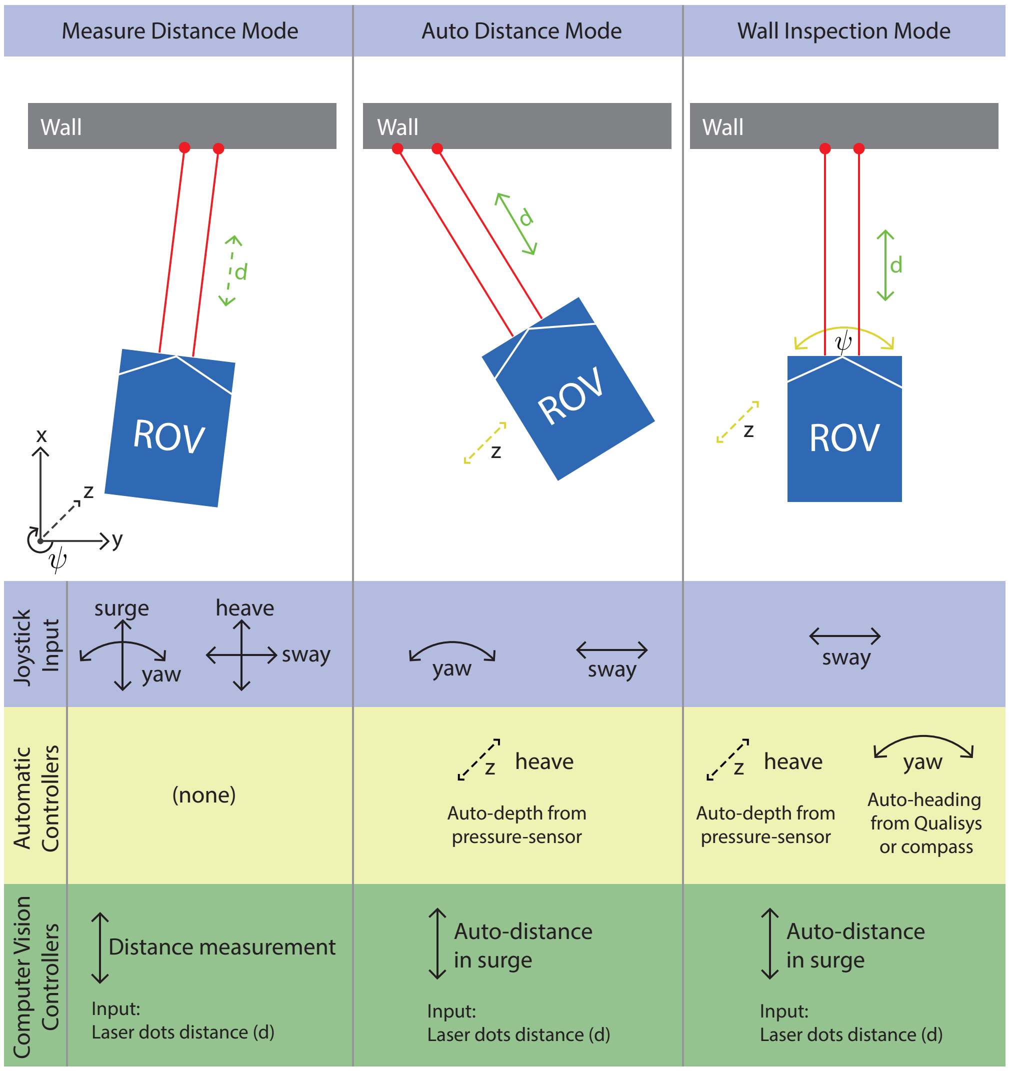


Figure 6.2: Control modes for the Range Finder module.

6.2.1 Measure Distance Mode

The first mode is just providing the distance as information to the user with no automatic control. This information is often very useful underwater, especially in new waters. Sometimes it can be very hard to get an impression of how big the objects in the camera really are.

6.2.2 Auto Distance Mode

Automatic Distance Mode keeps a constant distance to the object in front of the ROV. However, this mode is vulnerable if the ROV starts to yaw because then the distance to the wall will rapidly increase. But as long as the ROV does not continue to rotate in yaw, it will stabilize at the correct distance, even though it is not perpendicular to the wall. This mode gives the user a little more room to navigate while still have some guidance on controlling the ROV. This module could also be mounted as Auto-altitude if the lasers and a camera were pointed downwards. This could be implemented as an auto-reference function for depth as described earlier in chapter 5.

We have now added the Auto-depth controller, leaving yaw and sway motions controllable from the joystick.

6.3 Geometry of the Lasers

The distance to the wall in front of the ROV can be estimated by use of two parallel lasers. If the wall is far away, the distance between the laser dots will decrease. If the ROV gets closer to the wall, the distance between the dots will increase. As long as the lasers and the camera are fixed, it will be possible to find a mapping from the distance between the laser dots (pixels) and the distance from the ROV to the wall (cm). With only two parallel lasers it is not possible to know the angle of the wall relative to the ROV. This is because when the angle is changing, both laser dots will move to the same side, maintaining the same distance between the dots. The phenomenon is illustrated in figure 6.3. Angles up to 80 degrees was tested and verified in MC-lab. Angles above that was difficult to test because laser dot furthest away got hard to detect. However, this means that the Range Finder function will work on any surface, at any angle, as long as the material is reflecting the laser lights.

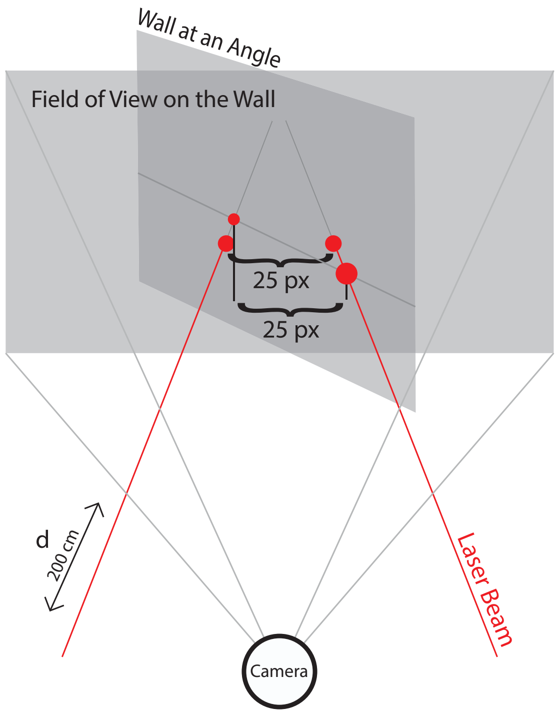


Figure 6.3: Laser geometry for Range Finder with two parallel lasers.

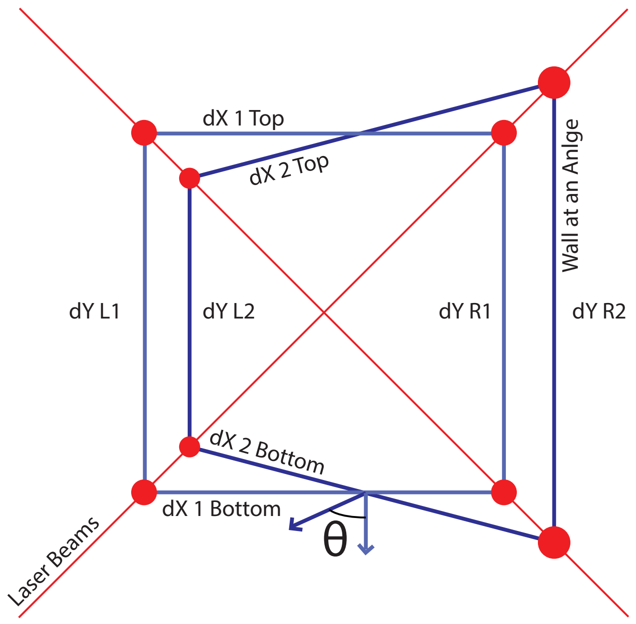


Figure 6.4: Proposed setup with four parallel lasers.

