Camera-based SLAM for Dynamic Positioning of Low-cost ROV

Tore Mo-Bjørkelund

Marine Technology
Submission date: June 2017
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:	Mo-Bjørkelund, Tore
Field of study:	Marine control engineering
Thesis title (Norwegian):	Kamera-basert SLAM for dynamisk posisjonering av lavkost ROV
Thesis title (English):	Camera-based SLAM for dynamic positioning of low-cost ROV

Background
As the interest in subsea exploration increases, the demand for cheap exploration vehicles has increased. The underwater environment can be chaotic and difficult to navigate in, so any help the user can get for maneuvering the vehicle is welcomed. A limitation of low-cost ROVs is the lack of sophisticated sensors. Hence, one needs to maximally utilize the standard onboard sensor suite, and possibly mitigate quality issues by more sophisticated software. A challenge in the subsea environment is localization and navigation, due to the lack of absolute position reference (posref) signals, like GPS. A way to circumvent this is to do motion state estimation of the vehicle relative to the local environment by the method of Simultaneous Localization And Mapping (SLAM) based on the onboard monocular camera. SLAM consists of the concurrent mapping of the environment and the estimation of the pose of the robot (vehicle; ROV) moving within it.

The low-cost ROV uDrone was developed in autumn 2015 by a student team at the Department of Marine Technology at NTNU. The uDrone has the following sensor suite: depth (pressure sensor), heading (IMU/gyro), cameras, and it is also possible to measure pose by the Qualisys underwater positioning system. The objective of the vehicle is to be a testbed for control, estimation, and detection algorithms in the NTNU MC-lab. This project aims to learn, understand, implement, and test relevant SLAM algorithms for monocular underwater cameras for underwater exploration, possibly aided by the onboard lasers.

Scope of Work
1) Perform a background and literature review to provide information and relevant references on:
   a) The ROV uDrone, MC-Lab, and BluEye Robotics drones, and related camera technologies.
   b) The ORB-SLAM and related SLAM methods for monocular cameras.
   c) Relevant computer vision and image processing.
   d) The Robot Operating System (ROS) and other relevant hardware/software for handling SLAM algorithms.
   e) Relevant sensor fusion.

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

2) Design and implementation of a scale-aware SLAM algorithm:
   a) Design and implement a range estimator using parallel lasers.
   b) Install ORB-SLAM and integrate this with scale awareness.
   c) Design depth observer and use it in conjunction with the SLAM-algortihm to estimate scale.

3) Design and implementation of a camera-assisted DP function on the ROV:
   a) Change control point to camera lens position.
   b) Design and implement a "freeze position" function for stationkeeping.

4) Verification and testing:
   a) Verify algorithms with computer simulations, where applicable.
   b) Design tests for the closed-loop system accuracy and robustness, using uDrone in MC-Lab.
   c) Compare estimated position from SLAM against true position from Qualisys.

Specifications
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. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts. It 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 with objective, background, and scope and delimitations, main body with problem formulations, 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, with the printed 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, 2017	**Due date:**	As specified by the administration.

Supervisor: Roger Skjetne
Co-advisor(s): Petter Norgren, Andreas V. Henriksen (BluEye).

Trondheim,

Roger Skjetne

Digitally signed by Roger Skjetne
Date: 2017.06.08 08:46:09 +02'00'

Roger Skjetne, Supervisor

Preface

This report is the result of the Master's Thesis in Marine Technology at the Norwegian University of Science and Technology carried out during the spring of 2017. The motivation for the thesis was to make a self contained position reference system for underwater vehicles using computer vision. The methods developed can be used and improved upon to gain stationkeeping and dynamic positioning capabilities of ROVs. The idea for using ORB-SLAM came up after a computer vision seminar at NTNU Trondheim in the autumn of 2016, while the idea for laser range measurements came from (Henriksen, 2016). The assumed background of the reader should be an education in control engineering with some knowledge within hydrodynamics and computer vision.

Trondheim, June 11th, 2017

Tore Mo-Bjørkelund

Acknowledgment

I want to thank my supervisor Professor Roger Skjetne and co-supervisors Andreas V. Henriksen at BluEye and PhD. candidate Petter Norgren for the support, discussions and inspiration. Further, I would like to thank Torgeir Wahl for all help in relation to practical issues in MCLab, especially for procuring batteries for the uDrone and granting me access to proper soldering equipment.

I am also thankful to BluEye Robotics which has been supporting during the project and given me much inspiration, also for lending me an IMU. A big thank you to my friends and co-students Mads S. Skinderhaug, for all he has taught me about Linux and soldering, Børge Mokleiv, for active problem solving in the MCLab, Erlend S. Harbitz, for motivation and inspiration, and all three for interesting and insightful discussions. I further would like to thank my girlfriend Karina Solheim for all the support along the way. Last, I would like to thank my parents, Anna Mo-Bjørkelund and Vidar Østerbø, for letting me find my own way in this world, and my siblings, Rune Mo-Bjørkelund and Nora Mo-Bjørkelund for inspiration and motivation.

Abstract

Low-cost ROVs are currently entering the market *en masse*, and providers face a great challenge in keeping the cost low while providing the best user experience. For this purpose, simple sensors and advanced software is needed. We explore the possibility of monocular camera-based *Simultaneous Localization And Mapping*(SLAM) on an ROV to achieve dynamic positioning. The main challenge of monocular SLAM is lack of scale, due to a camera being a bearing-only sensor. Thus, it tells us nothing about the scale of our map or the scale of the ROV position within the map. We implement the ORB-SLAM algorithm from (Mur-Artal et al., 2015), along with laser range measurements to estimate scale. The position estimate from SLAM is used for feedback control of uDrone, a low-cost ROV built by students at the Marine Technology department of the Norwegian University of Science and Technology. We present and discuss challenges, methods and results form the work, concluding on the feasibility of underwater monocular SLAM for dynamic positioning.

Index Terms – Simultaneous Localization And Mapping(SLAM), ORB-SLAM, ROV, underwater, computer vision, dynamic positioning, ORB-SLAM, laser range.

Summary

A range of low-cost *Remotely Operated Vehicles*(ROVs) are currently emerging in the consumer market. In this thesis we are exploring the possibilities of using monocular camera-based *Simultaneous Localization And Mapping*(SLAM) on a low cost ROV for real-time position feedback, stationkeeping and dynamic positioning. Monocular SLAM has an inherent weakness, stemming from that a normal camera is a bearing-only sensor, this means that the scale of the map and the camera position within the map is unknown. We present two methods for estimating the relation between SLAM and real world scales, one based on laser range measurements using computer vision and the other based on comparing offsets in the vertical direction. We present the low-cost ROV used in this thesis, the uDrone, and the *Marine Cybernetics Laboratory*(MCLab). ORB-SLAM (Mur-Artal et al., 2015) is chosen as the SLAM-algorithm for the thesis. ORB-SLAM is a graph-based SLAM-algorithm based on the ORB feature descriptor (Rublee et al., 2011) and *Bundle Adjustment*(BA) optimization. We then present the pinhole camera model, and the distortions made of the lens shape, camera imperfections and refraction. ORB-SLAM only outputs the pose of the camera, thus, we need to estimate velocity, this is done by first presenting the system model from (Sandøy, 2016), then developing two *Extended Kalman Filters*)(EKFs). The first solely based on the model and inputs from the scaled ORB-SLAM pose, the second we exchange the kinematic model by *Inertial Measurement Unit*(IMU)-measurements in all translatory degrees of freedoms. To achieve dynamic positioning, we implement *Proportional Integral Derivative*(PID) control laws in sway, heave and yaw degrees of freedom, and a *Pseudo Derivative Feedback*(PDF) (Phelan, 1971) based control law inspired by (Kjerstad et al., 2017) in surge. ORB-SLAM, the scale estimators, EKFs and control laws are implemented in the *Robot Operating System* on uDrone and tested in the MCLab basin. We present the results of the experiments and comment on their performance. Both the EKFs and the scaling factor estimates showed somewhat unsatisfactory results, while the laser range measurements, controllers and ORB-SLAM showed satisfactory performance. We conclude by arguing that the work done in this thesis is a successful proof of concept and that if implemented on a commercial product, it needs more robustness and refinement.

Samandrag

Dei siste åra har interessa og tilgjengelegheita av lågkost undervassfarkostar auka kraftig. I denne avhandlinga utforskar vi moglegheitane for å nytta monokulært kamerabasert *simultan lokalisering og kartlegging*(SLAM) på ein lågkost undervassfarkost for sanntid posisjonstilbakekopling og dynamisk posisjonering. Monokulær SLAM har ein veikskap i det at eit kamera kan berre måle retning, ikkje avstand. Dette fører til at skalaen til kartet er ukjend. Vi presenterar to metodar for å estimere denne skalaen, ved å samanlikne lasermålingar med avstandsmålingar i kartet, og ved å samanlikne verikale forskyvingar frå startpunktet. Vidare presenterar vi farkosten nytta i arbeidet presentert her, uDrone, og *Marin Kybernetikk Laboratoriet*(MCLab). Vi vel den grafbaserte ORB-SLAM-algoritmen (Mur-Artal et al., 2015) som vår SLAM-algoritme, den nyttar ORB-skildraren (Rublee et al., 2011) for datasyn og *bundtjustering*(BA) for optimering av grafen. Vi presenterar knappnålsholkameramodellen og forvrengingar forårsaka av utforminga av linsa, ufullkommenskap i kameraet og refraksjon. ORB-SLAM gjev berre posisjonen og retninga til kameraet, difor vil me estimere hastigheita. For å gjere dette, nyttar vi modellen presentert i (Sandøy, 2016) og presenterar to *Utvida Kalman Filter*(EKF) basert på denne modellen. For å oppnå dynamisk posisjonering nyttar vi oss av to ulike regulaotrar, i fridomsgradene svai, hiv og gir er *Proporsjonal Integral Derivat*(PID)-regulatorar brukt. I jag er ein *Pseudo Derivat Tilbakekopling*(PDF)-basert (Phelan, 1971) regulator brukt, denne er tungt inspirert av (Kjerstad et al., 2017). ORB-SLAM, skalaestimatorar, begge EKF, og regulatorane er implementert i *Robotoperativssystemet*(ROS) på uDrone og testa i MCLab. Begge EKF og skalaestimatorane viste lite tilfredsstillande resultat, medan laseravstansmålingane, ORB-SLAM og regulatorane viste tilfredsstillande resultat. Vi konkluderar med at arbeidet gjort i forbindelse med denne avhandlinga fungerar som eit prov for konseptet, men treng vidare utvikling og og arbeid for å gjere det robust og meir nøyaktig.

Table of Contents

List of Tables

List of Figures

Abbreviations

SLAM	=	Simultaneous Localization And Mapping
BA	=	Bundle Adjustment
KF	=	Kalman Filter
EKF	=	Extended Kalman Filter
PF	=	Particle Filter
PDF	=	Probability Density Function or Pseudo Derivative Feedback
BRIEF	=	Binary Robust Independent Elementary Features
FAST	=	Features from Accelerated Segment Test
ORB	=	Oriented FAST and Rotated BRIEF
SPLAM	=	Simultaneous Planning Localization And Mapping
RB	=	Rao-Blackwell
IMU	=	Inertial measurement unit
GPU	=	Graphics Processing Unit
CPU	=	Central Processing Unit
GHz	=	Giga Hertz
MB	=	Mega Byte
RAM	=	Random Access Memory
ROV	=	Remotely Operated underwater Vehicle
ESC	=	Electronic Speed Controller
ROS	=	Robot Operating System
MCLab	=	Marine Cybernetics Laboratory
HSV	=	Hue Saturation Value
RGB	=	Red Green Blue

Chapter 1

Introduction

"Home is behind, the world ahead,
and there are many paths to tread
through shadows to the edge of night,
until the stars are all alight."

The Lord of the Rings
J.R.R. Tolkien (1892-1973)

1.1 Background and Motivation

It is said that we know less about the ocean floor than about the lunar surface. Human urge for exploration never ceases, we will go where our technology permits us, from Mt. Everest to the Mariana Trench, humans have been there. At the risk of their lives. We are the explorers of this world, as Carl Sagan (1934-1996) said:

> *The cosmos is within us. We are made of star-stuff. We are a way for the universe to know itself.*

Our probes have journeyed the solar system and beyond, exploring space, the final frontier. Diverting our eyes from the sky, we turn to explore another world we know little of, the ocean space.
In recent years, the interest in subsea exploration has increased. At the same time the cost of personal, underwater drones has decreased, and small, low-cost ROVs are currently entering the market *en masse*. As opposed to commercial high-cost ROVs, the low-cost ROVs often have a limited sensor suite. As maneuvering a small underwater

craft can be a challenge, having a stationkeeping capability would help the user. That is, to be able to freeze the ROV position without any user input. This functionality would especially be useful when exploring some interesting feature of the underwater world, as the user can focus fully on the experience. Furthermore, when exploring a larger feature, eg. a wreck or coral reef, it would be useful to be able to move at slow speed around the feature without directly controlling the craft, and rather by setpoint regulation or path following.

1.2 Objective

Th e objective of this thesis is to explore the possibilities of monocular camera-based SLAM on a low cost ROV. Thus, it is our objective to make a self contained system for dynamic positioning of a low-cost ROV using the available on board sensors.

1.3 Problem Statement

In this thesis we seek to use ORB-SLAM as a position feedback for local dynamic positioning of a low cost *Remotely Operated Vehicle*(ROV). Consider the position, $\boldsymbol{\eta}$, and velocity, $\boldsymbol{\nu}$, of an ROV. Can a system be designed, using the camera as a sensor and the on board thrusters of an ROV as actuators, such that

$$\lim_{t \to \infty} \begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{\nu} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\eta_d} \\ \boldsymbol{\nu_d} \end{bmatrix} \tag{1.1}$$

for all $\boldsymbol{\eta}$, $\boldsymbol{\eta_d}$, $\boldsymbol{\nu}$, and $\boldsymbol{\nu_d} \in \mathbb{R}$ where ORB-SLAM is able to produce a position feedback. To clarify, ORB-SLAM can only give position feedback if there are enough visual features for it to detect. In addition, the position as provided by ORB-SLAM has an origin in its initialization point, making the position reference local, not global unless we have some knowledge about the global position of the ROV at initialization.

1.4 Scope

The main scope of this thesis is to design, implement and test dynamic positioning on a low-cost ROV based on position feedback from a monocular SLAM algorithm, ORB-SLAM from (Mur-Artal et al., 2015). The main challenge with position feedback from monocular SLAM is that scale is unknown, due to a camera being a bearing-only sensor. The work is divided into 7 main tasks, they are as follows:

- Perform a background and literature study.
- Implement ORB-SLAM on the topside processing unit of uDrone.
- Design and implement a scale estimator using parallel lasers.
- Design and implement a scale estimator using depth comparison.
- Design and implement a state estimator for velocity estimation.
- Design and implement a control law in order to achieve dynamic positioning.
- Test the implemented algorithms.

A background literature study was done in the fall of 2016, (Mo-Bjørkelund, 2016), some of the findings are repeated in this thesis for substantiation. The literature study concluded that for camera-based SLAM on a low-cost ROV, ORB-SLAM would be both suitable and feasible.

1.5 Delimitations

This project is constrained by a 20 week time frame, thus, there are some compromises made. In order to provide smooth setpoint regulation without implementing a reference filter, the PDF controller is chosen as the preferred control law. However, it is only implemented in one degree of freedom, surge, and we limit the regulation to only input steps in the desired position without any reference filtering. Furthermore, the system is designed to work in a laboratory setting, and is not robust enough for direct implementation on an ROV. All filters and controllers are also not tuned perfectly. Last, all motion models and controls are limited to the 4 degrees of freedom, under the assumption that the roll and pitch angles inherently stable due to the flotation center of uDrone being above its center of mass.

1.6 Low-cost ROVs

Recent years has seen a vast expansion in the availability of low-cost personal drones. In this section we will present the Blueye drones and the drone used in the work described in this thesis, the uDrone.