A method to detect the angle between the wall and ROV will now be proposed. If four parallel lasers are used instead of two, we get more distances to track, see figure 6.4. All the distances between the points can be compared to see how they are changing relative to each other. When distance dY L and dY R is found, it is possible to find the corresponding distance from the ROV to the right and the left side of the projected square. The angle θ can then be determined by the function:

$$f(dYL, dYR) = \arctan\left(\frac{dL - dR}{ParallelDistance}\right) = \theta \tag{6.1}$$

where θ is the angle between the ROV and the wall, dL and dR are the distances from the ROV to the wall, and Parallel Distance is the distance between the laser beams. For ROV uDrone this distance is 5.5 cm.

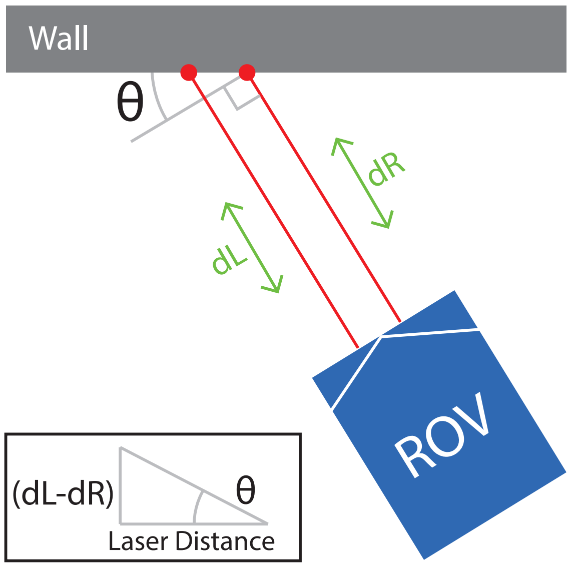
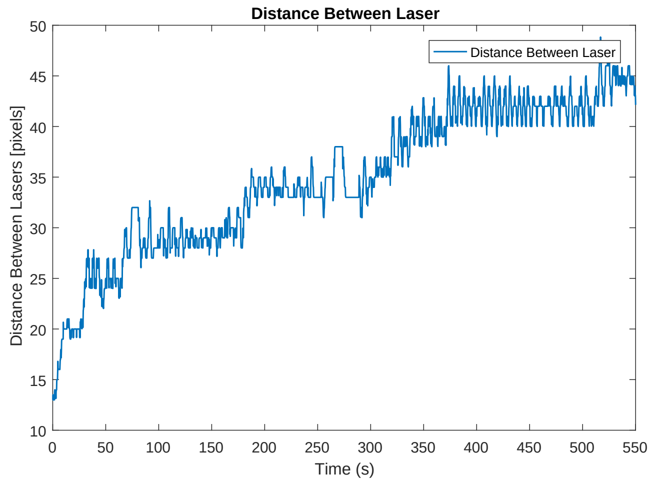


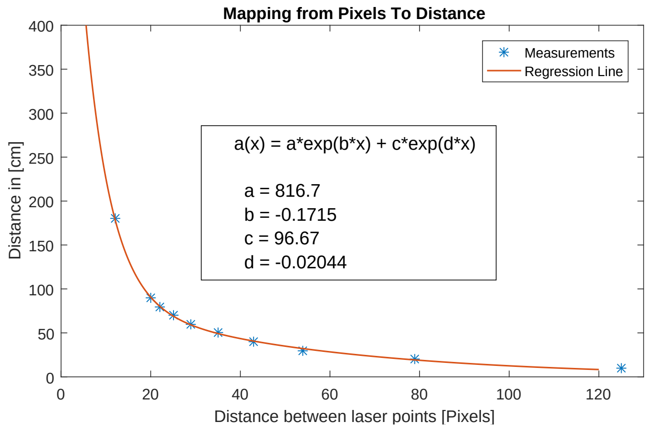
Figure 6.5: Detecting the ROVs heading relative to a wall.

6.4 Control Design and Signal Processing

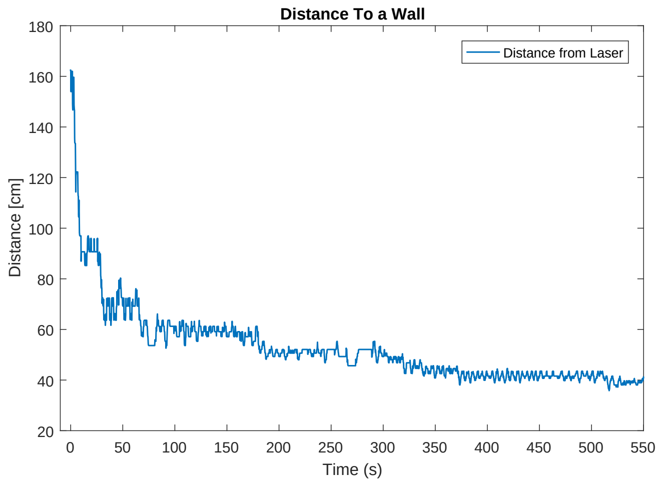
The Auto-distance controller is implemented as regular PID controller with distance to the wall as input and control force in surge as output. Low-pass filtering is applied in the Range Finder function as described in section 5.4.2. When the Range Finder has found how many pixels we have between the two dots, the number is mapped into a distance from the ROV to the wall. This relationship was done by plotting the pixels for every 10 cm between 10 and 180 cm. The fitted curve was then implemented in the Range Finder function to give the distance in cm right away. The conversion from pixels to distance is found in figure 6.6.



(a) Distance between laser dots.



(b) Mapping from pixels to distance in cm.



(c) Distance from ROV to object in front.

Figure 6.6: Mapping between laser dot distance in pixels and distance in cm.

6.5 Implementation and Laser Detection

Detecting lasers can be both easy and hard, depending on how powerful the lasers are. There are different approaches to get the reflected dots filtered out from the video frame. But after testing different algorithms, it was found that using the algorithm for Feature Tracking gave the best results. This algorithm was also much faster to implement since it was already done for the Feature Tracking. However, there was quite a bit of tweaking to get the software (OpenCV 2.4 vs 3.0) running on the Raspberry Pi 2. So the only difference in the algorithm is that we are now tracking the red color of the laser, filtered by a size constraint. The size of the dot will change some depending on the distance, but it is still within a range of $3-40px^2$. So if there is a larger red object, it will not disturb the algorithm. As Range Finder module is implemented locally on the Raspberry Pi 2, it allows connecting directly to the RPi2 camera, which gave some latency benefits. This camera is used only for laser detection, so the brightness is turned down to filter out some of the surroundings. The RPi2 camera also has a much narrower field of view, which gives more pixels in the area where the laser dots gets reflected. As discussed earlier, we have a problem with high resolution on the RPi2 because the matrices get to large and things slow down. To improve the resolution in the are of interest, the camera was therefore set to crop the image before sending it to the Range Finder function. This is done by a function in the PiCamera library for Python and OpenCV, shown in equation 6.2 and 6.3.

```
camera.crop = (0.15, 0.15, 0.85, 0.85)                (6.2)
```

The vector in the crop function tells how many percents of the image that should be cropped out in the format (start point x, start point y, end point x, end point y). This cropping leaves a padding of 15 % cropped out before the image is grabbed from the camera.

```
raw = PiRGBArray(camera, size=(320,240))              (6.3)
```

Equation 6.3 grabs an image from the camera stream with resolution 320 by 240 pixels. This gives us a small matrix with adequate steps for the laser dots variation.

6.6 Delay in the Control Loop

A simple test to check the response time was done by switching on the lasers, and measure how long it takes before the Range Finder gives a distance measurement, and then how long it takes for the Auto-distance controller to give a force command. The switch for the lasers are running at 100 Hz and is publishing directly to the Arduino. The corresponding delay for switching is therefore neglected. While logging all signals on the same clock, we can then see how much time each step of the sequence takes. The result is shown in figure 6.7. It takes 440 ms to get a distance measurement, and 500 ms to get the control force. The Range Finder node in ROS runs at about 7 to 10 Hz, which is more than fast enough for dynamics in this system at low velocities. However, the delay could be reduced for better control performance.