1.6.1 Blueye Robotics Drones

Blueye Robotics,(Blueye, 2017) is a company with its origins in the marine technology community in Trondheim. Their vision is to make an affordable, personal underwater drone. In doing so, they have made several prototypes and are currently working on developing the Pioneer intended for mass production. The current prototype for the Pioneer is fitted with one camera module and has allocated space for two lasers for distance estimation. The drone communicates to a device(PC or smart phone) via a fiber optic cable connected to a wifi transponder that floats on the surface. The video feed is available topside, and is intended for steering the drone, the current model has some lag from the drone and up, in the range of 150-500 ms. The Pioneer is estimated to be available for purchase in 2018.

1.6.2 BlueROV and uDrone

BlueROV is a product from Blue Robotics, now no longer available on the consumer market due to the release of the new version, BlueROV2. This drone is a built-it-yourself kit, where a chassis, thrusters and a watertight enclosure are included. Electronics and software has to be provided by the consumer. In the autumn of 2015 and spring of 2016, a team of students at NTNU outfitted a BlueROV kit with electronics, sensors and software. They chose to call it uDrone and their work laid some of the foundation for this masters thesis. In the spring of 2017, some upgrades and repairs were made to the uDrone by the author and Mads S. Skinderhaug. This includes changing the on board CPU and adding an additional *Inertial Measurement Unit*(IMU). As the test platform for this master thesis is the uDrone, we will give an account of its electronics, sensor suite, and software.

Hydrodynamic Properties

A hydrodynamic model of uDrone was identified in (Sandøy, 2016), we present it in this thesis in section 3.1. In addition, it was shown by experiment that uDrone floats right side up, even when submerged. This means that the center of buoyancy lies above the center og gravity, making roll and pitch angles inherently stable around zero.

Electronics

On board the uDrone, there is power electronics, responsible for reducing the voltage from the battery($14.8V$ nominal) to the voltage used by the Raspberry Pi and Arduino($5V$). Furthermore there are 6 *Electronic Speed Controllers*(ESC), one for each thruster. They are controlled by an Arduino Mega, which is responsible for the thrust allocation. The

Arduino also reads tempreature, data from the external IMU, and controls four laser lights mounted in the front of the drone. The uDrone is equipped with the following sensors and electronics:

- External IMU with pressure sensor.
- Internal IMU, located close to center of mass.
- Internal temperature sensor.
- Wide angle usb-camera with close to 180 degrees field of view.
- Raspberry Pi Camera with 62 degrees field of view.
- Battery voltage sensor.
- Arduino Mega.
- Raspberry Pi 3b.
- 6 Electronic Speed Controllers.
- Voltage converter.
- 4 Parallel laser lights mounted in a square.

In this thesis, we will focus on using the cameras, two of the parallel laser lights, and both of the IMUs as sensors.

1.6.3 Software

An extensive amount of software was developed by the above mentioned team of students in the course of their master and project thesis work in 2015-2016, (Sandøy, 2016). This software includes, but is not limited to:

- Thrust allocation algorithm.
- Direct motion control.
- Auto heading and depth.
- Laser range control algorithm.

The work done by the previous students laid the groundwork for the work done in this thesis. The thrust allocation algorithm is used as is, while the control algorithms are modified to fit the purposes of this thesis. Implementation was made easier due to the software architecture being already in place.

1.6.4 Topside Processing Module

Many of the computing tasks involving image processing and control are done on a PC connected to uDrone via Ethernet cable. Formally this is called the topside processing module, or topside. It is a Dell PC, with 32 GB memory and an Intel i7 processor, capable of running ORB-SLAM and all other tasks needed in the work presented in this report in real time. As a rule, if processing can be done by the topside, it is done by the topside.

1.7 System Architecture

For the purpose of achieving dynamic positioning using ORB-SLAM as position reference we propose the system architecture presented in figure 1.1.

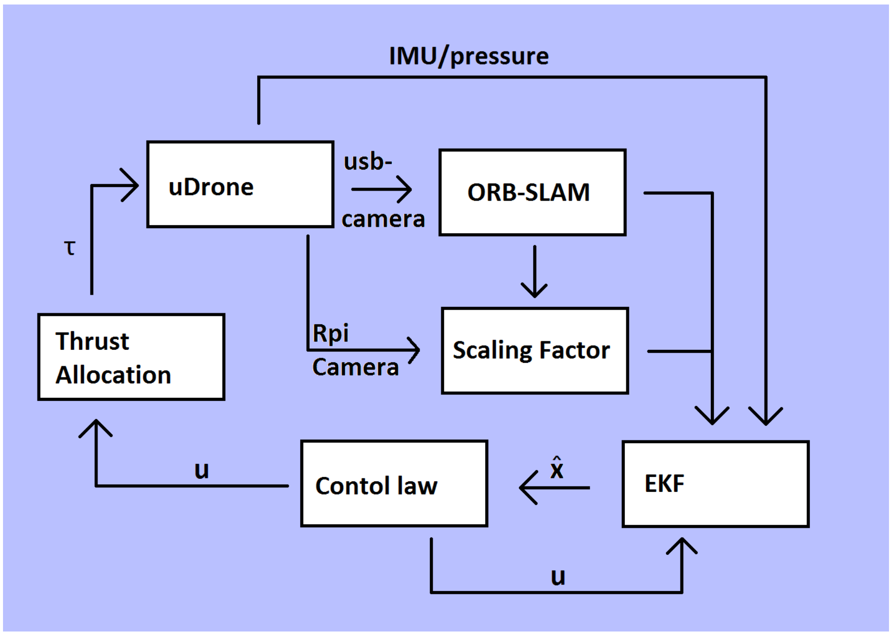


Figure 1.1: Proposed system architecture for implementing camera-based SLAM for dynamic positioning on ROV uDrone. The blocks are: uDrone; the physical process plant, ORB-SLAM; the monocular SLAM algorithm, Scaling Factor provides scale to ORB-SLAM, the EKF estimates position, velocity and bias, $\hat{x}$, the control law calculates a thrust command, and the Thrust Allocation converts the control, u, to force exerted by the actuators acting on the uDrone.

The system architecture in figure 1.1 will be changed depending on if we use the scaling factor found by laser range measurement or by depth comparison. However, this is the architecture used for the majority of the work presented in this thesis. It is also worth noting that the actual implemented software architecture is more complicated than presented in 1.1. All implementation done in this thesis are done in the *Robot Operating System*(ROS), writing code in C++. We continue to give a brief presentation of ROS.

1.8 Robot Operating System

The *Robot Operating System*(ROS) is an open source framework for getting robots to do things (Quigley et al., 2015). In its official distribution, there are over 2000 software packages, and a large community of users, making it easy to find tutorials and other help. We will here present the fundamentals of ROS.

1.8.1 Constituting Parts

ROS consists of many parts, building a solid framework for writing code for robots. First, a set of drivers enabling the user to read data from sensors and send commands to actuators in an abstracted, well-defined format. Second, a large and growing collection of fundamental algorithms that allows for the building of maps of the world and navigating within it. Third, a computational infrastructure that allows for movement of data in an organized manner. The publisher/subscriber structure of ROS makes it easy to see which components of the code talk to each other, and also the hierarchy of the code is easily identifiable. Fourth, a large set of tools for visualization, debugging, and recording of data. Finally, the ROS community, or ecosystem, provides an extensive set of resources, such as a Wiki page, forums, and Q- and A-page.

1.8.2 ROS Systems

ROS systems operate peer-to-peer, in that they consist of numerous small computer programs that connect to one another continuously, exchanging messages. There is no central routing of the messages, this makes for a more complicated network of connected programs, but scales better as the amount of data increases. Another strong side of ROS is that is multilingual, the different programs can be written in different languages. The languages available for ROS include, but is not limited to, C++, Python, Java, JavaScript, and MATLAB.

1.9 Structure of Report

We continue in chapter 2 to present simultaneous localization and mapping, ORB-SLAM, challenges to underwater monocular SLAM, and camera distortion. In chapter 3 we present the uDrone process model, scaling factor estimation, the EKF, and control laws. Experiments, results, and discussion of the specific results are presented in chapter 4, followed by an overall discussion in chapter 5. In chapter 6 we conclude and present a suggestion for future work.

Chapter 2

Simultaneous Localization and Mapping

2.1 Background

2.1.1 Introduction

Simultaneous Localization And Mapping(SLAM) is one of the most fundamental problems in robotics, as stated in (Thrun et al., 2005). SLAM problems arise when the robot does not have a map of the environment. In its most basic form, SLAM seeks to solve the problem formulated in (2.1).

$$p(\boldsymbol{x}_{1:t}, \boldsymbol{m}|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) \tag{2.1}$$

The expression in (2.1) seeks to estimate the current and the past pose of the robot, $\boldsymbol{x}_{1:t}$, and the map, $\boldsymbol{m}$, given all measurements, $\boldsymbol{z}_{1:t}$, and all control inputs, $\boldsymbol{u}_{1:t}$ as a probability density function. Furthermore, this is called the *full SLAM problem*, the *online SLAM problem*, presented in (2.2), seeks only to estimate the current pose of the robot.

$$p(\boldsymbol{x}_t, \boldsymbol{m}|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) \tag{2.2}$$

Over the years, many solutions to this problem has been proposed, the most notable being solutions involving the *Extended Kalman Filter*(EKF), graph-based optimization methods, and solutions using the *Particle Filter*(PF) as discussed in (Thrun and Leonard, 2008). The preferable solution may depend on several factors such as computing power, sensor suite and desired accuracy. Here, we focus on graph-based methods, and will give a brief introduction to the intuition of the method before moving on to ORB-SLAM.

2.1.2 Graph-Based SLAM

Graph-based methods solve the SLAM problem through nonlinear sparse optimization, as stated in (Thrun and Leonard, 2008). The robot poses and landmark positions are thought of as nodes in a graph. From each pose, $\boldsymbol{x}_t$, there is a link to the previous pose, $\boldsymbol{x}_{t-1}$ and the landmarks, m_i, observed at time t. As each of the measurements carry an inherent uncertainty, they can be thought of as soft constraints in an optimization function. The graph construction is illustrated in figure 2.1. Further, it is helpful to think of the graph as a mass-spring system, computing the solution is equivalent to computing the state of minimum energy of this system. The graph corresponds to the log-posterior of the full slam problem, as formulated in (Thrun and Leonard, 2008).

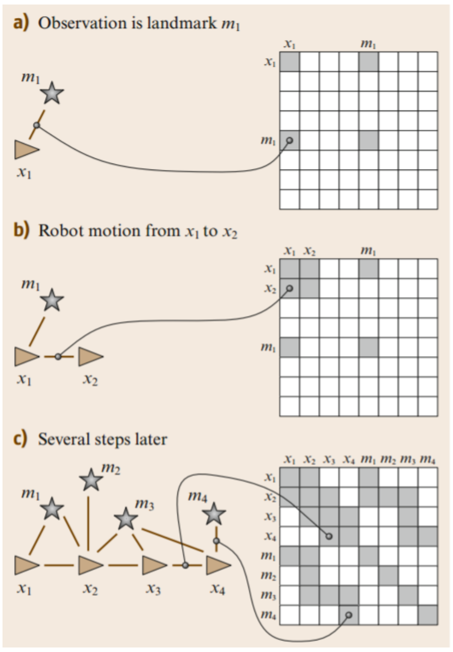

a) Observation is landmark m_1
b) Robot motion from x_1 to x_2
c) Several steps later
m_1
m_2
m_3
m_4
x_1
x_2
x_3
x_4

Figure 2.1: Illustration of graph construction, graph in the left diagram, constraints in matrix form to the right, figure source: (Thrun and Leonard, 2008).

$$\log\ p(\boldsymbol{x}_{1:t}, \boldsymbol{m} | \boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) \tag{2.3}$$

The solution of (2.3) can be formulated as in (2.4).

$$\begin{aligned} \log\ p(\boldsymbol{x}_{1:t}, \boldsymbol{m} | \boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) &= const \\ &+ \sum_t \log\ p(\boldsymbol{x}_t | \boldsymbol{x}_{t-1}, \boldsymbol{u}_t) \\ &+ \sum_t \log\ p(\boldsymbol{z}_t | \boldsymbol{x}_t, \boldsymbol{m}) \end{aligned} \tag{2.4}$$

The constraint of the form log $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)$ is the result of a single robot motion event at time t. It corresponds to one arc in the graph. As with the previous constraint, the term log $p(\boldsymbol{z}_t|\boldsymbol{x}_t, \boldsymbol{m})$ corresponds to one sensor measurement at time t, to which there is also a corresponding arc in the graph. The solution to the SLAM problem is then the solution to the mode of (2.5).

$$\boldsymbol{x}_t^*, \boldsymbol{m}^* = \text{argmax}_{\boldsymbol{x}_{1:t}, \boldsymbol{m}} \log\ p(\boldsymbol{x}_{1:t}, \boldsymbol{m}|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) \tag{2.5}$$

As stated in (Thrun and Leonard, 2008), the solution of (2.4) can, under the Gaussian noise assumption, be resolved to a sparse function on the quadratic form, here presented in (2.6).

$$\begin{aligned} \log\ p(\boldsymbol{x}_{1:t}, \boldsymbol{m}|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) &= const \\ + \sum_t \left[\boldsymbol{x}_t - g(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)\right]^\intercal \boldsymbol{R}_t^{-1} \left[\boldsymbol{x}_t - g(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)\right] & \\ + \sum_t \left[\boldsymbol{z}_t - h(\boldsymbol{x}_t, \boldsymbol{m})\right]^\intercal \boldsymbol{Q}_t^{-1} \left[\boldsymbol{z}_t - h(\boldsymbol{x}_t, \boldsymbol{m})\right] & \end{aligned} \tag{2.6}$$

In equation 2.6, $g(\cdot)$ is the model for state propagation, $\boldsymbol{R}$ is the corresponding covariance matrix for the states, $h(\cdot)$ is the measurement equation, and $\boldsymbol{Q}$ is the corresponding measurement covariance matrix. In order to solve this, a number of optimization techniques can be applied, in ORB-SLAM, *bundle adjustment* is used. We will now present the ORB-SLAM algorithm.

2.2 ORB-SLAM

ORB-SLAM was first presented in (Mur-Artal et al., 2015), and presents a versatile method for monocular SLAM using *Bundle Adjustment*(BA). The bundle adjustment method eliminates the need for filtering of the states in SLAM, such as landmark positions and robot pose. This makes ORB-SLAM more robust for prolonged use in contrast to classical filtering approaches to SLAM such as PTAM (Klein and Murray, 2007) or EKF-SLAM (Davison et al., 2007). In this section ORB-SLAM is presented briefly with focus on its strengths and weaknesses in underwater operation of ROVs. In his section we will present ORB-SLAM as it is presented in (Mur-Artal et al., 2015), the challenges of ORB-SLAM in the underwater environment, and the implemented modifications.

2.2.1 Fundamentals

In this section we will present ORB-SLAM as it is presented in (Mur-Artal et al., 2015). ORB-SLAM, using BA, is a graph-based SLAM algorithm. The algorithm is divided into three threads: tracking, local mapping and loop closing. Furthermore, ORB-SLAM is named after the *Oriented FAST and Rotated BRIEF*(ORB)-feature descriptor. Oriented FAST and Rotated BRIEF(ORB) is a feature descriptor based on the FAST corner detector and the BRIEF descriptor, from (Rublee et al., 2011). The descriptor is rotation invariant and resistant to noise. It was showed in (Rublee et al., 2011), that it is two orders of magnitude faster than SIFT. BRIEF is a descriptor with good robustness with respect to lighting, blur, and perspective distortion. However, it is not rotational invariant, this is introduced in the ORB descriptor, making it usable for real world purposes. The following is a description of ORB-SLAM, based on figure 2.2.

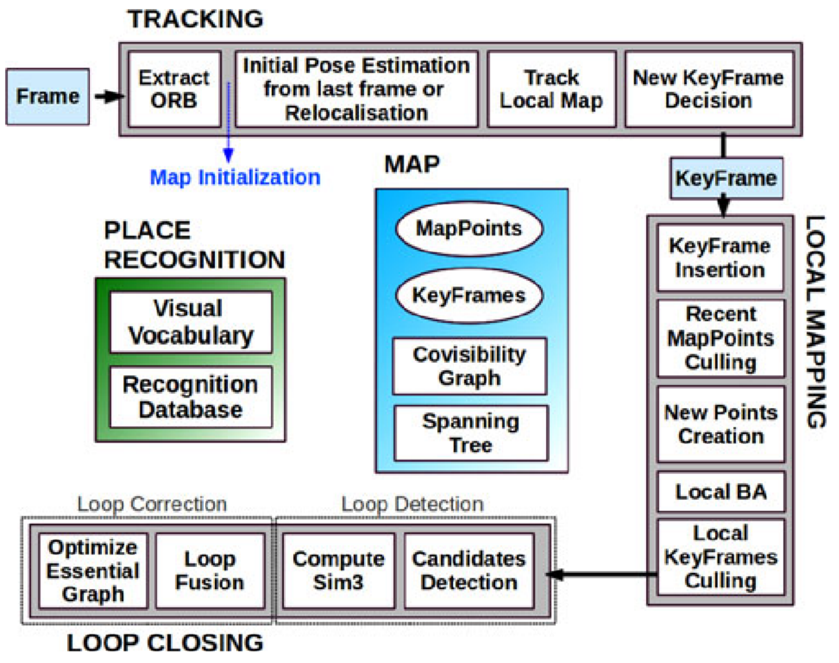


Figure 2.2: ORB-SLAM system overview, showing the tree threads and the main components of the algorithm, figure from (Mur-Artal et al., 2015).

Features

In order to operate at a frame rate of 30 FPS, ORB-SLAM needs feature extraction in less than 33ms, therefore ORB, (Rublee et al., 2011), is chosen. ORB has good invariance to viewpoint, which allows for matching with wide baselines, which in turn boosts the accuracy of BA.

Three Thread Structure

After initialization, the ORB-SLAM system incorporates three threads running in parallel: tracking, local mapping, and loop closing.

Tracking Tracking is responsible for localizing the camera in every frame and deciding when to insert a new keyframe. Tracking is done by doing initial feature matching with the previous frame and then optimize the pose by *motion-only* BA. If tracking is lost, global relocalization is initialized.

Local Mapping Local mapping processes the new keyframes and performers *local BA* to achieve an optimal reconstruction of the surroundings of the camera pose. New correspondences for unmatched ORB in the new keyframe are searched in connected keyframes in the covisibility graph in order to triangulate new points. Local mapping is also responsible for culling low quality points and redundant keyframes.

Loop Closing The loop closing searches for loops with every new keyframe. If a loop is detected, a similarity transform is computed in order to inform about the accumulated drift. Both sides of the loop is then aligned and the matched points are fused. Then, optimization over the *essential graph* is done. All optimization, including BA, is done using the Levenberg-Marquardt algorithm, (Moré, 1978).

The Map

The map consists map points, keyframes, the essential graph, the covisibility graph, and the spanning tree.

Map Points Each point, $\boldsymbol{p}_i$, in the map stores the following information:

- Its 3D position, $\boldsymbol{X}_{w,i}$,in the world frame.
- The viewing direction $\boldsymbol{n}_i$.
- A representative ORB descriptor $\boldsymbol{D}_i$.

- The maximum d_{max} and minimum d_{min} distances at which the point can be observed.

Keyframes Each keyframe, K_i, stores the following:

- The camera pose $\boldsymbol{T}_{iw}$
- The camera intrinsics, including focal point and principal point.
- All ORB features extracted in the frame whose coordinates are undistorted.

Map points and keyframes are generated generously, while the culling process happens after it is assessed which points and keyframes are bad or redundant.

Covisibility Graph and Essential Graph

Covisibility Graph The covisibility graph describes how many map points are visible from the different keyframes. It is represented as a undirected weighted graph between the keyframes. The weight is dependent on how many points are visible in the keyframes. The weight, θ, is defined at the number of common map points.

Essential Graph The essential graph is a sparser, but more robust version of the covisibility graph. For the essential graph, the minimum threshold of common map points is $\theta_{min} = 100$ points. The covisibility graph also contains the spanning tree and loop closure edges. The spanning tree is built from the initial keyframes and when a new keyfram is added, it is added to the tree linked to the keyframe which shares the most observations.

Bags of Words Place Recognition

Bags of words place recognition is embedded in the system to perform loop detection and relocalization.

2.2.2 Map Initialization

There are two ways to initialize the map, by a homography assuming a planar scene or by a fundamental matrix assuming a nonplanar scene. ORB-SLAM computes both in parallel and give each a score in order to decide which one to use for initialization. The initialization process is broken into five steps, which are presented below.

Finding Initial Correspondences : Extract ORB features from the current frame F_c, and search for matches in the reference frame F_r, if the number of matches is to low, reset the reference frame.

Parallel Computing of Two models The homography, H_{cr}, and fundamental matrix, F_{cr}, are defined as

$$\begin{aligned} \boldsymbol{x}_c &= \boldsymbol{H}_{cr}\boldsymbol{x}_r & (2.7) \\ \boldsymbol{x}_c^T \boldsymbol{F}_{cr}\boldsymbol{x}_r &= 0 & (2.8) \end{aligned}$$

where they are calculated by the normalized direct linear transformation and the eight-point algorithm respectively, (Hartley and Zisserman, 2003), inside a RANSAC scheme. Scores for the individual models $M(H$ for homography and F for fundamental matrix) are calculated as

$$\begin{aligned} S_M &= \sum_i (\rho_M(d_{cr}^2(\boldsymbol{x}_c^i, \boldsymbol{x}_r^i, M)) + \rho_M(d_{rc}^2(\boldsymbol{x}_c^i, \boldsymbol{x}_r^i, M))) & (2.9) \\ \rho_M(d^2) &= \begin{cases} \Gamma - d^2 & if \quad d^2 < T_M \\ 0 & if \quad d^2 \geq T_M \end{cases} & (2.10) \end{aligned}$$

where d_{cr}^2 and d_{rc}^2 are the symmetric transfer errors from one frame to the other. T_M is the outlier rejection threshold based on the χ^2 test at 95%, assuming standard deviation of 1 pixel in the measurement error. Γ is defined equal to T_H.

Model Selection In order to choose which model to use, the scores are combined as

$$R_H = \frac{S_H}{S_H + S_F} \tag{2.11}$$

and the homography is selected if $R_H > 0.45$.

Structure From Motion When a model is selected, the motion hypothesis associated with it is retrieved. For homography, eight motion hypotheses are retrieved, and attempted triangulated. If this fails, there might be insufficient parallax leading to the choice of a wrong solution. For the fundamental matrix, it is converted to an essential matrix, $\boldsymbol{E}_{rc}$, using the calibration matrix $\boldsymbol{K}$ as

$$\boldsymbol{E}_{rc} = \boldsymbol{K}^T \boldsymbol{F}_{rc} \boldsymbol{K} \tag{2.12}$$

Bundle Adjustment When a good initial guess is generated from the motion hypothesis, BA is performed to refine the initial reconstruction.

2.2.3 Tracking

In this section, we describe the tracking thread in ORB-SLAM, in tracking, all steps are performed for each frame.

ORB Extraction FAST corners at eight scale-levels with a scale factor 1.2 are extracted. Form images of size 512×384 to 751×480, 1000 corners are extracted, for larger images, 2000. Homogeneous distribution is achieved by dividing each scale level in a grid, attempting to extract at least five corners per cell. Orientation and ORB descriptor are then computed on the FAST corners.

Initial Pose Estimation If tracking was successful for the last frame, a constant velocity model to predict the camera pose and perform a guided search of the map points observed in the last frame. If not enough matches were found, the search is widened. The pose is then optimized with the found correspondences.

Pose Estimation via Global Relocalization If the tracking is lost, the frame is converted into bag of words and and query the recognition database from keyframe candidates for global relocalization. Correspondences with ORB associated with map points in each keyframe is computed. If a camera pose with enough inliers is found, the pose is potimized and tracking continues.

Tracking the Local Map Once the camera pose is estimated, and there is an initial set of feature matches, the map is projected into the frame, and more point correspondences are searched for. In order to bound the complexity in large maps, only a local map is projected. The local map is the set $\mathcal{K}_1$ of keyframes which share map points witht he current frame, and the set $\mathcal{K}_2$ with neighbours to the keyframes $\mathcal{K}_1$ in the covisibility graph. Each map point in $\mathcal{K}_1$ and $\mathcal{K}_2$ is searched for in the current frame by algorithm 1 and is fully optimized when all map points are found in the frame.

Algorithm 1 Local map point search in tracking.

```
1: Compute the point projection $\boldsymbol{x}$ in the current frame, discard if out of bounds.
2: Compute the angle between the current viewing ray $\boldsymbol{v}$ and the map point viewing direction $\boldsymbol{n}$, discard if $\boldsymbol{v} \cdot \boldsymbol{n} < cos(60°)$.
3: Compute the distance $d$ from the map point to the camera center, discard if $d \notin [d_{min}, d_{max}]$.
4: Compute the scale in the frame by $d/d_{min}$.
5: Compare the representative descriptor $\boldsymbol{D}$ of the map point with all unmatched ORB features in the frame, at the predicted scale, and near $\boldsymbol{x}$, associate the point with the best match.
```

New Keyframe Decision In order to insert a new keyframe the following condition must be met:

- More than 20 frames must have passed from the last global relocalization.
- Local mapping is idle, or more than 20 frames have passed since the last keyframe insertion.
- Current frame tracks at least 50 points.
- Current frame tracks less than 90% of points in K_{ref}.

Where $K_{ref} \in \mathcal{K}_1$ is the keyframe which shares the most map points with the current frame. If a keyframe is inserted while the local mapping is busy, local BA is stopped in order to process the new keyframe.

2.2.4 Local Mapping

In this section, we present the seps performed when inserting a new keyframe K_i.

Inserting Keyframe The covisibility graph is first updated, adding a new node for K_i, then the spannig tree is updated. Last, the bags of words representation of the keyframe is computed.

Culling Recent Map Points In order to be retained in the map, a map point needs to fulfill two conditions:

- Tracking must find the point in more than 25% of the frames in which predicted to be visible.

- If more than one keyframe have passed from map point creation, it must be observed from at least three keyframes.

Once a map point has passed this test, it can only be removed if at any time it is observed from less than three keyframes.

Map point Creation New map points are generated by triangulation from connected keyframes $\mathcal{K}_c$ in the covisibility graph. For each unmatched ORB in the keyframe K_i, the point is searched for in the other keyframes. To be accepted the new points are checked for positive depth in both cameras, parallax, reprojection error, and scale consistency.

Local BA Local BA optimizes the currently processed keyframe along with all keyframes connected to it in the covisiblity graph, $\mathcal{K}_c$, and all map points seen by those keyframes. The local BA also include the keypoints not connected in the covisibility graph, but they remain fixed.

Local Keyframe Culling Keyframes which contain redundant information are deleted. Keyframes in $\mathcal{K}_c$ that share at least 90% of its map points with three other keyframes in $\mathcal{K}_c$ are deleted.

2.2.5 Loop Closing

Loop closure is an important part of SLAM, and is used in order to correct drift and not generating overlaying maps. In this section we present the steps of loop closing in ORB-SLAM.

Loop Candidate Detection First, the similarity between the bag of word vector of K_i and all its neighbours in the covisibility graph is computed. Then a query of the recognition database is made. In order to be accepted, three consistent loop candidates must be made consecutively.

Similarity Transform In computing the similarity transform, *Sim(3)*, the drift along all 7 DOFs in the loop is found. The 7 DOFs are three translations, three rotations and scale.

Loop Fusion In loop fusing, covisibility graph is updated, such that the current keyframe, K_i, is corrected by the similarity transform. The correction is then propagated to all neighbours of K_i, concatenating transformations, so that both sides of the loop get aligned. After correcting all the keyframes in the loop, overlapping map points are fused.

Essential Graph Optimization In order to effectivly close the loop, pose graph potimisation over the essential graph is done. The optimization distributes the closing error along the graph, and map points are transformed according to the transformation according to the transformation of one of the keyframes that observes it.

2.2.6 Bundle Adjustment

The core of ORB-SLAM is based on *Bundle Adjustment*(BA). In this section we will present the general form of BA and the specific form used in ORB-SLAM

Using the description from (Mur-Artal et al., 2015), we present the bundle adjustment algorithm as follows. Consider N map keypoints, positioned in $\boldsymbol{X}_{w,j} \in \mathbb{R}^3$ appearing in K images with poses $\boldsymbol{T}_{w,i} \in SE(3)$, where the index w indicates the world reference frame. The positions are optimized minimizing the reprojection error with respect to the matched keypoints $\boldsymbol{x}_{i,j}$ in the image frame. The error for keypoint j in image i is

$$\boldsymbol{e}_{i,j} = \boldsymbol{x}_{i,j} - \pi_i(\boldsymbol{T}_{w,i}, \boldsymbol{X}_{w,j}) \tag{2.13}$$

where π_i is the projection function

$$\pi_i(\boldsymbol{T}_{w,i}, \boldsymbol{X}_{w,j}) = \begin{bmatrix} f_{i,u}\frac{x_{i,j}}{z_{i,j}} + c_{i,u} \\ f_{i,v}\frac{y_{i,j}}{z_{i,j}} + c_{i,v} \end{bmatrix} \tag{2.14}$$

$$[x_{i,j}, y_{i,j}, z_{i,j}]^T = \boldsymbol{R}_{w,i}\boldsymbol{X}_{w,j} + \boldsymbol{t}_{w,i} \tag{2.15}$$

where $\boldsymbol{R}_{w,i} \in SO(3)$ and $\boldsymbol{t}_{w,i} \in \mathbb{R}^3$ are rotation and translation parts of $\boldsymbol{T}_{w,i}$ respectively, and $(f_{i,u}, f_{i,v})$ and $(c_{i,u}, c_{i,v})$ are the focal length and principle point associated with image i. The cost function used is formulated as

$$\boldsymbol{C} = \sum_i^K \sum_j^N \boldsymbol{\rho}_h(\boldsymbol{e}_{i,j}^T \boldsymbol{\Omega}_{i,j}^{-1} \boldsymbol{e}_{i,j}) \tag{2.16}$$

Where $\boldsymbol{\rho}_h$ is the Huber robust cost function, and $\boldsymbol{\Omega}_{i,j} = \sigma_{i,j}^2 I_{2\times 2}$ is the covariance matrix associated with the full scale at witch the keypoint was detected. With a tunable parameter, δ, the Huber robust cost function is defined as

$$\boldsymbol{\rho}_h(a) = \begin{cases} \frac{1}{2}a^2 & \text{for } |a| < \delta, \\ \delta(|a| - \frac{1}{2}\delta) & \text{otherwise.} \end{cases} \tag{2.17}$$

This puts quadratic penalty on small errors and linear penalties on outliers($|a| > \delta$).

Thus, we have a method for optimizing both the pose of the camera and the position of the keypoints. For this to be valid, we make some assumptions in regards to the world. First, we assume the world to be static, and that all our keypoints are fixed in the world frame. Second, that we have a large enough sample of different images to optimize over. These images must contain the same or parts of the same environment and be taken from different poses. In other words, the image sample must be persistently exited in the position of the keypoints, in computer vision this effect is called sufficient parallax. BA can also be used to solve for the camera parameters needed for camera calibration.