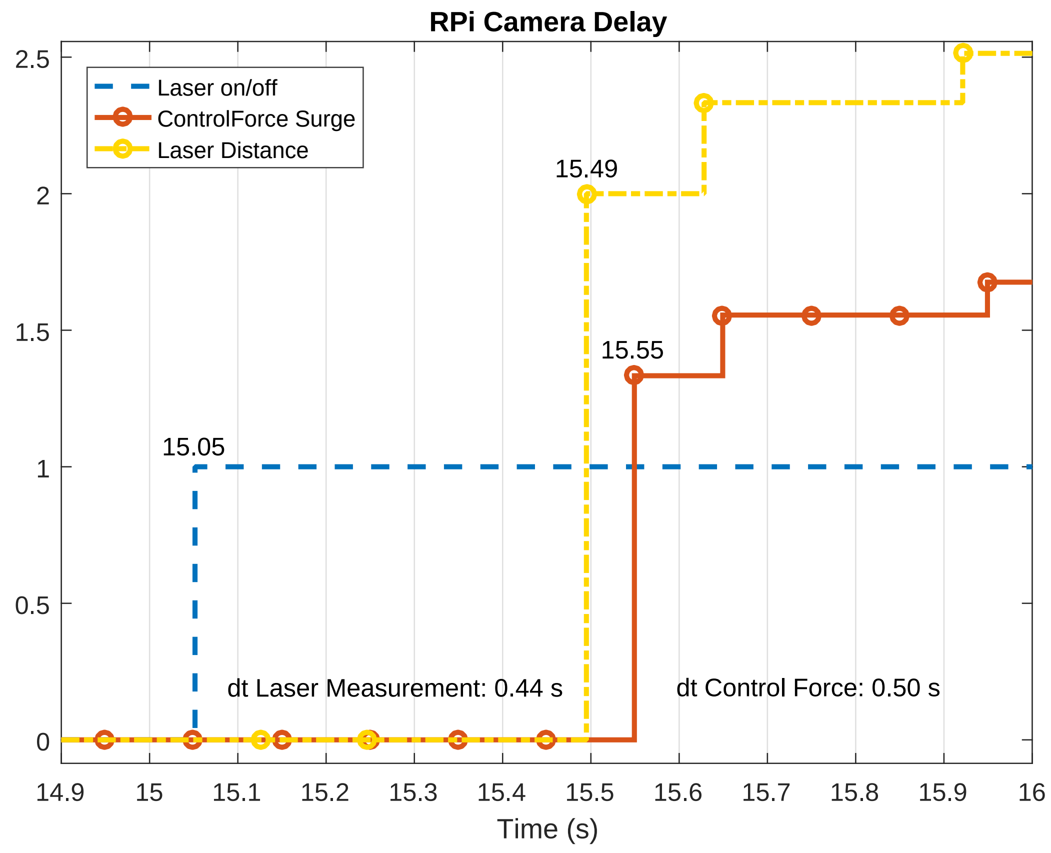
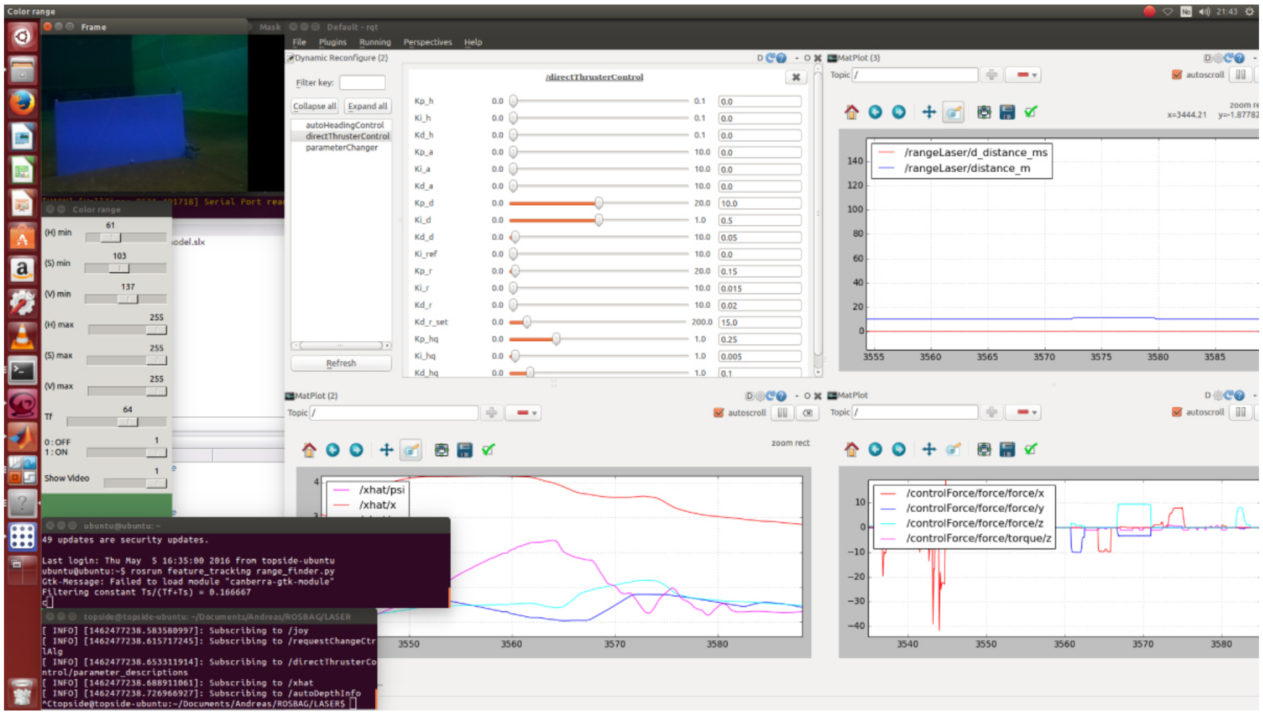


Figure 6.7: Delay in Range Finder with RPi Camera.

6.7 Graphical User Interface

The graphical user interface is based on the same as the Feature Tracking module, but since it is running on RPi2 we have to connect via *SSH* with the *-X* flag to be able the see the graphical interface. Displaying graphic and especially video over SSH is not the fastest, but it enables the user to tune in the right color limits before the video displays are turned off. When color limits are set, the Range Finder only publishes the distance, without displaying any video to save computational power. The interface is shown in figure 6.8.



Color range

(H) min	61
(S) min	103
(V) min	137
(H) max	255
(S) max	255
(V) max	255
Tf	64
0 : OFF 1 : ON	1
Show Video	1

/directThrusterControl

Kp_h	0.0	0.1	0.0
Ki_h	0.0	0.1	0.0
Kd_h	0.0	0.1	0.0
Kp_a	0.0	10.0	0.0
Ki_a	0.0	10.0	0.0
Kd_a	0.0	10.0	0.0
Kp_d	0.0	20.0	10.0
Ki_d	0.0	1.0	0.5
Kd_d	0.0	10.0	0.05
Ki_ref	0.0	10.0	0.0
Kp_r	0.0	20.0	0.15
Ki_r	0.0	10.0	0.015
Kd_r	0.0	10.0	0.02
Kd_r_set	0.0	200.0	15.0
Kp_hq	0.0	1.0	0.25
Ki_hq	0.0	1.0	0.005
Kd_hq	0.0	1.0	0.1

```
ubuntu@ubuntu: ~
49 updates are security updates.

Last login: Thu May  5 16:35:00 2016 from topside-ubuntu
ubuntu@ubuntu:~$ rosrun feature_tracking range_finder.py
Gtk-Message: Failed to load module "canberra-gtk-module"
Filtering constant Ts/(Tf+Ts) = 0.166667
c
```

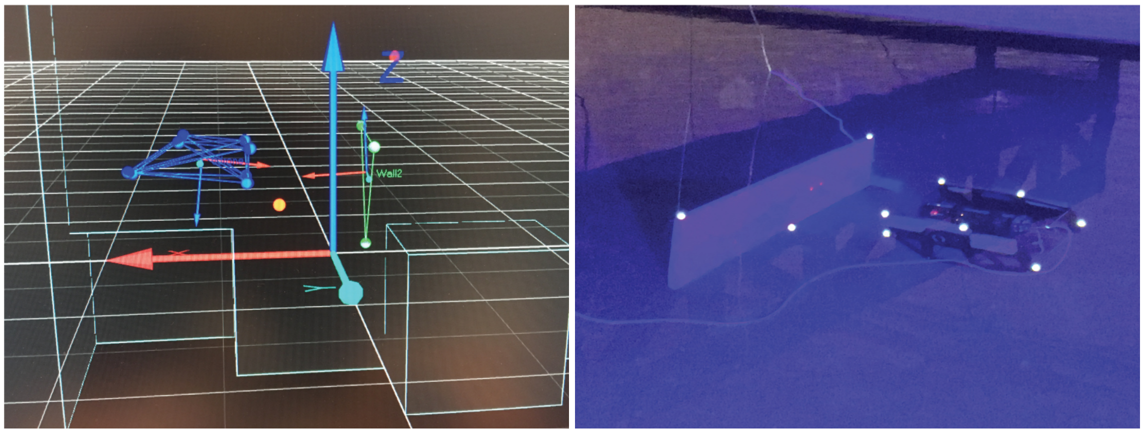
```
topside@topside-ubuntu: ~/Documents/Andreas/ROSBAG/LASER
[ INFO] [1462477238.583580997]: Subscribing to /joy
[ INFO] [1462477238.615717245]: Subscribing to /requestChangeCtr
lAlg
[ INFO] [1462477238.653311914]: Subscribing to /directThrusterCo
ntrol/parameter_descriptions
[ INFO] [1462477238.688911061]: Subscribing to /xhat
[ INFO] [1462477238.726966927]: Subscribing to /autoDepthInfo
^Ctopside@topside-ubuntu:~/Documents/Andreas/ROSBAG/LASER$
```

Figure 6.8: User interface when operating Range Finer.