2.2.7 ORB-SLAM in the Underwater Environment

ORB-SLAM is originally intended for above water operation in a feature rich environment, on a moving camera. As stated in (ORB-SLAM, 2017):

> *In general our Monocular SLAM solution is expected to have a bad time in the following situations:*
>
> - *No translation at system initialization (or too much rotation).*
> - *Pure rotations in exploration.*
> - *Low texture environments.*
> - *Many (or big) moving objects, especially if they move slowly.*

Due to light scattering in water, light has a much smaller penetration length in water than in air. This means that we need to be relatively close to objects before we are able to see them compared to above water. In turn, this reduces the amount of features available in the environment for ORB-SLAM. Another limiting factor in using ORB-SLAM underwater is the ORB-vocabulary, a feature vocabulary trained on a set of images. According to (Mur-Artal and Tardós, 2014), the image set used in ORB-SLAM is the Bovisa 2008-09-01 dataset, (Bonarini et al., 2006). This image set has the types mixed, static, and natural lighting. This in turn means that for ORB-SLAM to function better underwater it would probably need a vocabulary trained on underwater images.

2.2.8 Changes Made to ORB-SLAM

The original ORB-SLAM algorithm, available on GitHub (ORB-SLAM, 2017), was changed in some respect to fit our application. In this section we will present the changes made and why.

Initialization

In the initialization, the number of key points needed for initialization was reduced from 100 to 70. This was done to ease the initialization process due to the lack of features in the *Marine Cybernetics Laboratory*(MCLab) basin compared to an outdoor scene. The reduction also increases the risk of wrongful initialization and decreases its robustness. However it was found that the change had significant positive impact on the initialization success rate.

System State

ORB-SLAM can have different operation states, three of them are:

- Initializing.
- Tracking.
- Track lost, trying to re-localize.

For the rest of our system, the state estimator and control law, it is useful to know the state of ORB-SLAM. In order to make use of the states, they are published in ROS, and made available to the rest of the system.

Range Measurement

As our system is dependent on a scale estimate, we want to measure the distance from the camera to the points in near vicinity of where the laser range measurement is taken. The procedure for doing this, described in section 3.2.4, was also implemented in the ORB-SLAM algorithm.

2.3 Camera Calibration

Proper reprojection is dependent on a properly calibrated camera model. Camera calibration undistorts images taken by said camera such that they can be represented by the pinhole camera model, found in (Wöhler, 2012), and presented in figure 2.3. OpenCV and ROS offers tools for camera calibration, using a chess board pattern where the size

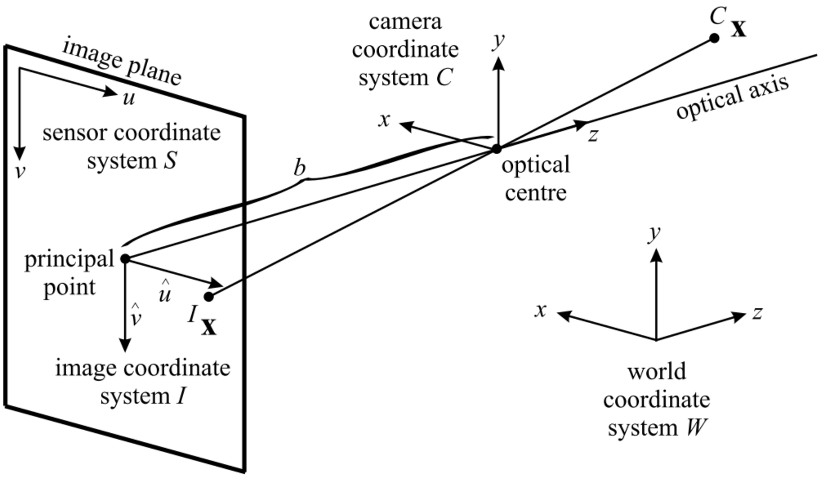


Figure 2.3: Pinhole camera model, figure from Wöhler (2012).

of the tiles are known. In this section we will present lens and refractive distortion, along with camera calibration. In this section, we will use the notation presented in figure 2.3.

Lens Distortion

Due to lens distortion, there is often a more complex mapping between the sensed image and the image plane. The distorted coordinates, $^I\boldsymbol{x}_d$, are obtained by the undistorted coordinates, $^I\boldsymbol{x} = (\hat{u}, \hat{v})$. Here the lens distortion model from (Heikkilä and Silvén, 1997) is described in (2.18).

$$^I\boldsymbol{x}_d = (1 + k_1 r^2 + k_3 r^4 + k_5 r^6)\,^I\boldsymbol{x} + \boldsymbol{d_t} \tag{2.18}$$

$$\boldsymbol{d}_t = \begin{bmatrix} 2k_2\hat{u}\hat{v} + k_4(r^2 + 2\hat{u}^2) \\ k_2(r^2 + 2\hat{v}^2) + 2k_4\hat{u}\hat{v} \end{bmatrix} \tag{2.19}$$

$$r^2 = \hat{u}^2 + \hat{v}^2 \tag{2.20}$$

Finding the values for k_1, k_2, k_3, k_4, and k_5 is discussed next. As the image perceived by the camera is distorted by the lens, and improper image sensor alignment, we wish to ”undistort” the image by use of (2.18). Notice that the transformation is done by taking the original image and distorting it to an ”undistorted” state. The constants k_1, k_3, and k_5 describes radial distortion, while k_2 and k_4 describes tangential distortion. Radial distortion is effects such as barrel, pincushion or mustache distortion, as described in (Jenkins and White, 1957). Tangential distortion appears when the camera sensor and the camera lens are not perfectly parallel. This implies that straight lines in the object space crossing the optical axis appear bent.

Refractive Distortion

Underwater, the distortion of an image can be exotic, due to refraction between several surfaces. On the uDrone, the camera has to perceive light that is coming from water, through acrylic glass, then through air. The refractive indexes of water, air and acrylic glass are, according to (Wikipedia, 2017b), given in table 2.1.

Table 2.1: Refractive indexes of water, air and acrylic glass.

Water, n_w	Air, n_a	Acrylic Glass, n_g
1.330	1.000293	1.491 ± 0.001

The refractive distortion will then be as depicted in figure 2.4.

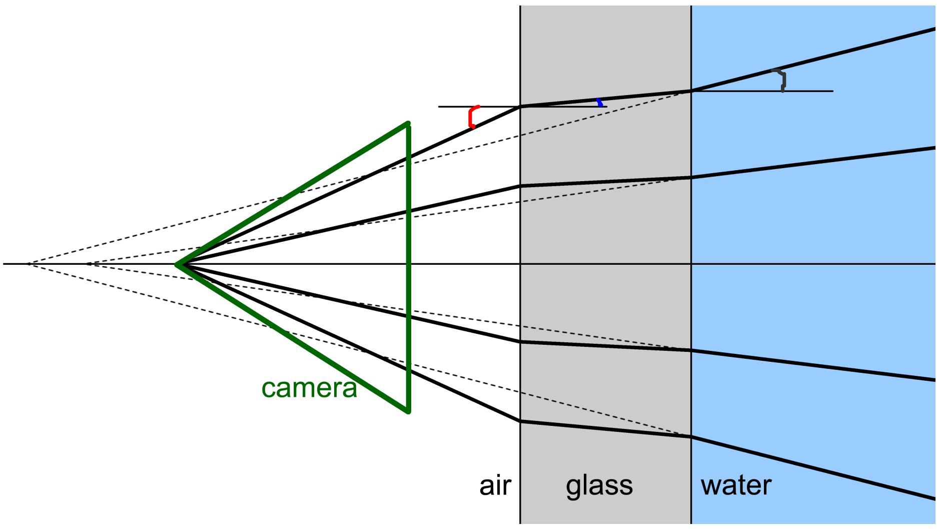
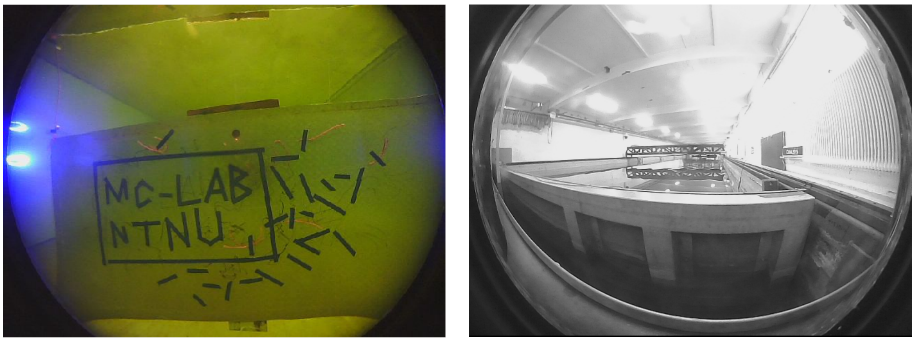


Figure 2.4: Illustration of refractive distortion due to water-acrylic glass and acrylic glass-air interfaces. Figure from (Processing, 2015).

The effect of the refractive distortion becomes visible by comparing images from the usb-camera on the uDrone above and below the water surface. In figure 2.5, we see that lines that should have been straight, such as the bottom of the plate in figure 2.5a and the edge at the bottom of figure 2.5b are distorted. We see that these lines, although they are straight in the real world, are distorted in different ways. The underwater image shows mustache distortion while the above water image shows barrel distortion.



(a) Underwater image.

(b) Above water image.

Figure 2.5: Images from the on board usb-camera on the uDrone, showing refractive distortion.

Due to the effect of refractive distortion, we have to perform the camera calibration underwater. For more on refractive distortion, see (Kunz and Singh, 2008) and (Kwon and Casebolt, 2006).

Calibration Software

Finding the constants needed for proper calibration is a highly involved process to preform analytically. We can therefore use available software for calibration, such as OpenCV, Adobe Photoshop or ROS. Usually these kinds of software need input in the form of images where a known pattern is in the image, usually a chessboard or a circle. In OpenCV, calibration is done by feeding images to the software, where a calibration is done recursively by calibrating for each image. The calibration usually relies on a set of images and converges to an optimal calibration, given that the known object is properly flat and the image set is varying in the objects placement in the image.

In this thesis, the ROS calibration software was used, (ROS, 2017). This was done by mounting a chessboard pattern on a plate which was submerged in water. The chessboard during above water calibration is shown in figure 2.6, this procedure was repeated underwater for underwater calibration.

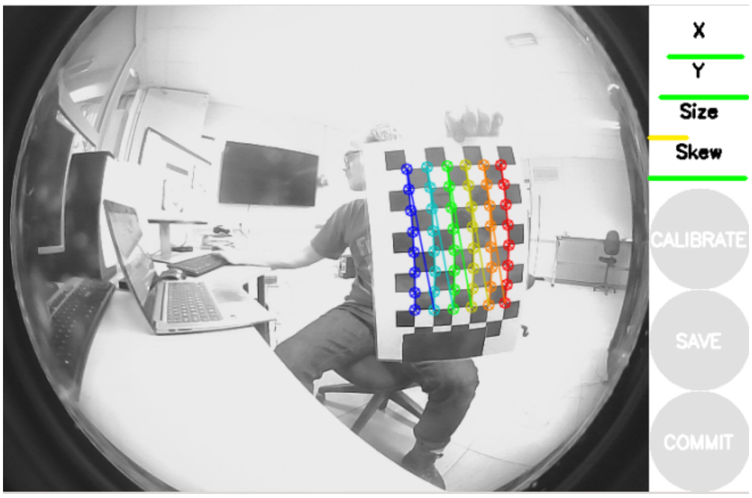


Figure 2.6: Above water calibration of the usb-camera, using the ROS camera calibration package.

Chapter 3

State Estimation

3.1 Process Model

3.1.1 Reference Frames

In this thesis, we will use three frames of reference, the body-fixed, world-fixed and the camera-fixed reference frame. We start by presenting the definitions of notation from SNAME (1950) in table 3.1. This notation describes the forces and velocities relative to the body-fixed reference frame and the pose relative some world-fixed frame of reference. The *North-East-Down*-frame(NED) is often used as the world-fixed frame of reference. However, since we are doing local positioning, we are bound to use the somewhat arbitrary initialization point for the SLAM-algorithm as the origin for our frame of reference.

Table 3.1: Notation for marine vessel position and motion from SNAME (1950)

Degree of Freedom	Force	Velocity	Position
Surge	X[N]	u	x[m]
Sway	Y[N]	v	y[m]
Heave	Z[N]	w	z[m]
Roll	K[Nm]	p	ϕ[rad]
Pitch	M[Nm]	q	θ[rad]
Pitch	N[Nm]	r	ψ[rad]

Relation Between Frames

From figure 3.1, we see that the camera center, $\boldsymbol{Cc}$, is offset from the body center of gravity, $\boldsymbol{CG}$, by $140mm$ along the body x-axis and $-20mm$ along the body z-axis. The camera frame is also rotated in relation to the body frame this rotation can be described by

$$\boldsymbol{x_c} = \boldsymbol{R}(\boldsymbol{\Theta}_b^c)\boldsymbol{x_b} + \boldsymbol{v_o} \tag{3.1}$$

where $\boldsymbol{x_c}$ is a position vector in the camera-frame, $\boldsymbol{R}_b^c(\boldsymbol{\Theta})$ is the rotation matrix describing the rotation between the two coordinate frames, $\boldsymbol{\Theta} \in \mathcal{S}^3$ is the vector of rotation between the frames, and $\boldsymbol{v_o}$ is the offset vector between the frames.

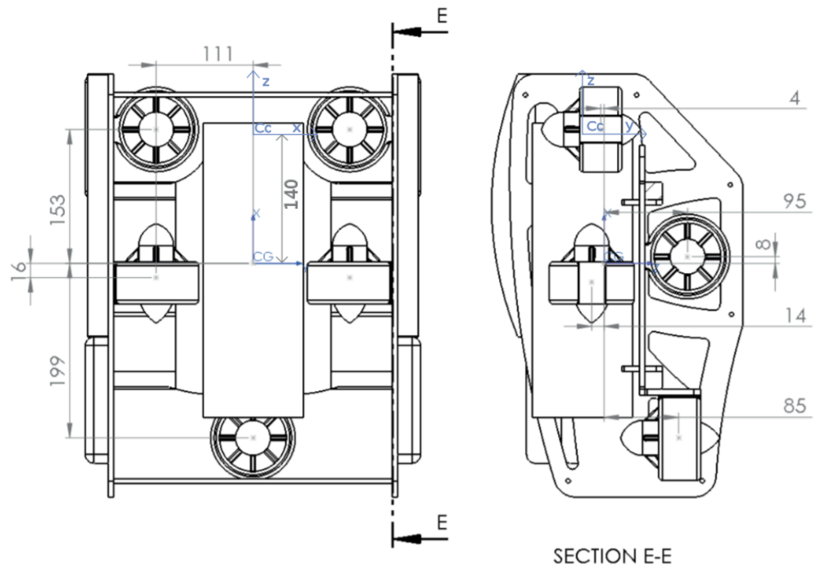


Figure 3.1: ROV uDrone with camera center, $\boldsymbol{Cc}$, and center of gravity, $\boldsymbol{CG}$, indicated. Original figure from Sandøy (2016), edited by author.