6.8 Testing of CADC Based on Laser Lights

From the Range Finder algorithm, two test cases are chosen for a closer presentation. The first one is used to measure the "true" distance accurately from Qualisys versus estimated distance from the Range Finder function. While the second test is testing the Auto-distance controller together with a manual control input to see how they can be combined in inspection missions. Both tests are done in what we previously defined as the "Wall Inspection Mode" in figure 6.2.

This is because it was challenging to control the heading and depth manually, to keep the lasers on the small testing wall in the basin.



(a) Qualisys 3D view. (b) Reflection balls for positioning lit up.

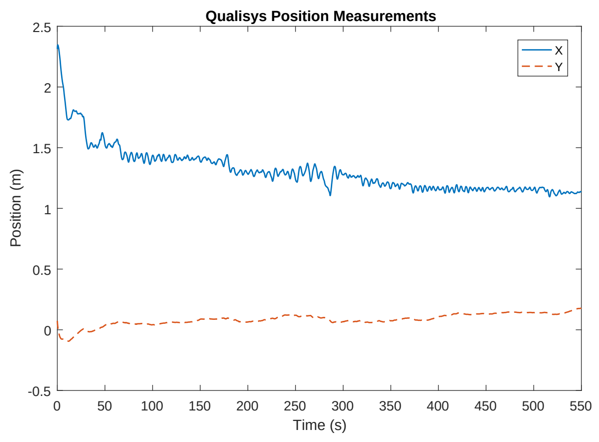
Figure 6.9: Qualisys positioning system in MC-lab. (a) shows the ROV in blue and the wall in green.

6.8.1 Test 1: Wall Inspection Mode

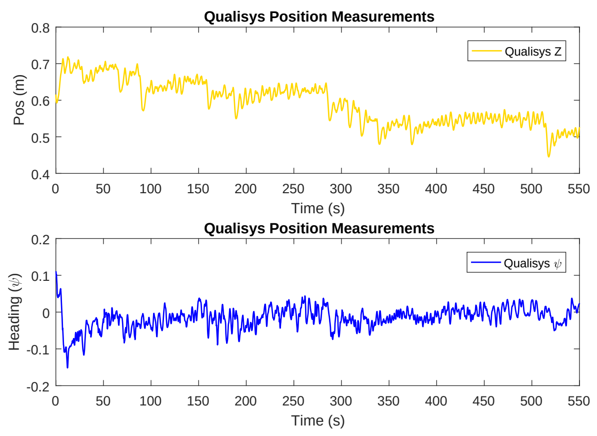
The Auto-distance control function will now be tested together with Auto-depth (pressure sensor) and Auto-heading (Qualisys). To get accurate measurements on the distance variation, a wall with positioning balls were lowered into the basin at MC-lab. With this setup, it is possible to get position measurements on both the ROV and the wall at the same time, in the same reference system. The bodies can be seen in figure 6.9.

The ROV is set to approach the wall at certain increments to test the accuracy the hole mapping range. The depth position is varying at steps as well. This was done manually to make sure the lasers would stay on the wall. Every time the Auto-distance button is hit on the x-box joystick, the desired depth is set to the current depth. The Auto-heading controller is regulating the heading to zero with input from the Qualisys system. All position measurements for the ROV can be seen in figure 6.10.

In figure 6.11 the motion of the ROV is presented together with estimates of the distance from the Range Finder function (shown as red stars). We see that it is "guessing" within +/- 10 cm of the actual distance, and for the most part, even better. In green, we see the trace of the



(a) X and Y position measurement.



(b) Z and ψ position measurement.

Figure 6.10: Position measurement of the ROV from Qualisys positioning system in MC-lab.

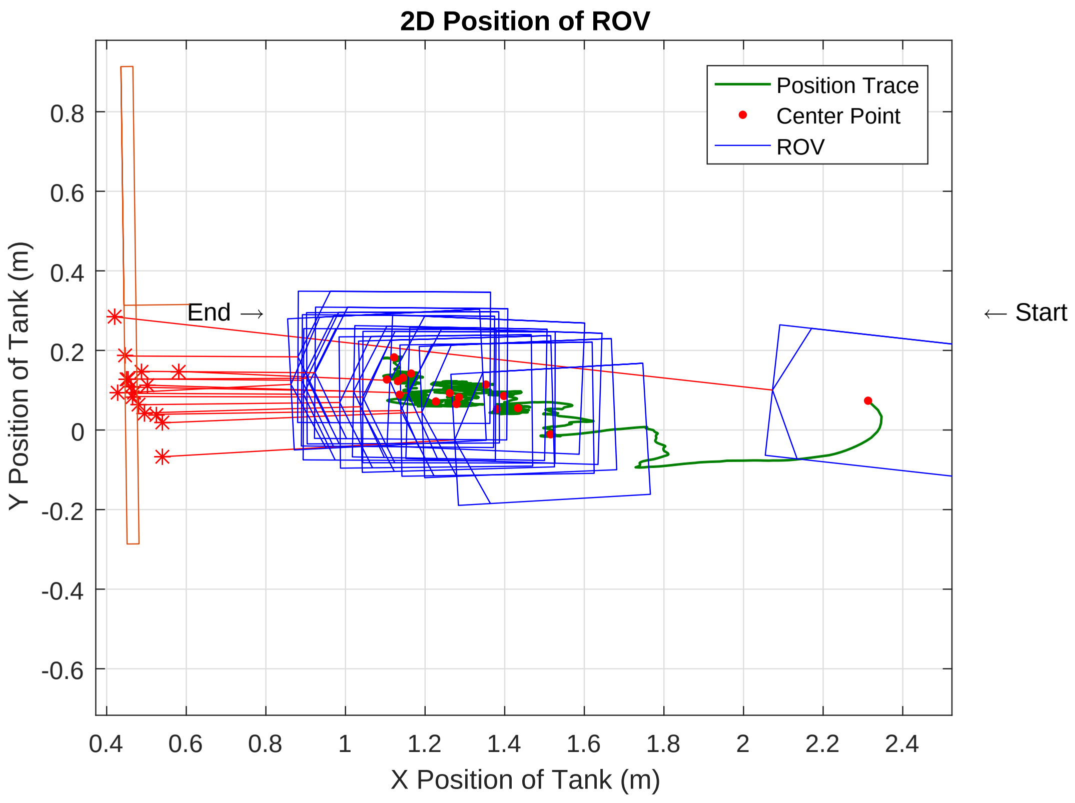


Figure 6.11: Position of the ROV in the basin seen from above.

ROV CG, and the blue ROV gives an impression on how long it has spent at each location. The ROV is plotted with constant time intervals throughout the time series.