3.1.2 Kinematics Model

For the uDrone process model, we use the model presented in Sandøy (2016). This model is a 4 DOF model where surge, sway, heave, and yaw motion is included. Roll and pitch angles are excluded due to their inherent stability caused by the center of flotation of uDrone being above the center of gravity. Using the notation from Fossen (2011), we define $\boldsymbol{\eta}$ and $\boldsymbol{\nu}$, as they are used in this thesis, in (3.2) and (3.3).

$$\boldsymbol{\eta} := \begin{bmatrix} x & y & z & \psi \end{bmatrix}^{\mathsf{T}} \tag{3.2}$$
$$\boldsymbol{\nu} := \begin{bmatrix} u & v & w & r \end{bmatrix}^{\mathsf{T}} \tag{3.3}$$

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3.4}$$
$$\boldsymbol{M}\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} = \boldsymbol{\tau} \tag{3.5}$$

Where

$$\boldsymbol{M} = \boldsymbol{M_{RB}} + \boldsymbol{M_A} \tag{3.6}$$
$$\boldsymbol{C}(\boldsymbol{\nu}) = \boldsymbol{C_A}(\boldsymbol{\nu}) + \boldsymbol{C_{RB}}(\boldsymbol{\nu}) \tag{3.7}$$
$$\boldsymbol{D}(\boldsymbol{\nu}) = \boldsymbol{D_L} + \boldsymbol{D_{NL}}(\boldsymbol{\nu}) \tag{3.8}$$

Rewriting gives

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3.9}$$
$$\dot{\boldsymbol{\nu}} = \boldsymbol{M}^{-1}\left[\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}\right] \tag{3.10}$$

Further, we present the individual parts of the above equations. We define the position vector, $\boldsymbol{\eta}$, in the SLAM xyz-frame with one rotation ψ. For simplification, the remaining two degrees of freedom, ϕ and θ are not included. This frame relates to the *North-East-Down*-frame(NED) (Fossen, 2011), as it is earth-fixed, but has no global origin. If we know the global NED position of the SLAM origin, we can relate the two frames, however since our objective is local positioning, the local frame will be used. An important relation between the NED-frame and the SLAM-frame is that because of roll and pitch stability, it is assumed that the N- and z- axis of the two frames are parallel.

The *NED*-position, $\boldsymbol{\eta}$, and the body velocity, $\boldsymbol{\nu}$, make up the state-space of the mathematical model of the ROV. We will now present the remaining parts of the model. First we present the transformation between the body- and SLAM-frames, $\boldsymbol{J}(\boldsymbol{\eta})$, in (3.11).

$$\boldsymbol{J}(\boldsymbol{\eta}) = \begin{bmatrix} \boldsymbol{R}(\psi) & 0_{2\times2} \\ 0_{2\times2} & \boldsymbol{I}_{2\times2} \end{bmatrix} \tag{3.11}$$

$$\boldsymbol{R}(\psi) = \begin{bmatrix} cos(\psi) & -sin(\psi) \\ sin(\psi) & cos(\psi) \end{bmatrix} \tag{3.12}$$

We continue to the mass matrices for the rigid-body mass, $\boldsymbol{M_{RB}}$, and the added mass, $\boldsymbol{M_A}$. The rigid body mass is equal in all translation DOFs. In rotation DOFs the rigid-body mass is equal to the mass moment of inertia about the axis of rotation. The added mass matrix is dependent on the geometry of the vehicle and can be calculated using methods outlined in Sandøy (2016) or Faltinsen (1993). In Sandøy (2016), the added mass matrices are presented as in equations 3.13 and 3.14.

$$\boldsymbol{M_{RB}} = \begin{bmatrix} m \cdot \boldsymbol{I}_{3\times3} & 0 \\ 0 & I_{zz} \end{bmatrix} \tag{3.13}$$

$$\boldsymbol{M_A} = diag\left(\begin{bmatrix} -X_{\dot{u}} \\ -Y_{\dot{v}} \\ -Z_{\dot{w}} \\ -N_{\dot{r}} \end{bmatrix}\right) \tag{3.14}$$

$$\boldsymbol{D_L} = diag\left(\begin{bmatrix} X_u \\ Y_v \\ Z_w \\ N_r \end{bmatrix}\right) \tag{3.15}$$

$$\boldsymbol{D_{NL}}(\boldsymbol{\nu}) = diag\left(\begin{bmatrix} X_{|u|u}|u| + X_{uuu}u^2 \\ Y_{|v|v}|v| + Y_{vvv}v^2 \\ Z_{|w|w}|w| + Z_{www}w^2 \\ N_{|r|r}|r| + N_{rrr}r^2 \end{bmatrix}\right) \tag{3.16}$$

$$\tag{3.17}$$

In order to find the Coriolis matrices, we need the vectors $\boldsymbol{r}_g$ and $\boldsymbol{r}_b$, these vectors descrive the distance from the body frame to the *center of gravity*(CG) and to the *center og bouyancy*(CB) respectively. These were found for the uDrone in Sandøy (2016), and we present them here in table 3.2. Furthermore, we assume that off-diagonal added mass-terms are zero, this is due to the lack of information about the vessel. Using the Coriolis matrices presented in Fossen (2011), we find the Coriolis matrices $\boldsymbol{C_A}$ and

$\boldsymbol{C_{RB}}$

Table 3.2: $\boldsymbol{r}_g$ and $\boldsymbol{r}_b$ for ROV uDrone, as presented in Sandøy (2016).

Parameter	Value	Unit
$\boldsymbol{r}_g$	$[\ 0 \quad 0 \quad 0\ ]^\intercal$	[m]
$\boldsymbol{r}_b$	$[\ 0 \quad 0 \quad 0.00019\ ]^\intercal$	[m]

We define $\boldsymbol{r}_g^b$, as done in (Fossen, 2011), as the vector from CB, $\boldsymbol{r}_b$ to the CG, $\boldsymbol{r}_g$.

$$\boldsymbol{r}_g^b = [\ x_g \quad y_g \quad z_g\ ]^\intercal = [\ 0 \quad 0 \quad -0.00019\ ]^\intercal \tag{3.18}$$

We see that $\boldsymbol{r}_g^b$ has small values, and thus we choose to neglect any effect that is caused by the offset between the center of gravity and the center of buoyancy. Thus, we formulate the Coriolis matrices for added mass, $\boldsymbol{C_A}$, and rigid body, $\boldsymbol{C_{RB}}$, as done in (Fossen, 2011).

$$\boldsymbol{C_A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Z_{\dot{w}} \cdot w \\ 0 & 0 & 0 & -Y_{\dot{v}} \cdot v \\ 0 & -Z_{\dot{w}} \cdot w & Y_{\dot{v}} \cdot v & 0 \end{bmatrix} \tag{3.19}$$

$$\boldsymbol{C_{RB}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -m \cdot w \\ 0 & 0 & 0 & m \cdot v \\ 0 & m \cdot w & -m \cdot v & 0 \end{bmatrix} \tag{3.20}$$

From (3.19) and (3.20), we get the total Coriolis matrix, $\boldsymbol{C}(\boldsymbol{\nu})$.

$$\boldsymbol{C}(\boldsymbol{\nu}) = \boldsymbol{C_A} + \boldsymbol{C_{RB}} \tag{3.21}$$

The damping coefficients are presented in table 3.3

Table 3.3: Damping constants for ROV uDrone, as presented in Sandøy (2016).

DOF	Linear	Value	Quadratic	Value	Cubic	Value
Surge	X_u	-4.03	$X_{\lvert u \rvert u}$	-18.18	X_{uuu}	-24.24
Sway	Y_v	-6.22	$Y_{\lvert v \rvert v}$	-21.66	Y_{vvv}	-128.52
Heave	Z_w	-5.18	$Z_{\lvert w \rvert w}$	-36.99	X_{uuu}	-123.15
Yaw	N_r	-0.07	$N_{\lvert r \rvert r}$	-1.55	N_{rrr}	0

We use the parameters given for the ROV uDrone in Sandøy (2016), and present them here in table 3.4

Table 3.4: Mass and moment of inertia constant parameters for ROV uDrone a presented in Sandøy (2016).

Property	Notation	Value	Unit
Mass	m	7.31	$[kg]$
Yaw moment of inertia	I_{zz}	0.16	$[kgm^2]$
Surge added mass	$X_{\dot{u}}$	-5.50	$[kg]$
Sway added mass	$Y_{\dot{v}}$	-12.70	$[kg]$
Heave added mass	$Z_{\dot{w}}$	-14.57	$[kg]$
Yaw added moment of inertia	$N_{\dot{r}}$	-0.12	$[kgm^2]$

3.2 Scale

The scale of ORB-SLAM is somewhat arbitrary, depending on the initialization and camera calibration, as discussed in chapter 2. In order to have good estimates of the true states, we need to have a measurement of the relationship between the scale of the real world and the scale of ORB-SLAM. In this section, we present a method for calculating this relationship. We start with presenting the estimation of the real world range estimate using parallel lasers to an object. We then present the method of estimating the ORB-SLAM range to the same area as the lasers are pointed at, last we find and filter the relationship between the two estimates. Last, we present a method for estimating scale using depth offsets from pressure sensor data and ORB-SLAM.

3.2.1 Real World Range

Theory

In this section, we will use the notation presented in figure 2.3 to develop a method for finding the distance from a camera lens to a flat plane. Starting from the pinhole camera model, as presented in Wöhler (2012) and shown in figure 2.3, we can model two lines, $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, parallel with he camera z-axis and their projection into the image plane. Modeling the lines parallel to the camera z-axis, with offsets, a_1 and a_2, in the camera x-axis and no offset in the camera y-axis the lines become as presented in (3.22) and (3.23).

$$\boldsymbol{l}_1 = \begin{bmatrix} a_1 & 0 & z \end{bmatrix}^\mathsf{T} \tag{3.22}$$

$$\boldsymbol{l}_2 = \begin{bmatrix} a_2 & 0 & z \end{bmatrix}^\mathsf{T} \tag{3.23}$$

Using the transformation from camera coordinate frame to image plane, (3.24), presented in Wöhler (2012), we can find the relationship between the z-position of any point along a line and the $\hat{u}$-position in the image frame.

$$\frac{\hat{u}}{b} = \frac{x}{z} \tag{3.24}$$

Inserting $x = a_1$ and solving for z of a point along $\boldsymbol{l}_1$, $\boldsymbol{p}_1$, we get the relationship presented in (3.25).

$$z_{\boldsymbol{p}_1} = \frac{a_1 b}{\hat{u}_1} \tag{3.25}$$

Similarly, for a point along $\boldsymbol{l}_2$, $\boldsymbol{p}_2$, we get the relationship presented in equation 3.26.

$$z_{\boldsymbol{p}_2} = \frac{a_2 b}{\hat{u}_2} \tag{3.26}$$

Consider now that the two points, $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, are placed at the intersection of a plane, $\boldsymbol{A}$, not parallel to the z-axis of the camera coordinate system, and the lines $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, as shown in figure 3.2. Using the measurements from the image plane, we seek to find the point in which the optical axis intersects the plane. We call this point $\boldsymbol{p}_o = \begin{bmatrix} 0 & 0 & z_o \end{bmatrix}^\mathsf{T}$. In the special case where $\boldsymbol{A}$ is orthogonal to the optical axis, $z_o = z_{\boldsymbol{p}_1} = z_{\boldsymbol{p}_2}$, in all other cases $z_{\boldsymbol{p}_1}$ will be offset from z_o by a value, c, and $z_{\boldsymbol{p}_2}$ will have an offset equal to $\frac{a_2}{a_1}c$, due $\boldsymbol{A}$ being a flat plane. We can express this as in (3.27) and (3.28).

$$z_{\boldsymbol{p}_1} = z_o + c \tag{3.27}$$

$$z_{\boldsymbol{p}_2} = z_o + \frac{a_2}{a_1}c \tag{3.28}$$

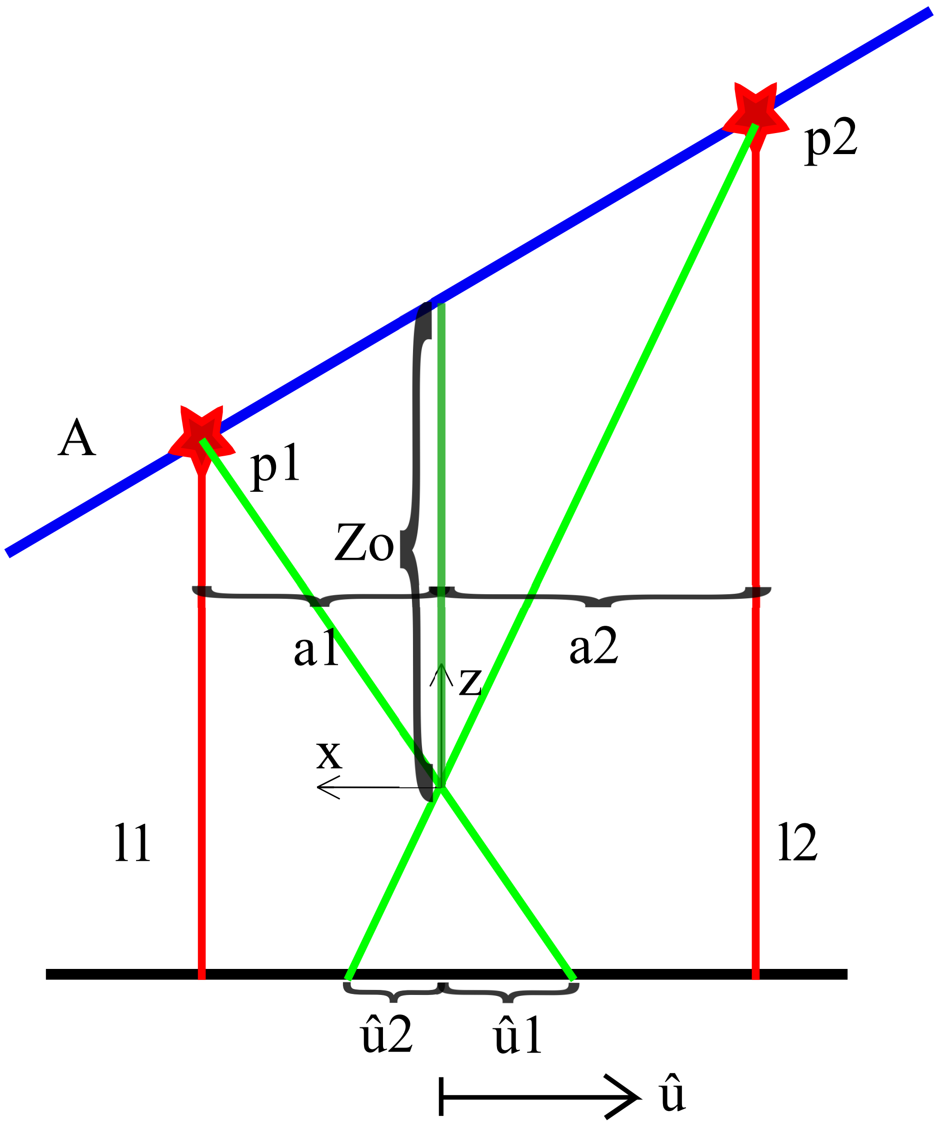


Figure 3.2: Two parallel lines, $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, intersecting a plane, $\boldsymbol{A}$, at points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$, and their projection on to the image plane.

Inserting (3.27) and (3.28) in (3.25) and (3.26) respectively, and solving for $\hat{u}_1$ and $\hat{u}_2$.

$$\hat{u}_1 = \frac{a_1 b}{z_o + c} \tag{3.29}$$

$$\hat{u}_2 = \frac{a_2 b}{z_o + \frac{a_2}{a_1} c} \tag{3.30}$$

Thus, we have two equations, (3.29) and (3.30), and two unknowns, c and z_o. We assume a_1, a_2, and b to be known, solving for z_o gives (3.31).

$$z_o = \frac{a_1 a_2 b}{a_1 - a_2} \left(\frac{1}{\hat{u}_2} - \frac{1}{\hat{u}_1} \right) \tag{3.31}$$

By (3.31), we see that z_o is not defined for $a_1 = a_2$, $\hat{u}_1 = 0$, or $\hat{u}_2 = 0$. This introduces some constraints in the use of parallel lines:

- a_1 and a_2 must be different values.
- $\hat{u}_n$ must be be nonzero for finite $z_{\boldsymbol{p}_n}$, for $n \in \{1, 2\}$.
- a_n must be be non-zero since it would render $\hat{u}_n$ to be zero for finite $z_{\boldsymbol{p}_n}$, for $n \in \{1, 2\}$.
- The points $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ on the surface $\boldsymbol{A}$ must be have a corresponding position in the camera coordinate frame, thus, $z_{\boldsymbol{p}_1}$ and $z_{\boldsymbol{p}_2}$ must be strictly positive values.

This derivation is illustrated in figure 3.2, we will now present a practical application for this theory.