The reference for the distance is plotted together with the measured position from Qualisys

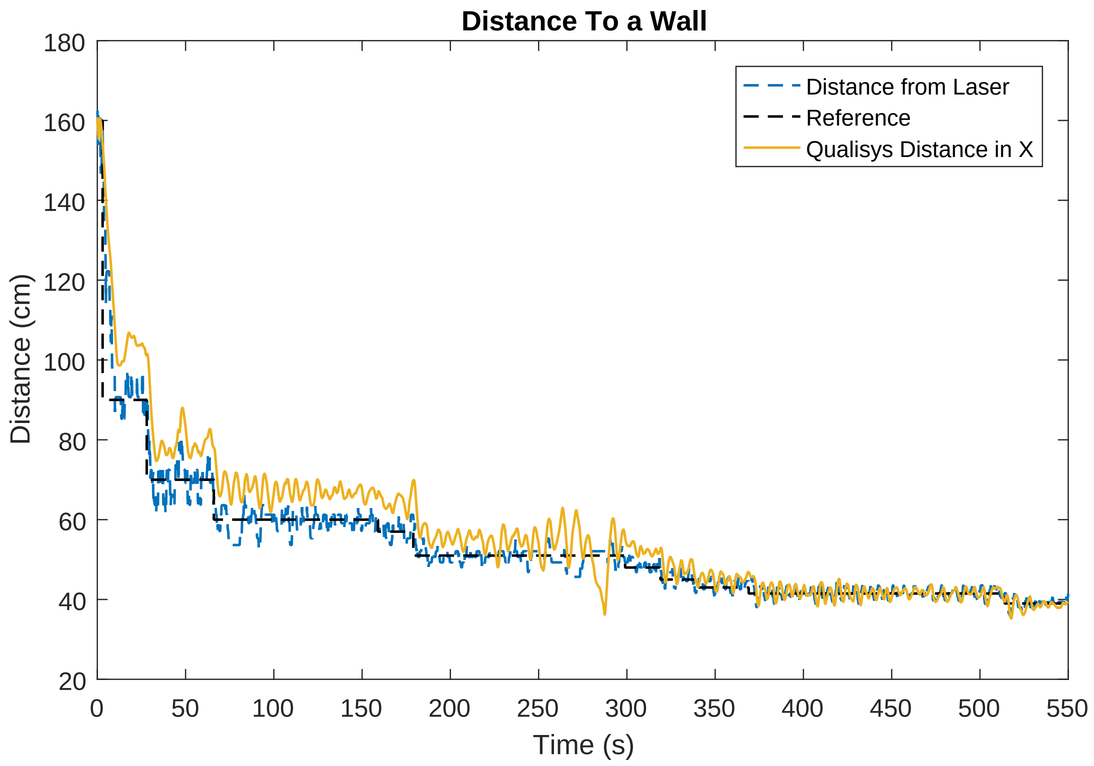


Figure 6.12: Distance to the wall with reference steps.

and estimated distance from the Range Finder in figure 6.12. We see that the Auto-distance controller is following nicely, but the distance estimate is off by about 15 cm in the time interval from 0 to 170 seconds. The errors between the signals are plotted in figure 6.13. Again, we see that the error is greatest in the same interval. From 300 seconds and out, the error goes to zero. This means the Range Finder is most accurate for shorter distances. Since the difference between estimated distance and the reference is zero-mean, we know that main contribution for the error must be the mapping from pixels to cm. It is also interesting to see how the noise gets lower and lower as the distance decrease. Two things cause this, for longer distances, the error will also be greater in magnitude, but for shorter distances, the number of pixels will also increase. Hence, we get a higher resolution on the shorter distances. If we recall figure 6.6, we can see that the measured amount of pixels starts at 13 pixels and ends up at 45 pixels during this time sires. This means that a jump from one integer to next is a relatively smaller step, hence a smoother measurement signal for closer distances. The resolution of the camera can, therefore, be increased for smoother performance.

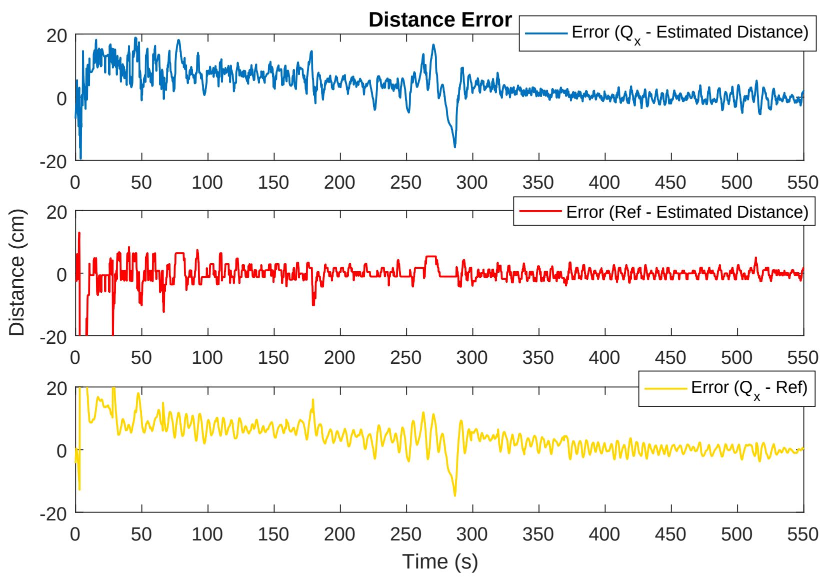


Figure 6.13: Comparison for the errors in distance to the wall.

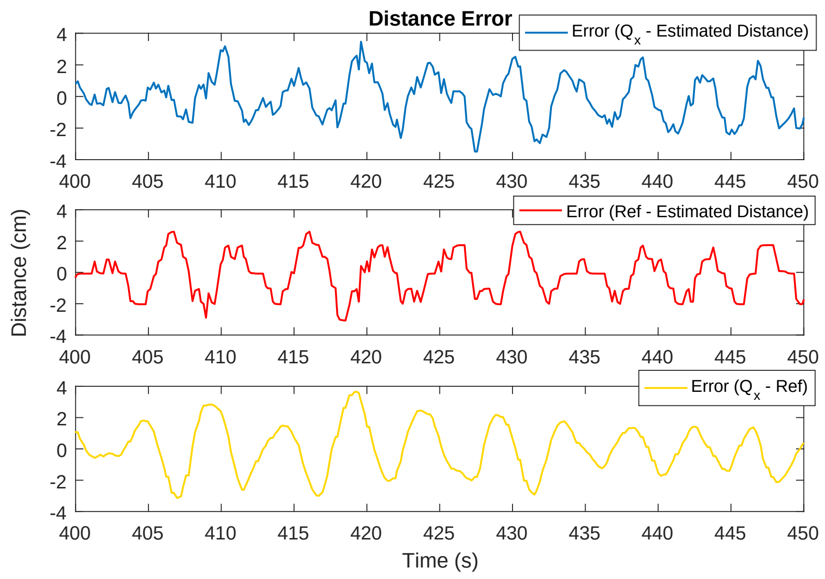


Figure 6.14: Zoomed in to see the steady state oscillations.

A close up view of 50 seconds in the steady-state interval is presented in figure 6.14. All the three errors are staying well within +/- 3 cm. We can also see that the position measured from Qualisys is less noisy than the estimated distance from the Range Finder, even though the main oscillations are captured well.

The control forces from 100 - 150 seconds are shown in figure 6.15. We see Auto-distance and Auto-heading have mean control force close to zero with low peaks, while Auto-depth has

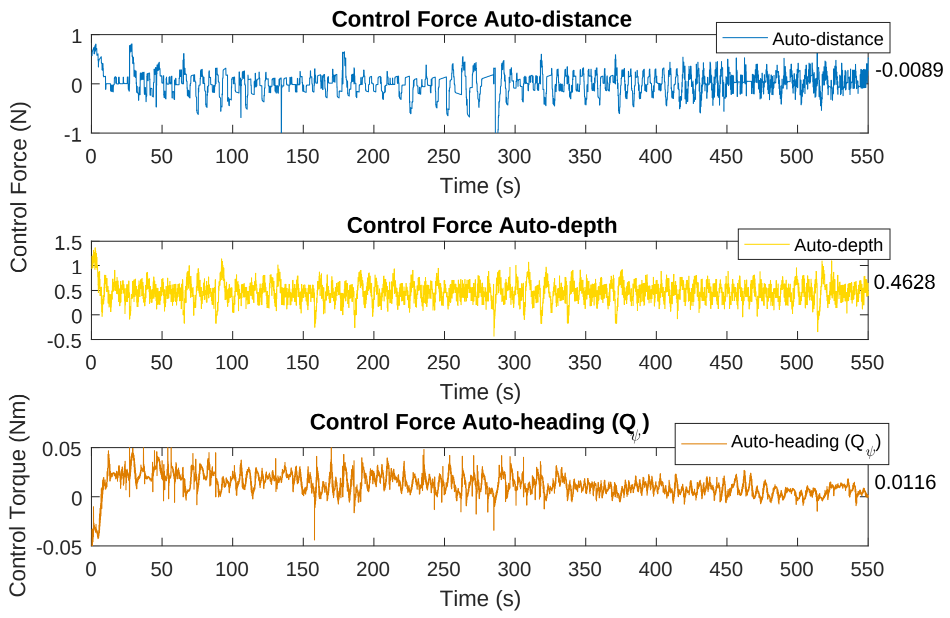


Figure 6.15: Control Forces for Automatic controller in Wall Inspection Mode.