Solving for Constants

We seek to find the distance from the Raspberry Pi camera mounted on the uDrone to the object its optical axis is pointed at. In order to do this, we assume the following:

- The object is a flat surface(e.g. a wall).
- The distance, z_o, is always positive since the camera cannot observe things behind it.
- The lasers mounted on the uDrone are perfectly parallel.
- The $\hat{v}$-value of the lasers in the camera coordinate frame is always zero.
- The Raspberry Pi camera module behaves like a pinhole camera.

In order to find the constant term in (3.31), the distance form the camera to a wall was measured using a tape measure. The term $abs\left(\frac{1}{\hat{u}_2} - \frac{1}{\hat{u}_1}\right)$ was logged at four different distances, presented in table 3.5. Since we assume a positive distance, the absolute value is taken. Linear regression was then performed on the data points, and the results are presented in figure 3.3. From the linear regression we obtain the expression in (3.32).

Data point	z_o[m]	$abs\left(\frac{1}{\hat{u}_2} - \frac{1}{\hat{u}_1}\right)$
1	0.35	0.0440411
2	0.50	0.0626376
3	0.80	0.102853
4	1.00	0.129572

Table 3.5: Measurements for linear regression of laser range equation estimation.

$$z_o = 7.569abs\left(\frac{1}{\hat{u}_2} - \frac{1}{\hat{u}_1}\right) + 0.02[m] \tag{3.32}$$

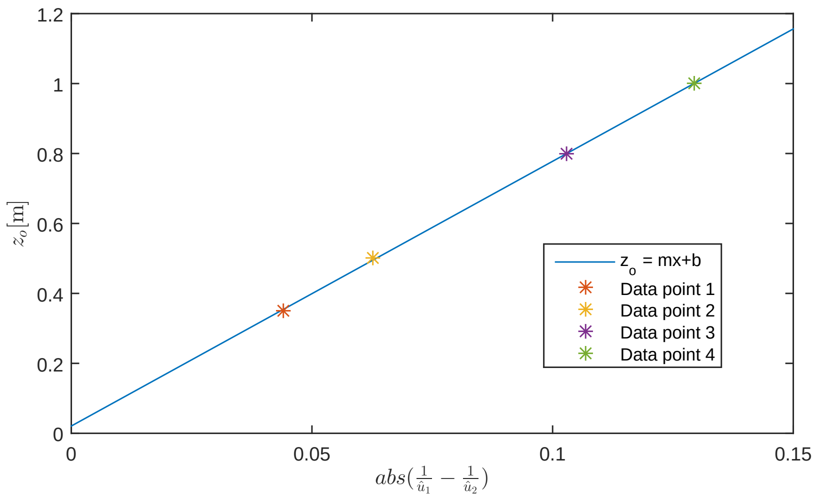


Figure 3.3: Linear regression of relationship between z_o and $abs(\frac{1}{\hat{u}_1} - \frac{1}{\hat{u}_2})$

Discussion

As the constant term, $0.02m$, in equation 3.32 is small, we can choose to neglect it in the final range estimation. As (3.31) has no additional terms, the source of the constant might be measurement error and modeling errors, as the Raspberry Pi camera is not a true pinhole camera. A true pinhole camera has an infinitesimally small lens, which is not possible in the real world as stated in Wöhler (2012). The tests were also conducted in air, not water. In order to verify if the distance estimation behaves well under water, we do an experiment. We measure the true distance and compare with the estimated distance to flat plate. The results of the experiment are presented in section 4.2.

3.2.2 Computer Vision and Lasers

As described in Henriksen (2016), we can find the lasers in an image using HSV-colors. Using the OpenCV (OpenCv, 2017) library in conjunction with ROS, we can run the computer vision algorithms on the topside processing unit. This enables us to run the algorithm at $30Hz$ or more if needed. In this section we will present the methods used for laser detection in the image frame. We start by presenting the OpenCV library and continue to present the HSV color scheme, last we present the method used for automatically finding the laser points in the image.

OpenCV

As stated on the home page of OpenCV:

> *OpenCV is released under a BSD license and hence it's free for both academic and commercial use. [...] Adopted all around the world, OpenCV has more than 47 thousand people of user community and estimated number of downloads exceeding 14 million. Usage ranges from interactive art, to mines inspection, stitching maps on the web or through advanced robotics.*
> -OpenCV Home page

OpenCV is an open-source library for computer vision for the C, C++, and Python programming languages. It is supported by a large community of users and contributors, making almost any computer vision task solvable. This makes OpenCv he go-to library for computer vision. Other libraries, such as libCVD or ccv could be considered. However, OpenCV seemed to be the best choice. In this project, computer vision is used extensively as ORB-SLAM, laser detection, and video pipelines relies on it.

HSV-colors

Hue Saturation Value(HSV) color representation (Smith, 1978), illustrated in figure 3.4, is an alternative to the classic *Red Gree Blue*(RGB) color scheme used in most image processing (Hunt, 1967). The RGB color scheme is intuitive in that it represents each of the primary colors, red, green and blue with a number, $\{R, G, B\} \in \{0, 255\}$ in a way such that $\{255, 0, 0\}$ is red, $\{0, 255, 0\}$ is green, and $\{0, 0, 255\}$ is blue.
In the HSV color-scheme the hue, saturation and value of the color is represented, in stead of the color combination of red, green, and blue. The hue, $H \in [0, 360]$, value determines the "color" or hue of the color. A low hue value corresponds to a red color and as the value increases, the hue moves through the rainbow towards violet. The saturation, $S \in [0, 180]$ determines the deepness of the color, its intensity. Staring from $S = 0$ where there is no color to the maximum intensity of $S = 180$. Finally the value, $V \in [0, 1]$ determines the brightness or lightness of the color. At $V = 0$ the color is black, and going towards $V = 1$, the color becomes lighter and lighter, until there is no black left in the color at $V = 1$. This is illustrated in figure 3.4. The properties of the HSV-image make it easy to pick out colors with a certain hue from an image within a range of saturations and brightnesses and vice versa. This will be helpful in locating the lasers in the image.

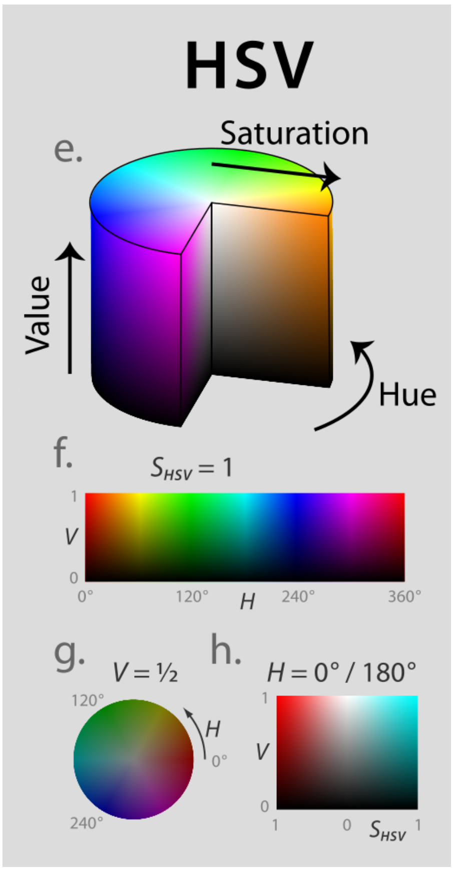


Figure 3.4: HSV color-scheme, image source: (Wikipedia, 2017a), edited by author.

Laser Localization

In order to locate the laser points in the image frame of the Raspberry Pi camera, we need to filter out all other sources of light and disturbances. Due to the above men-

]

tioned properties of the HSV-color scheme, we choose it for filtering. First, the image is cropped to include the area where the lasers are to be found in order to lessen the computational load and remove unnecessary noise. We choose to crop more than what was done in (Henriksen, 2016), as it was found that the lasers kept inside the cropped image for all relevant distances from the camera to the object in front of it.

```
this->myMatImage1 = this->myMatImage1(Rect(20,130,600,100));
```

After cropping, the image is converted into an HSV-image. Then, a filter is applied filtering out all pixels whose colors are not within a dynamic range. The range of the H, S and V values is determined by algorithm 2. It is important to note that the range of H, S, and V is changed as we move in to computer vision, where the values are discretized to integers such that $\{H, S, V\} \in \{0, 255\}$. This is convenient for computing, as each value can be described by one byte.

Algorithm 2 Automatic color range calibration for detecting laser points in HSV-image.

```
**Initialization**
```

$$\begin{aligned} H_{min} &= 0 & H_{max} &= 25 \\ S_{min} &= 0 & S_{max} &= 120 \\ V_{min} &= 100 & V_{max} &= 255 \end{aligned} \tag{3.33}$$

```
for each new image do
    if points detected in last image $< 2$ then
        Increase $H_{max}$ and $S_{max}$ by 1
        Decrease $V_{min}$ by 1
    else if points detected in last image $> 2$ then
        Increase $V_{min}$ by 1
        Decrease $H_{max}$ and $S_{max}$ by 1
    else
        All values remain the same
    end if
end for
```

After the for the filter is determined, the image is filtered using the inRange-function from OpenCV, documentation available in (OpenCV, 2017b).

```
cv::inRange(matOriginalHSV, cv::Scalar(hmin, smin, vmin),
cv::Scalar(hmax, smax, vmax), matProcessed);
```

The processed image is then an image with only black or white pixels. White pixels indicate pixels that were within the range, and black pixels indicate pixels that were outside the range.

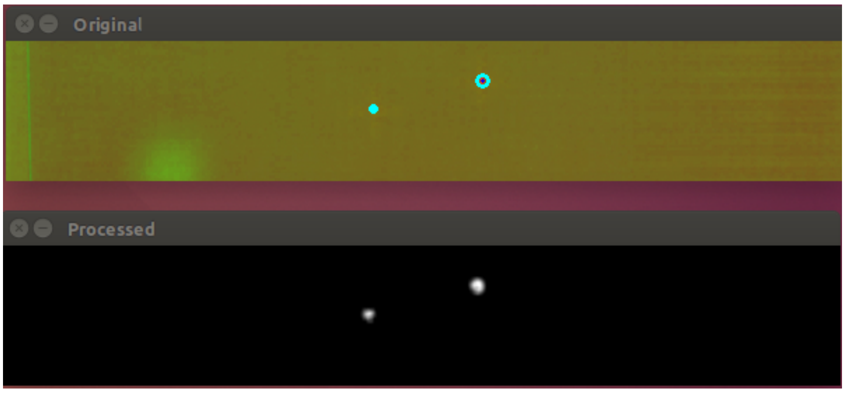


Figure 3.5: Original HSV-image with Hough circles overlaid and the processed, blurred image after range filtering. Two laser dots appear in both images.

We call this image the processed image, or matProcessed in the code. After filtering, the processed image is blurred with Gaussian blur.

```
cv::GaussianBlur(matProcessed, matProcessed, cv::Size(5, 5), 2.2);
```

After blurring, the image is searched for Hough Circles, presented in (Yuen et al., 1990) and (OpenCV, 2017a), that have a radius, r, that is in the range $8 \leq r \leq 30$ pixels. The radius range is limited at 8 pixels in order to reduce the effect of image noise. The upper bound of 30 pixels is chosen as to not include other circles present in the image, that may not have been filtered out. The original image containing two laser dots, with overlaid circles and the filtered, blurred image are presented in figure 3.5.

```
 cv::HoughCircles()
```

This function return a vector of circles found in the blurred image. If two circles were found, the distances $\hat{u}_1$ and $\hat{u}_2$ are found and the distance z_o is calculated using (3.32).

Outlier Rejection

There are several error modes in the above presented approach. In order to handle false and bad measurements of z_o, we need some way to reject the bad measurements. First, we present the known sources of bad measurements, second, we present the methods developed for handling them.

The most common source of bad measurements is the detection of a Hough circle where there is no laser. The reason for this may be bad filtering of the image, light scattering from the laser or image noise. Another source may be adjustments bade by the camera, such as automatic white balance and shutter time, causing the color of the laser points to shift out from the range we are filtering. We propose the outlier rejection algorithm presented in 3 in order to compensate for the effects of wrongful measurements.

Algorithm 3 Automatic outlier rejection of laser range measurements

```
for each new image do
    Solve for $z_{o,t}$
    if $|\dot{z}_{o,t}| < 0.3m/s$ then
        Keep $z_{o,t}$
    else
        Discard $z_{o,t}$
    end if
end for
```

The calculation of $\dot{z}_{o,t}$ in algorithm 3 is done by Euler differentiation, as presented in equation 3.34. The choice of a threshold of $0.3m/s$ is based on the heuristic of maximum expected surge velocity of uDrone.

$$\dot{z}_{o,t} = \frac{z_{o,t} - z_{o,t-n}}{n \cdot h} \tag{3.34}$$

$$h = \frac{1}{fps} \tag{3.35}$$

In (3.34), the term $z_{o,t-n}$ denotes the last inlier, n denotes the number of frames that have passed since the last inlier measurement, and in (3.35), fps denotes the frame rate, or *frames per second*, of the image stream.

The outlier rejection procedure has some flaws due to its simplicity. First, the laser points may jump from some nearby object to some object in the distance, making $|\dot{z}_{o,t}| > 0.3m/s$, thus rejecting a true measurement. Second, there may be bad measurements that are treated as inlier measurements, however, these would not cause a great change in z_o, as the rate of change is the restricting factor.

3.2.3 Depth Measurement

In (Sandøy, 2016), a depth estimator was made based on the measurements from the pressure sensor on the external IMU. The depth is estimated using the hydrostatic relation $P = \rho g h$, where P is the pressure in pascal, ρ is the density of the medium in $\frac{kg}{m^3}$, and h is the depth in meters. When submerging something in water on the surface of the earth, we have to account for the atmospheric pressure, P_{atm}, such that the depth of the drone below the water surface can be found by (3.36).

$$h = \frac{P - P_{atm}}{\rho g} \tag{3.36}$$

In (3.36), the atmospheric pressure is set to $P_{atm} = 101 kPa$, the water density is set to $\rho = 1000 \frac{kg}{m^3}$, and the gravitational acceleration os set to $g = 9.81 \frac{m}{s^2}$. The last term, P, is the pressure sensed by the pressure sensor.

3.2.4 ORB-SLAM Range

In order to get a notion of the scale of the ORB-SLAM reference frame and compare it to the real-world. In the ORB-SLAM frame, we measure the distance from the current camera position, $\boldsymbol{p_{cc}}$, to map points, $\boldsymbol{M_p} = [\ \boldsymbol{p}_1 \quad \boldsymbol{p}_2 \quad ... \quad \boldsymbol{p}_N\ ]^\intercal$, located in an area in the center of the image frame. This area is then assumed to be centered around the laser points discussed in section 3.2.1. This distance is assumed to be the average distance from the camera to the map points. In order to measure this distance, we calculate the Euclidean norm for each vector and average the norms.

$$\bar{d} = \frac{\sum_{i=1}^{N} \|\boldsymbol{p_{cc}} - \boldsymbol{p}_i\|}{N} \tag{3.37}$$

Implicit in (3.37) is the assumption that the distance from the camera to the map points are Gaussian distributed. To check if this assumption is valid, we examine the estimated position relative to the measured position in section 4.2.s

3.2.5 Comparison and Filtering

In this thesis, we propose two approaches for estimating the scale of ORB-SLAM. First, by directly comparing and filtering the laser range measurements presented in section 3.2.1 and the and the ORB-SLAM range presented in section 3.2.4. Second, by comparing the offset in heave from the initialization point as estimated by ORB-SLAM and by the pressure sensor. In this section we will present both methods.