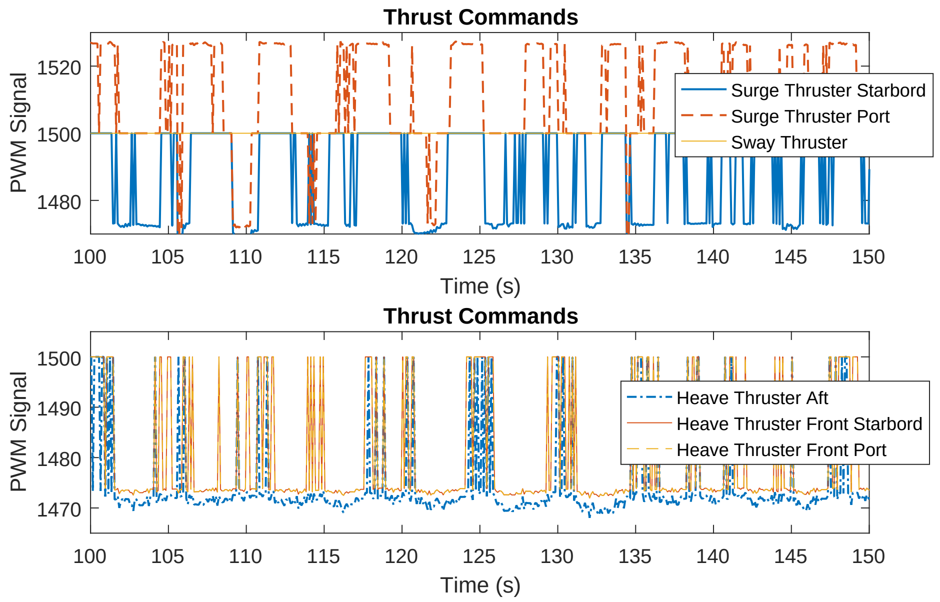


Figure 6.16: Commanded thrust signals, showing only 50 seconds.

a mean close to 0.5 N. Since the peaks for Auto-heading are much smaller, the mean value of 0.0116 will contribute to a summed clockwise rotation of the vehicle. This is most likely a steady-state bias from the umbilical that is compensated for by the controller. This is very visible in the thrust commands from the thrust allocation in figure 6.16, where the port thruster moves forward, and the starboard thruster moves backward. The thrusters are working on really low RPM, which makes them turn and stop all the time. The heave thrusters in the next row are running more steady to compensate for the buoyancy of the vehicle and provide that 0.5 N from the Auto-depth controller. Also, this time, it is possible to see that the aft thruster is working more than the front ones. In some areas, like around 115 seconds, we see that both thrusters are backing up, due to the Auto-distance controller. As mentioned earlier, since the thrusters have a dead-zone of 25 μs, the thrust allocation has to add this dead-zone gap imminently to turn the thruster at low control forces.

6.8.2 Test 2: Wall Inspection Mode with Manual Input

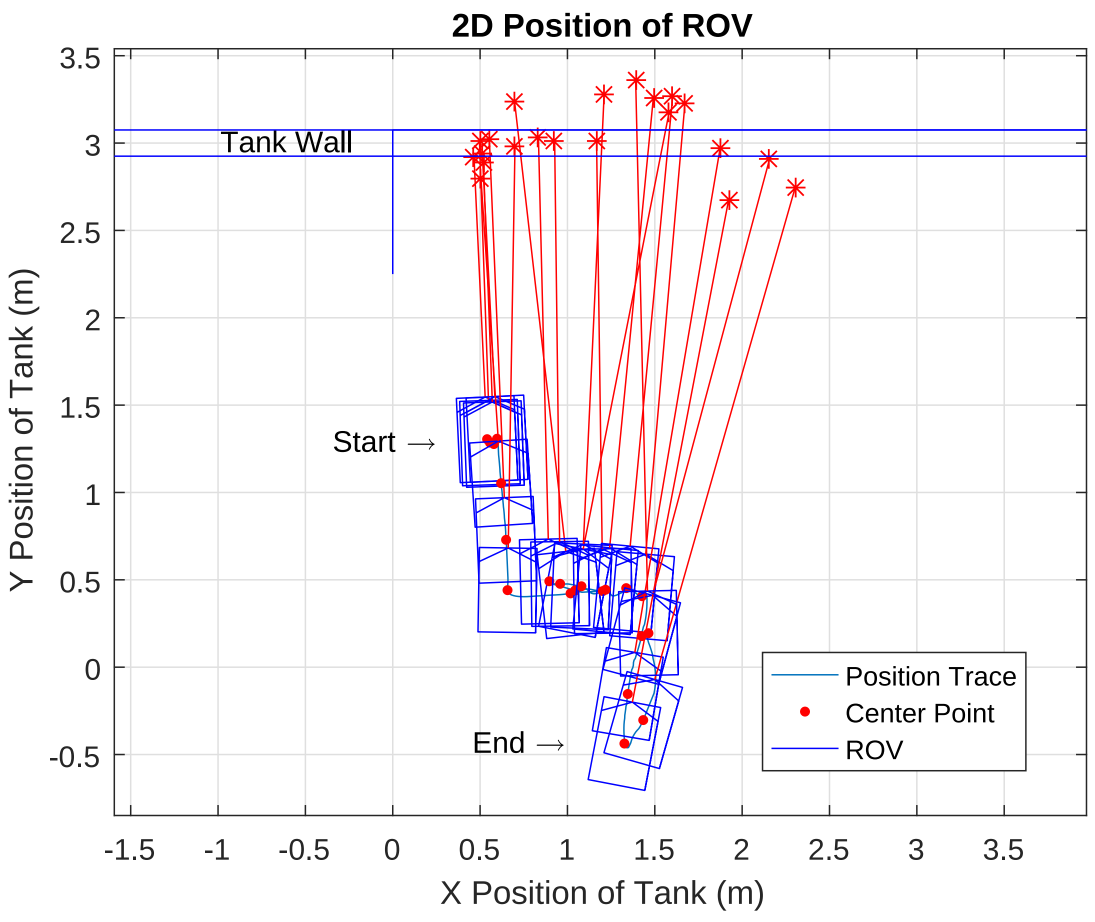


Figure 6.17: Wall Inspection Mode, showing how the ROV can hold distance while moving sideways.

After the Auto-distance controller had been verified, it was time to test the Range Finder with manual input as well. The ROV was also set to be working further away from the wall to see how good it could cope with longer distances while the lasers get significantly harder to detect in the video image. The basin in MC-lab is 6 meters wide, which means there are approximately 3 meters to the side wall. This test will not be as accurate as the previous test, but still of interest. The ROV starts at about 1.5 meters from the wall and begins to move backward incrementally with the Auto-distance controller. Once its 2.5 meters from the wall, a manual sway force of 10 N is applied to the right. In figure 6.17 we can see that the ROV maintains the distance of 2.5 meters while Auto-heading is holding the heading towards the wall. In the end, we continue backward to see how far the lasers reach, and we end up at about 3.5 meters. This distance is actually outside the measured range done to create the mapping shown in figure 6.6, but is still giving good measurements, with 20 cm off on 3.5 meters (corresponds to 6 % error).