Scale From Laser Range Measurements

Scale is obtained by comparing the distance to the same area measured in the ORB-SLAM frame and by laser range measurements. The approach is based on the assumption that ORB-SLAM has found map points in the area where the lasers are measuring the distance. Furthermore, it is assumed that this area is relatively flat. When comparing, we get the scaling factor, s_l, by dividing the laser range measurements, obtained by (3.32), by the ORB-SLAM range measurements, obtained by (3.37).

$$s_l = \frac{z_o}{\bar{d}}[m] \tag{3.38}$$

We give the scaling factor the unit of meter in (3.38), due to the lack of any knowledge about the ORB-SLAM units of length. For each update of z_o or $\bar{d}$, s_l is updated, generating an array of scaling factors, $\boldsymbol{s_l} = \begin{bmatrix} s_{l,1} & s_{l,2} & ... & s_{l,k} \end{bmatrix}$. The scaling factor is then low pass filtered in a discrete low pass filter, presented in (3.39), where $\beta \in (0,1)$ is set to $\beta = 0.3$. For each new measurement of z_o or $\bar{d}$, the filter is updated.

$$s_{lf,k} = \beta s_{l,k} + (1-\beta) s_{lf,k-1} \tag{3.39}$$

In this case, a cutoff frequency of $\omega_c = 4.3[rad/s]$ was found to be suitable, as the scale drift of ORB-SLAM is assumed to be a slowly varying process.

Scale From Heave Offset

In comparing the heave offsets in the ORB-SLAM frame and from the depth measurements, a depth measurement is made when ORB-SLAM is initialized. This leads to the z-position in the ORB-SLAM frame is equal to its offset. This is based on the assumption that ORB-SLAM is initialized with sufficiently small roll and pitch angles, such that the z unit vector in ORB-SLAM is approximately parallel to the gravitational acceleration vector in the NED-frame. Depth is measured by (3.36) at the same time as ORB-SLAM is initialized, the initial depth is denoted h_0. For each new measurement of z or h, the heave scaling factor, s_h, is updated by (3.40).

$$s_h = \frac{h - h_0}{z} \tag{3.40}$$

As the drone comes closer ot the ORB-SLAM xy-plane, the scaling factor will become more sensitive to measurement noise and eventual biases in h_0. This effect os minimized by rejecting all measurements of z or $h - h_0$ that lies within a range around zero. This approach is presented in algorithm 4, where $sign(\cdot)$ outputs the sign of its argument.

Algorithm 4 Algorithm for rejecting small measurements of z and $h - h_0$.

```
for Each new $s_h$ do
    if $|h - h_0| < 0.05m$ or $|z| < 0.05$ then
        Reject $s_h$
    else if $sign(h - h_0)$ not equal to $sign(z)$ then
        Reject $s_h$
    else
        Keep $s_h$
    end if
end for
```

Measurements that pass through algorithm 4 are then low pass filtered, using the same approach as presented in (3.39). However, this filter is running at a fixed frequency, $10Hz$. This means that we can calculate the cutoff frequency, ω_c, for $\beta = 0.3$ by (3.41). The cutoff frequency was found to be $\omega_c = 4.3[rad/s]$, this is high, but easily tunable.

$$\beta = \frac{\omega_c \Delta T}{1 + \omega_c \Delta T} \tag{3.41}$$

3.3 Observers

When using ORB-SLAM, one only obtains the relative pose from the current camera position relative to the initialization frame as stated in (Mur-Artal et al., 2015). Furthermore, the current frame has a delay of about $350ms$ as stated in (Henriksen, 2016). This means we have no real measurement of the current velocity and position. Using an observer, we can estimate the velocities. In order to do so, we must first check if the system is observable.

3.3.1 Observability

In order to make a state observer, our system must be observable. We start with the system model presented in section 3.1 and augment it to include bias estimation as described in (Sørensen, 2012).

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{\eta} \\ \boldsymbol{\nu} \\ \boldsymbol{b} \end{bmatrix} \qquad \boldsymbol{y} = \boldsymbol{\eta} \tag{3.42}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} 0_{4\times4} & \boldsymbol{J}(\boldsymbol{\eta}) & 0_{4\times4} \\ 0_{4\times4} & -\boldsymbol{M}^{-1}\left[\boldsymbol{C}(\boldsymbol{\nu}) + \boldsymbol{D}(\boldsymbol{\nu})\right] & \boldsymbol{M^{-1}J}(\boldsymbol{\eta}) \\ 0_{4\times4} & 0_{4\times4} & -\boldsymbol{T}_b^{-1} \end{bmatrix} \boldsymbol{x} + \begin{bmatrix} 0_{4\times4} \\ \boldsymbol{M}^{-1} \\ 0_{4\times4} \end{bmatrix} \boldsymbol{\tau} \tag{3.43}$$

From (3.42) and (3.43) we get the $\boldsymbol{A}$ and $\boldsymbol{C}$ matrices presented in (3.44).

$$\boldsymbol{A} = \begin{bmatrix} 0_{4\times4} & \boldsymbol{J}(\boldsymbol{\eta}) & 0_{4\times4} \\ 0_{4\times4} & -\boldsymbol{M}^{-1}\left[\boldsymbol{C}(\boldsymbol{\nu}) + \boldsymbol{D}(\boldsymbol{\nu})\right] & \boldsymbol{M^{-1}J}(\boldsymbol{\eta}) \\ 0_{4\times4} & 0_{4\times4} & -\boldsymbol{T}_b^{-1} \end{bmatrix} \tag{3.44}$$

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{I}_{4\times4} & 0_{4\times4} & 0_{4\times4} \end{bmatrix} \tag{3.45}$$

From the matrices $\boldsymbol{A}$ and $\boldsymbol{C}$, we check the rank of the observability matrix, $\boldsymbol{\mathcal{O}}$.

$$\boldsymbol{\mathcal{O}} = \begin{bmatrix} \boldsymbol{C} \\ \boldsymbol{CA} \\ \boldsymbol{CA^2} \end{bmatrix} = \begin{bmatrix} \boldsymbol{I}_{4\times4} & 0_{4\times4} & 0_{4\times4} \\ 0_{4\times4} & \boldsymbol{J}(\boldsymbol{\eta}) & 0_{4\times4} \\ 0_{4\times4} & -\boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{M}^{-1}\left[\boldsymbol{C}(\boldsymbol{\nu}) + \boldsymbol{D}(\boldsymbol{\nu})\right] & \boldsymbol{J}(\boldsymbol{\eta})^2 \end{bmatrix} \tag{3.46}$$

The system is observable if $\boldsymbol{\mathcal{O}}$ has full rank for all $\boldsymbol{x}$. We see that is the case if and only if $\boldsymbol{J}(\boldsymbol{\eta})$ and $\boldsymbol{J}(\boldsymbol{\eta})^2$ has full rank. Since the rotation matrix in $\boldsymbol{J}(\boldsymbol{\eta})$ has full rank for all $\boldsymbol{\eta}$(see appendix, section A.1.2) and all other elements are diagonal, we can conclude that $\boldsymbol{J}(\boldsymbol{\eta})$ has full rank. Furthermore, we can prove that $\boldsymbol{J}(\boldsymbol{\eta})^2 = \boldsymbol{J}(\boldsymbol{2\eta})$(see appendix, section A.1.1) and by the same logic, $\boldsymbol{J}(\boldsymbol{\eta})^2$ must have full rank for all $\boldsymbol{\eta}$.

3.3.2 Extendend Kalman Filter

In order to estimate the velocity and reject measurement and process noise, we will use the discrete time Extended Kalman Filter(EKF). In this section we will present the algorithm as it is presented in (Fossen, 2011), and comment on the implementation in our specific case.

General EKF

The EKF is the nonlinear version of the Kalman Filter, linearizing for each time step. It applies to systems of the form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{B}\boldsymbol{u} + \boldsymbol{E}\boldsymbol{w} \tag{3.47}$$

$$\dot{\boldsymbol{y}} = \boldsymbol{H}\boldsymbol{x} + \boldsymbol{v} \tag{3.48}$$

Where $\boldsymbol{f}(\boldsymbol{x})$ is a nonlinear vector field. Furthermore, $\boldsymbol{w}$ is the process noise, and $\boldsymbol{v}$ is the measurement noise. Both are assumed to be zero-mean Gaussian white noise. The discrete-time linearizations are found by forward Euler integration:

$$\mathcal{F}(\hat{\boldsymbol{x}}(k), \boldsymbol{u}(k)) \approx \hat{\boldsymbol{x}}(k) + h\left[\boldsymbol{f}(\hat{\boldsymbol{x}}(k)) + \boldsymbol{B}\boldsymbol{u}(k)\right] \tag{3.49}$$

$$\boldsymbol{\Phi}(k) \approx \boldsymbol{I} + h \left.\frac{\partial \boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k))}{\partial \boldsymbol{x}(k)}\right|_{\boldsymbol{x}(k)=\hat{\boldsymbol{x}}(k)} \tag{3.50}$$

$$\boldsymbol{\Gamma}(k) \approx h\boldsymbol{E} \tag{3.51}$$

where h is the sample time.

Algorithm 5 Discrete-time extended Kalman Filter

Initialization

$$\boldsymbol{Q} = \boldsymbol{Q}^{\mathsf{T}} > 0 \tag{3.52}$$

$$\boldsymbol{R} = \boldsymbol{R}^{\mathsf{T}} > 0 \tag{3.53}$$

$$\bar{\boldsymbol{x}}(0) = \boldsymbol{x}_0 \tag{3.54}$$

$$\bar{\boldsymbol{P}}(0) = E\left[(\boldsymbol{x}(0) - \hat{\boldsymbol{x}}(0))(\boldsymbol{x}(0) - \hat{\boldsymbol{x}}(0))^{\mathsf{T}}\right] = \boldsymbol{P}_0 \tag{3.55}$$

For each time step do

$$\boldsymbol{K}(k) = \bar{\boldsymbol{P}}(k)\boldsymbol{H}^{\mathsf{T}}(k)\left[\boldsymbol{H}(k)\bar{\boldsymbol{P}}(k)\boldsymbol{H}^{\mathsf{T}}(k) + \boldsymbol{R}(k)\right]^{-1} \tag{3.56}$$

$$\hat{\boldsymbol{x}}(k) = \bar{\boldsymbol{x}}(k) + \boldsymbol{K}(k)\left[\boldsymbol{y}(k) - \boldsymbol{H}(k)\bar{\boldsymbol{x}}(k)\right] \tag{3.57}$$

$$\hat{\boldsymbol{P}}(k) = \left[\boldsymbol{I} - \boldsymbol{K}(k)\boldsymbol{H}(k)\right]\bar{\boldsymbol{P}}(k)\left[\boldsymbol{I} - \boldsymbol{K}(k)\boldsymbol{H}(k)\right]^{\mathsf{T}} + \boldsymbol{K}(k)\boldsymbol{R}(k)\boldsymbol{K}^{\mathsf{T}}(k) \tag{3.58}$$

$$\bar{\boldsymbol{x}}(k+1) = \mathcal{F}(\hat{\boldsymbol{x}}(k), \boldsymbol{u}(k)) \tag{3.59}$$

$$\bar{\boldsymbol{P}}(k+1) = \boldsymbol{\Phi}(k)\hat{\boldsymbol{P}}(k)\boldsymbol{\Phi}^{\mathsf{T}}(k) + \boldsymbol{\Gamma}(k)\boldsymbol{Q}(k)\boldsymbol{\Gamma}^{\mathsf{T}}(k) \tag{3.60}$$

The matrices $\boldsymbol{Q}$ and $\boldsymbol{R}$ correspond to the covariance of the process and measurement noise, respectively. The matrix $\boldsymbol{P}$ represents the error covariance, and it can be shown that if

$$p_{min}\boldsymbol{I} \leq \boldsymbol{P}(t) \leq p_{max}\boldsymbol{I} \tag{3.61}$$

$$p_{max} > 0 \tag{3.62}$$

$$p_{min} > 0 \tag{3.63}$$

the continuous time EKF is incremental globally exponentially stable, as stated in (Fossen, 2011). That is, the error state converges exponentioally to zero or the estimated states converge to the actual states.

3.3.3 Model Specific EKF for uDrone

In order to estimate the velocity of the drone, we choose to use an EKF, in this section we will present the model used in the EKF. It is based on the model presented in section 3.1, for simplification, the Coriolis term, $\boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu}$, is removed. As the objective of this thesis is dynamic positioning at low speeds, the cross terms of the Coriolis matrix will become minimal. However, in maneuvering at speed it will become relevant. We also add a bias state, $\boldsymbol{b}$, in the system to compensate for constant or slowly varying unmodeled dynamics. The bias dynamics are of first order, as done in (Sørensen, 2012). Furthermore, the bias is largely a noise driven process, as unmodeled dynamics are treated as model noise. The bias can also be modeled as zeroth-order dynamics, or purely noise driven, this means that the bias can grow unbounded. We choose to restrain the growth by choosing large $\boldsymbol{T}_b$, such that the bias dynamics in (3.66) become globally exponentially stable, but slowly converging in absence of noise.

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3.64}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{M}^{-1}\left[\boldsymbol{\tau} - \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}\right] + \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{b} \tag{3.65}$$

$$\dot{\boldsymbol{b}} = -\boldsymbol{T}_b^{-1}\boldsymbol{b} + \boldsymbol{E}_b\boldsymbol{w}_b \tag{3.66}$$

$$\boldsymbol{y} = \boldsymbol{\eta} + \boldsymbol{v} \tag{3.67}$$

Using (3.64)-(3.67) and our knowledge of the measurements, we can find $\boldsymbol{f}(\boldsymbol{x})$, $\boldsymbol{B}$, $\boldsymbol{E}$, and $\boldsymbol{H}$. $\boldsymbol{T}_b$ is a diagonal matrix containing the time constants for bias convergence. As the diagonal entries in $\boldsymbol{T}_b$ increase, the convergence rate of the bias states in absence of white noise will increase. We start by defining $\boldsymbol{x}$ in (3.68).

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{\eta}^{\mathsf{T}} & \boldsymbol{\nu}^{\mathsf{T}} & \boldsymbol{b}^{\mathsf{T}} \end{bmatrix}^{\mathsf{T}} \tag{3.68}$$

We move on to define $\boldsymbol{f}(\boldsymbol{x})$ in equation 3.69.

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{f}_1 \\ \boldsymbol{f}_2 \\ \boldsymbol{f}_3 \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \\ -\boldsymbol{M}^{-1}\left[\boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{J}(\boldsymbol{\eta})^{\intercal}\boldsymbol{b}\right] \\ -\boldsymbol{T}_b^{-1}\boldsymbol{b} \end{bmatrix} \tag{3.69}$$

The matrices $\boldsymbol{B}$ and $\boldsymbol{H}$ are defined next in equations 3.70 and 3.71.

$$\boldsymbol{B} = \begin{bmatrix} 0_{4\times4} \\ \boldsymbol{M}^{-1} \\ 0_{4\times4} \end{bmatrix} \tag{3.70}$$

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_{4\times4} & 0_{4\times4} & 0_{4\times4} \end{bmatrix} \tag{3.71}$$

In order to estimate how the states propagate, we need to define $\mathcal{F}(\hat{\boldsymbol{x}}(k), \boldsymbol{u}(k))$, it is approximated by forward Euler integration in (3.72).

$$\mathcal{F}(\hat{\boldsymbol{x}}(k), \boldsymbol{u}(k)) \approx \hat{\boldsymbol{x}}(k) + h\left[\boldsymbol{f}(\hat{\boldsymbol{x}}(k)) + \boldsymbol{B}\boldsymbol{u}(k)\right] \tag{3.72}$$

Next, we need to define $\boldsymbol{\Phi}(k)$, this is done using the first order Taylor expansion of $\boldsymbol{f}(\boldsymbol{x})$, as presented in (3.73).

$$\boldsymbol{\Phi}(k) \approx \boldsymbol{I}_{12\times12} + h\frac{\partial \boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k))}{\partial \boldsymbol{x}(k)} \tag{3.73}$$

Equation 3.73 calls for the differentiation of $\boldsymbol{f}(\boldsymbol{x})$, this differentiation is presented in the following.

$$\frac{\partial \boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k))}{\partial \boldsymbol{x}(k)} = \begin{bmatrix} \frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{\eta}} & \frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{\nu}} & \frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{b}} \\ \frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{\eta}} & \frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{\nu}} & \frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{b}} \\ \frac{\partial \boldsymbol{f}_3}{\partial \boldsymbol{\eta}} & \frac{\partial \boldsymbol{f}_3}{\partial \boldsymbol{\nu}} & \frac{\partial \boldsymbol{f}_3}{\partial \boldsymbol{b}} \end{bmatrix} \tag{3.74}$$

The individual elements of (3.74) are presented in the appendix, section A.2.1.The co-variance matrices, $\boldsymbol{R}$ and $\boldsymbol{Q}$, are presented in the appendix, section A.2.2.