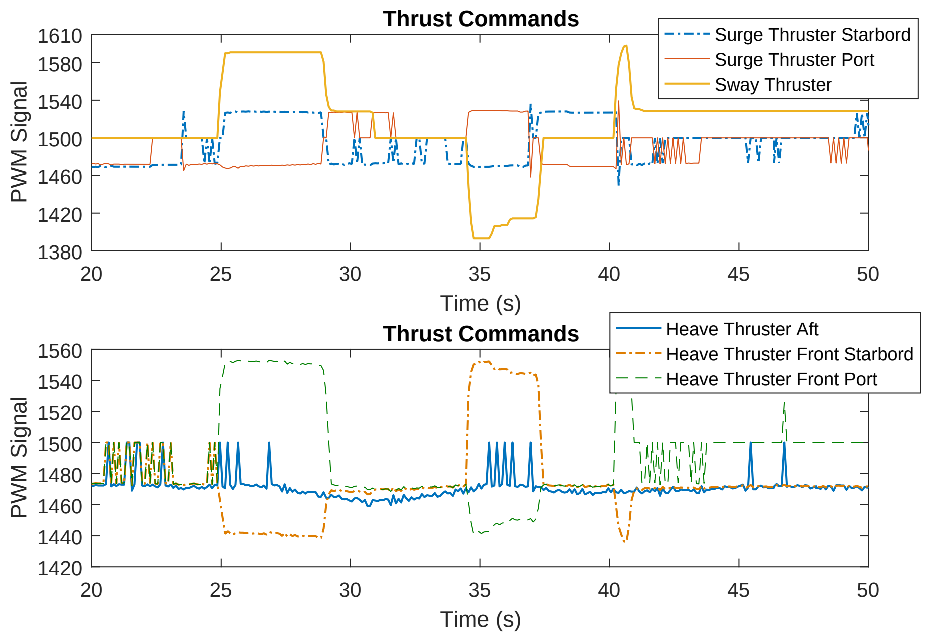


Figure 6.18: Manual input in sway while other thrusters are controlled automatically.

In figure 6.18 it is clear that the sway force is much higher than the automatic control forces. Mark how the front heave thrusters are compensating for the roll motion induced by the sway thruster.

6.9 Discussion

The Range Finder function was performing very well and was very reliable, the filtering of the lasers was done easily with the HSV filtering interface through "SSH -X" for different light conditions. Installation of the lasers was more difficult, especially alignment to get the perfectly parallel. The plan was to align all four lasers, but the intensity was so different on the lasers, so it was hard to detect all of them at once. Better lasers should, therefore, be considered for further development. This will also increase the range of the Range Finder. Also, the accuracy of the pixel measurement was not able to detect the small differences that would be on the left and right side to compute the angle to the wall. The method proposed for estimating the angle to the wall should be fully executable, but it will require much higher image resolution in the operating area of the laser beams. It would be ideal to increase the distance between the lasers as well. This will then require external lights mounted on the outside of the body frame, which will require some more effort. The accuracy was also a bit too low for optimal control performance. The control problem was solved, but it should be possible to control the ROV with even smaller oscillations and errors if the resolution of the camera were to be increased and the delay was reduced. Maybe it is worth running Range Finder topside instead, to increase the resolution for the images in OpenCV. When comparing the delay tests for Range Finder and Feature Tracking, we see that it is actually faster to send the video to the topside computer, which also enables higher resolution. However, if the ROV is going to towards AUV mode it will be recommended with a more powerful on-board computer. The mapping from pixels to cm could also have been done over, based on the measured data shown in figure 6.12, that is more accurate than the initial manual measurements.

Overall, the Range Finder was functioning over all expectations and did also handle sharp angles to the wall better than expected. I, personally, think Range Finder and Auto-distance control has a great potential in the consumer market for ROVs.

Chapter 7

Conclusion with Recommendations for Further Work

7.1 Conclusion

Camera based control modes for the ROV uDrone were successfully developed and tested extensively in MC-lab. An object was tracked by a proposed Feature Tracking module that reduced the control effort for the user. The feature (color) was tracked successfully and never left the cameras field of view, even with high manual control input. All PID-controllers, Auto-heading, Auto-distance, Auto-depth, and Auto-heading based on Qualisys, were implemented and tuned with satisfactory results. An Auto-reference function for the depth controller was also working surprisingly well, keeping the object in the center of the camera frame vertically. All controllers were tuned to give smooth motions to reduce the risk of motion sickness when watching the video stream. However, further work on reduction of the camera delay and tuning of controller gains may improve the performance of transients even further.

When the delay for the two cameras was tested, it was shown that it was faster to stream the video to the topside computer before doing the image manipulations. Delay for the fish-eye camera was measured to 324 ms, while the delay for the RPi Camera reached 440 ms (at half the resolution). A stronger CPU should, therefore, be considered in the ROV uDrone for further development of computer vision and autonomous functionalities.

Two cameras and four laser lights were mounted on a customized 3D-printed bracket to keep the number of uncertainties to a minimum. The bracket was very helpful when opening the ROV (for changing batteries etc.), to make sure the camera and laser angles did not change from test to test. The laser lights were working quite ok, even though they could have been brighter and more equal in brightness. The pixel to distance mapping should also be re-calibrated to get more accurate estimates. However, the Range Finder with the Auto-distance controller did solve the control problem with satisfactory coherence with the Qualisys positioning system in MC-lab. The distance estimates were off by 6 % at 3.5 meters. This precision is considered to be acceptable for recreational underwater drones.

A set of thruster tests were executed to increase control performance. The results seemed to be good enough for low velocities. However, for higher speeds it is recommended to tune the thrustallocation even further.

Overall, computer vision is a low-cost solution for recreational ROVs to aid the user in maneuvering the ROV. The Feature Tracking module also reduces the motions in the camera when recording video. Range Finder provides valuable information for the distance, which is nice to have when entering new environments. Computer vision does definitely provide new and useful functionality for underwater drones, and it was successfully applied in this thesis. I also believe that many more tasks also can be solved with computer vision in the future development of low-cost ROVs.

A summarized video from testing in MC-lab is uploaded to YouTube and is available on: https://youtu.be/xKo3VHelcIk

7.2 Recommendations for Further Work

Hopefully will the contribution of this work be valuable in further development of computer vision in low-cost ROVs. In this section, recommendations for further work and ideas will be discussed.

7.2.1 ROV uDrone

- As discussed earlier, there is a demand for reducing the delay of the camera stream to OpenCV. The best way of reducing this is most likely to find a new on-bard computer with greater computing power; that is compatible with Ubuntu. However, it is still recommended to develop on the topside computer, before implementing complete modules on the ROV. This is because you much easier access to files and motoring.
- Today the electronic tube must be opened to unplug the batteries, because there is no main switch. A magnetic power switch is, therefore, advised for safety and less wear and tear on the o-rings.
- Further tuning on the thrust allocation is recommended in order to get better control performance.
- Install a proper IMU to get reliable pose estimates. This will be important for further development of sensor fusion etc.
- Install new lasers that are more powerful and easier to adjust to the correct angle. If possible mount the lasers further apart to increase the amount of pixels between the laser dots. This will increase the resolution of the distance measurement.

7.2.2 Feature Tracking module

- The Feature Tracking module should be more robust towards objects that look similar to the one that is being tracked. The contours should therefore be more detailed than just a rectangle approximation in order to validate the interference of a new object.

- A better solution for selecting a new object to track is of interest before the module is ready for the consumer market. This could be done with an algorithm that automatically finds possible features in the scene. The possible features could then appear as buttons on a touchscreen, and give you the options of the different control modes.

- Tracking of multiple objects at the same time could give good estimates for positions relative to the features. Hence, also give good velocity estimates. This information could also be merged with IMU for dead reckoning.

- Find a good way to map area in px^2 to m^2, hence the distance to the object in meters. This should be possible based on the good measurements from Auto-distance mode with area as input. Maybe the Range finder can be used to do find the ratio. This can be done by storing the area of the feature when the ROV is 1 meter away. Then it should be possible to know how small it should be 5 meters away, which is out of Range Finders range. That way we have extended the range for distance estimates.

- Camera calibration of the fisheye camera if distortion is a problem when converting to meters. This will be more important when objects are tracked away from the center of the camera, which will be important when tracking multiple objects at the same time.

- Object recognition could be interesting if we have data from known objects that we can recognize in the basin. In the long term a database for fish can be developed so the ROV can recognize a fish, a star fish or a crab?

- Utilization of GPU in OpenCV. OpenCV has support for GPU programming which can increase performance dramatically on the RPi2. This could be interesting to look further into as well.

7.2.3 Range Finder module

- Stronger lasers will increase the range and detectability in daylight.
- Increase the distance between the lasers to get higher accuracy. Might have to be mounted on the outside of the ROV.
- Consider a different color for the lasers, red is used today, and is a color that easily get absorbed in water.
- Is it possible to use lasers that are invisible to the human eye, with a hyper spectral camera that is able to detect them. This way the video will not be polluted by the laser beams.
- Is it possible to use a grid of laser lights for estimation of planes? This can be used for seabed mapping or SLAM.
- A laser line can be used to map the sea bead if it has an angle to the camera.

The opportunities are endless, just think how much a diver can do based on his vision under water. He remembers where he has been, and has learned since he was a child how to differentiate plants, fish, and spices from each other based on their unique features.

Appendix A

ROS Node Communication

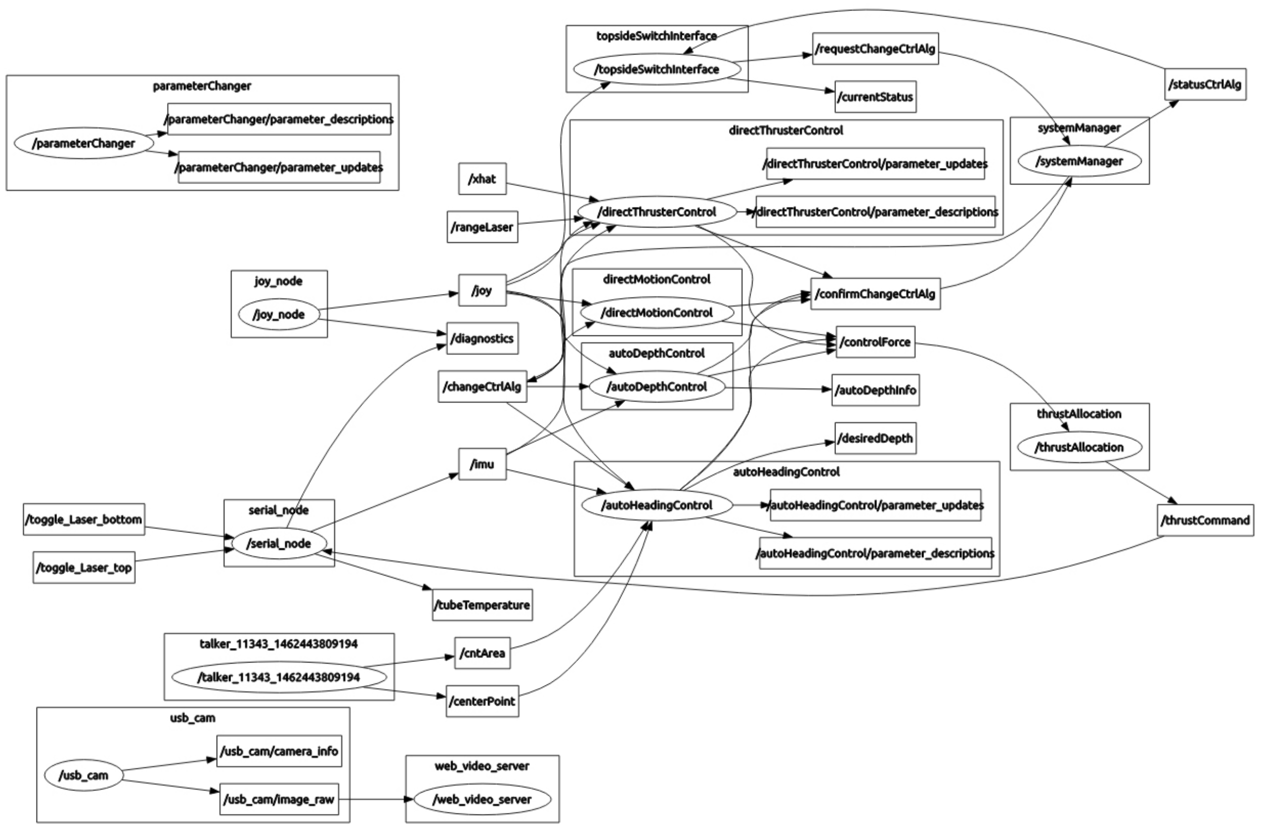


Figure A.1: Auto-generated map of communication between ROS nodes.

Bibliography

Bay, H., Tuytelaars, T. & Van Gool, L. (2006), Surf: Speeded up robust features, *in* '9th European Conference on Computer Vision'.

DeepFar (2016), 'White Shark MAX', `http://www.deepinfar.com/col.jsp?id=146`. [Online; accessed July-2016].

DeepTrekker (2016), 'Deep Trekker DTG2 Specifications', `https://deeptrekker.com/wp-content/uploads/2015/04/Deep-Trekker-DTG2-Specifications.pdf`. [Online; accessed July-2016].

FiFish (2016), 'FiFish Atlantis Specifications', `www.prnewswire.com/news-releases/rediscover-the-sea-meet-fifish-the-first-ever-consumer-market-oriented-smart-rov-by-t` `html`. [Online; accessed July-2016].

Forbes (2016), 'Business news and financial news', `http://www.forbes.com/sites/cfeng/2016/03/30/the-underwater-drone-manufacturer-that-wants-to-be-chinas-second-dji/#5564030476fc`. [Online; accessed July-2016].

Fossen, T. I. (2011), *Handbook of Marine Craft Hydrodynamics and Motion Control*, Wiley.

Henriksen, A. V. & Sandøy, S. (2015), *Hardware and Software Design of uDrone*, NTNU, Trondheim.

Heshmati-Alamdari, S., Eqtami, A., Karras, G. C., Dimarogonas, D. V. & Kyriakopoulos, K. J. (2014), A self-triggered visual servoing model predictive control scheme for under-actuated underwater robotic vehicles, *in* 'Robotics and Automation (ICRA), 2014 IEEE International Conference on', IEEE, pp. 3826–3831.

Hu, Y., Zhao, W., Wang, L. & Jia, Y. (2009), Underwater target following with a vision-based autonomous robotic fish, *in* '2009 American Control Conference', pp. 5265–5270.

Karras, G. C. & Kyriakopoulos, K. J. (2008), Visual servo control of an underwater vehicle using a laser vision system, *in* '2008 IEEE/RSJ International Conference on Intelligent Robots and Systems', pp. 4116–4122.

Karras, G. C., Loizou, S. G. & Kyriakopoulos, K. J. (2010), A visual-servoing scheme for semi-autonomous operation of an underwater robotic vehicle using an imu and a laser vision system, *in* 'Robotics and Automation (ICRA), 2010 IEEE International Conference on', pp. 5262–5267.

Karras, G. C., Panagou, D. J. & Kyriakopoulos, K. J. (2006), Target-referenced localization of an underwater vehicle using a laser-based vision system, *in* 'OCEANS 2006', pp. 1–6.

Lowe, D. G. (2004), Distinctive image features from scale-invariant keypoints, *in* 'International Journal of Computer Vision', pp. 91–110.

Narimani, M., Nazem, S. & Loueipour, M. (2009), Robotics vision-based system for an underwater pipeline and cable tracker, *in* 'OCEANS 2009 - EUROPE', pp. 1–6.

NOAA (2014), 'How much of the ocean have we explored?', http://oceanservice.noaa.gov/facts/exploration.html. [Online; accessed July-2016].

OpenCV (2013*a*), 'Feature Detection and Description', http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_table_of_contents_feature2d/py_table_of_contents_feature2d.html. [Online; accessed July-2016].

OpenCV (2013*b*), 'Pose Estimation', http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_pose/py_pose.html. [Online; accessed July-2016].

OpenCV (2013*c*), 'Thresholding in OpenCV', http://docs.opencv.org/2.4/doc/tutorials/imgproc/threshold/threshold.html. [Online; accessed July-2016].

OpenROV (2016), 'OpenROV Trident Specifications', http://www.openrov.com/products/1-trident.html. [Online; accessed July-2016].

Raabe, C., Henell, D., Saad, E. & Vian, J. (2014), Aggressive navigation using high-speed natural feature point tracking, *in* '2014 IEEE Aerospace Conference', pp. 1–13.

ROS (2015), 'Web Video Server', `http://wiki.ros.org/web_video_server`. [Online; accessed July-2016].

Rosebrock, A. (2014), *Image Search Engine Resource Guide*, PyimageSearch.

Shi, J. F., Hua, C. J. & Li, G. H. (2010), A simplifying method of vision attention simulating human vision in machine vision system, *in* '2010 International Conference on Machine Learning and Cybernetics', Vol. 6, pp. 3097–3100.

Zhang, B. (2010), Computer vision vs. human vision, *in* 'Cognitive Informatics (ICCI), 2010 9th IEEE International Conference on'.