3.3.4 Kinematic EKF

After the model specific EKF was implemented and tested, it was found not to give satisfactory linear velocity estimates. We therefore propose to include *Inertial Measurement Unit*(IMU)-measurements in order to have a better estimate of the linear velocities. In this section we will present the new EKF structure, starting by presenting the IMU.

IMU

The IMU installed in the uDrone is mounted as near as possible to the center of mass. It measures all linear accelerations in the body-frame, along with angular velocities and angles. In this implementation, we will use the linear accelerations along with the roll and pitch angles. The angles are used for canceling the gravitational acceleration experienced by the IMU. We present this approach in (3.75), where $\boldsymbol{y_{IMU}}$ is the measurement vector compensated for gravity, $\boldsymbol{y^*_{IMU}}$ is the raw measurement, $\boldsymbol{R^b_n}(\phi, \theta)$ is the rotation matrix for roll and pitch from the NED-frame to the body-frame, where roll, ϕ, and pitch, θ, are measured by the IMU, and $\boldsymbol{g}$ is the gravitational acceleration vector in the NED-frame.

$$\boldsymbol{y_{IMU}} = \begin{bmatrix} \dot{u}_{IMU} & \dot{v}_{IMU} & \dot{w}_{IMU} \end{bmatrix}^\intercal = \boldsymbol{y^*_{IMU}} - \boldsymbol{R^b_n}(\phi, \theta)\boldsymbol{g} \tag{3.75}$$

EKF formulation

The kinematic EKF is formulated by starting with the model specific EKF and switching the model based accelerations for measurements done by the IMU. In our case this means modifying the $\boldsymbol{f_2}$-term in (3.69), we call this term $\boldsymbol{g_2}$. We keep the model based angular acceleration model and exchange the linear accelerations as presented in (3.76).

$$\boldsymbol{g_2}(\boldsymbol{x}, \boldsymbol{y_{IMU}}) = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \dot{u}_{IMU} \\ \dot{v}_{IMU} \\ \dot{w}_{IMU} \\ \frac{Z - N_r r - N_{r|r|} r|r|}{I_{zz} - N_{\dot{r}}} \end{bmatrix} + \boldsymbol{J}(\boldsymbol{\eta})^\intercal \boldsymbol{b} + \boldsymbol{E_{IMU}} \boldsymbol{w_{IMU}} \tag{3.76}$$

As there might be a constant bias in the IMU-measurements, we choose to keep the bias model, $\boldsymbol{f_3}$, as it is, we also choose to keep $\boldsymbol{f_1}$ as it is. Because of this change, we also need to update the process covariance matrix, $\boldsymbol{Q}$, as a measurement now has become a part of the model. We update the covariance terms relating to u, v, and w to the covariance of the IMU measurements, $\boldsymbol{y_{IMU}}$. The covariance propagation expression, $\boldsymbol{\Phi}$ should also be changed to fit the new model, however, this was not done.

3.4 Control Algorithm

In this project, two different control algorithms are used, *Pseudo-Derivative-Feedback*(PDF) and *Proportional-Integral-Derivative*(PID). PDF is used in surge and PID is used in sway, heave and yaw. In this section the PDF control algorithm is presented, we will not present the PID control algorithm due to its commonness and the assumption that the reader is familiar with PID-control.

3.4.1 Pseudo Derivative Feedback

A control law for a surface vessel based on *Pseudo Derivative Feedback*(PDF) was proposed in (Kjerstad et al., 2017). It is an adaptation of the linear time invariant concept proposed in (Phelan, 1971). In this section we will present a control law similar to the one presented in (Kjerstad et al., 2017). Further, we will implement the controller in a simulation to indicate the stability of the system. Last, we implement the control law in DOF on uDrone and test its performance.

Controller Fomulation

In (Phelan, 1971), the PDF controller is formulated for a linear time invariant system. Consider the system in (3.77)-(3.78) where $\ddot{x}, \dot{x}, x \in \mathbb{R}$ is the acceleration, velocity and position, respectively, $m, d \in \mathbb{R}_{>0}$ are known system parameters, and b is an unknown, constant bias. Full state feedback is assumed, such that both PID and PDF control laws can be formulated.

$$m\ddot{x} + a\dot{x} = u + b \tag{3.77}$$

$$\dot{b} = 0 \tag{3.78}$$

For the system (3.77)-(3.78), the PDF control law is formulated as in (3.79).

$$u_{PDF} = -k_p x - k_d \dot{x} - k_i \int_0^t (x - x_d)dt \tag{3.79}$$

In order to indicate stability, we look at the closed-loop transfer function, as was done in (Kjerstad et al., 2017).

$$s^2mx + sax = -k_px - sk_dx - \frac{k_ix}{s} + \frac{k_ix_d}{s} + b \tag{3.80}$$

$$x(s) = \frac{k_ix_d + bs}{(ms^3 + (k_d + a)s^2 + k_ps + k_i} \tag{3.81}$$

We see by 3.81 that k_p, k_i, and k_d can be tuned such that the poles of the transfer function lies in the left half plane.

Model Specific PDF

We move on to design a controller for our nonlinear system, heavily inspired by the formulation found in (Kjerstad et al., 2017). In order to design the controller, we start by looking at the model for our system without bias or disturbance.

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3.82}$$

$$\dot{\boldsymbol{\nu}} = \boldsymbol{M}^{-1}(\boldsymbol{\tau} - \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu}) \tag{3.83}$$

Based on the model, we propose the control law in (3.84)-(3.85), where $\boldsymbol{J}(\boldsymbol{\eta})^{-1} = \boldsymbol{J}(\boldsymbol{\eta})^\intercal$, $\boldsymbol{\xi}$ is the integration state, $\boldsymbol{\eta_d}$ is the desired position, $\boldsymbol{K_d}$ is the derivative gain matrix, $\boldsymbol{K_p}$ is the proportional gain matrix, and $\boldsymbol{K_i}$ is the integral gain matrix. In this controller design, we assume that the pose, $\boldsymbol{\eta}$, and velocities $\boldsymbol{\nu}$ are perfectly known.

$$\dot{\boldsymbol{\xi}} = \boldsymbol{K_i}(\boldsymbol{\eta} - \boldsymbol{\eta_d}) \tag{3.84}$$

$$\boldsymbol{\tau} = \boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{C}(\boldsymbol{\nu})\boldsymbol{\nu} - \boldsymbol{M}(\boldsymbol{K_d}\boldsymbol{\nu} + \boldsymbol{J}(\boldsymbol{\eta})^\intercal(\boldsymbol{K_p}\boldsymbol{\eta} + \boldsymbol{\xi})) \tag{3.85}$$

This renders our system

$$\dot{\boldsymbol{\xi}} = \boldsymbol{K_i}(\boldsymbol{\eta} - \boldsymbol{\eta_d}) \tag{3.86}$$

$$\dot{\boldsymbol{\eta}} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3.87}$$

$$\dot{\boldsymbol{\nu}} = -\boldsymbol{K_d}\boldsymbol{\nu} - \boldsymbol{J}(\boldsymbol{\eta})^\intercal(\boldsymbol{K_p}\boldsymbol{\eta} + \boldsymbol{\xi}) \tag{3.88}$$

In order to check if the closed loop system is stable, we propose the error states in (3.89)-(3.91).

$$\boldsymbol{z_1} = \boldsymbol{\eta} - \boldsymbol{\eta_d} \tag{3.89}$$

$$\boldsymbol{z_2} = \boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3.90}$$

$$\boldsymbol{z_3} = \boldsymbol{K_p}\boldsymbol{\eta} + \boldsymbol{\xi} \tag{3.91}$$

In order to find the closed loop error dynamics, the error states are differentiated. The controller is going to be used in setpoint regulation using steps without a reference filter, this renders $\frac{\partial \boldsymbol{\eta_d}}{\partial t} = \mathbf{0}$, thus rendering $\boldsymbol{\nu_d} = 0$. The closed loop error dynamics then become as in (3.92)-(3.94).

$$\dot{\boldsymbol{z}}_1 = \boldsymbol{z_2} \tag{3.92}$$

$$\dot{\boldsymbol{z}}_2 = \boldsymbol{J}(\boldsymbol{\eta})(\boldsymbol{S}(\boldsymbol{\nu}) - \boldsymbol{K_d})\boldsymbol{J}(\boldsymbol{\eta})^\mathsf{T}\boldsymbol{z_2} - \boldsymbol{z_3} \tag{3.93}$$

$$\dot{\boldsymbol{z}}_3 = \boldsymbol{K_i}\boldsymbol{z_1} + \boldsymbol{K_p}\boldsymbol{J}(\boldsymbol{\eta})^\mathsf{T}\boldsymbol{z_2} \tag{3.94}$$

Choosing $\boldsymbol{K_d} = \boldsymbol{S}(\boldsymbol{\nu}) + \boldsymbol{J}(\boldsymbol{\eta})^\mathsf{T}\boldsymbol{K_d^*}\boldsymbol{J}(\boldsymbol{\eta})$ and $\boldsymbol{K_p} = \boldsymbol{K_p^*}\boldsymbol{J}(\boldsymbol{\eta})$ renders the system linear. Defining $\boldsymbol{z} := \begin{bmatrix} \boldsymbol{z_1} & \boldsymbol{z_2} & \boldsymbol{z_3} \end{bmatrix}^\mathsf{T}$, we can define the state-space as in (3.95).

$$\dot{\boldsymbol{z}} = \underbrace{\begin{bmatrix} \mathbf{0} & \boldsymbol{I} & \mathbf{0} \\ \mathbf{0} & -\boldsymbol{K_d^*} & -\boldsymbol{I} \\ \boldsymbol{K_i} & \boldsymbol{K_p^*} & \mathbf{0} \end{bmatrix}}_{=\boldsymbol{A}} \boldsymbol{z} \tag{3.95}$$

This renders the equilibrium, $\boldsymbol{z} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix}^\mathsf{T}$, globally exponentially stable if $\boldsymbol{K_d^*}$, $\boldsymbol{K_i}$ and $\boldsymbol{K_p^*}$ are designed such that $\boldsymbol{A}$ is Hurwitz. We continue to constrain the convergence velocity.

Convergence Velocity Constraint

In (Kjerstad et al., 2017), the convergence velocity is constrained by saturating the error in the integration state, $\boldsymbol{\xi}$, as presented in (3.96) and (3.97).

$$\dot{\boldsymbol{\xi}} = \boldsymbol{sat}(\boldsymbol{K_i}(\boldsymbol{\eta} - \boldsymbol{\eta_d})) \tag{3.96}$$

$$sat(x) = \begin{cases} x & if|x| \leq x_{max} \\ sign(x)x_{max} & else \end{cases} \tag{3.97}$$

In (3.97), $x \in \mathbb{R}$, $x_{max} \in \mathbb{R}_{>0}$, and $sign(\cdot)$ is the sign operator. From the saturation, it follows that we can limit the rate of convergence. This stems from the intuition of the integration state has to overcome the proportional state. The choice of x_{max} will have an impact on the maximum convergence rate, high x_{max} will lead to fast convergence rate and vice versa.

Tuning

We follow the procedure of tuning presented in (Kjerstad et al., 2017), and augment it to include heave motion as well. For each degree of freedom, we can calculate k_p, k_i and k_d by (3.98)-(3.100), where k_n are the diagonal entries in the matrix $\boldsymbol{K_n}$ for $n = p, i, d$.

$$k_d = 2\zeta\omega_0 + \alpha \tag{3.98}$$

$$k_p = \omega_0^2 + 2\zeta\omega_0\alpha \tag{3.99}$$

$$k_i = \alpha\omega_0^2 \tag{3.100}$$

In (3.98)-(3.100), ζ is the damping coefficient, ω_0 is the natural frequency of the system, and α acts as an inverse first order lowpass filter time constant as stated in (Kjerstad et al., 2017).

Verification by Simulation

The controller is verified by simulation in Simulink. In this section we will present the parameters used and the simulation results. A simulation model of the uDrone was made in Simulink using the mathematical model presented in section 3.1. Furthermore, the control law was implemented and tuned with the tuning parameters presented next. For simplicity and proof of concept, ζ, ω_0, and α were chosen to be equal for all degrees of freedom. We present the tuning parameters in table 3.6.

Table 3.6: Tuning parameters for simulation of PDF control on the ROV uDrone.

ζ	ω_0	α	x_{max}
1.5	0.7	2	0.3

We continue to present the simulation results for the north position change and the yaw motion. In this simulation, we change $\boldsymbol{\eta_d}$ from $\boldsymbol{\eta_d} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$ to $\boldsymbol{\eta_d} = \begin{bmatrix} 0.5 & 0 & 0 & 0.2 \end{bmatrix}^\mathsf{T}$ at time $t = 1sec$. The results from the simulation is presented in figures 3.6 and 3.7.

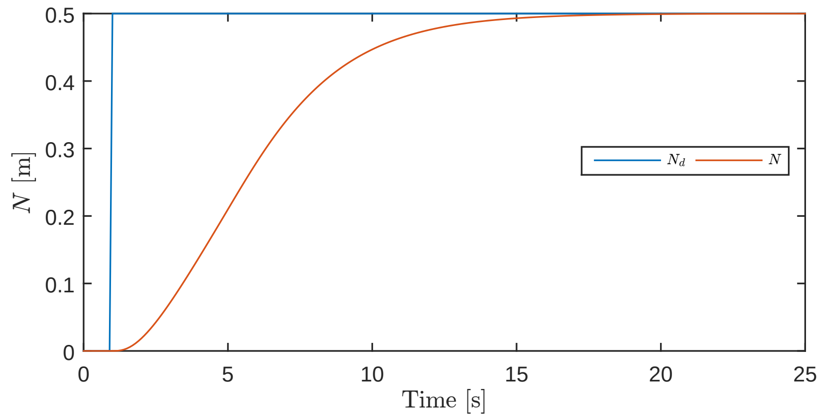


Figure 3.6: Time domain plot of desired north position, N_d, and actual north position, N, in simulation of step response.

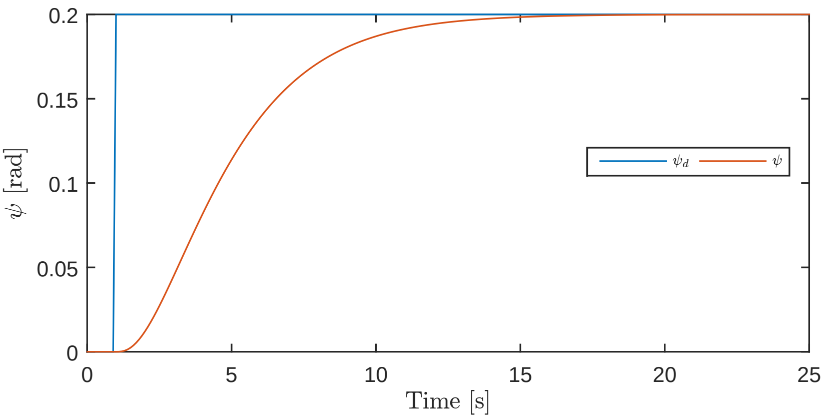


Figure 3.7: Time domain plot of desired heading, ψ_d, and actual heading, ψ, in simulation of step response.

We move on to test the robustness of the controller by adding band-limited white noise in the process model and run the simulations again. We plot the results in figures 3.8 and 3.9.

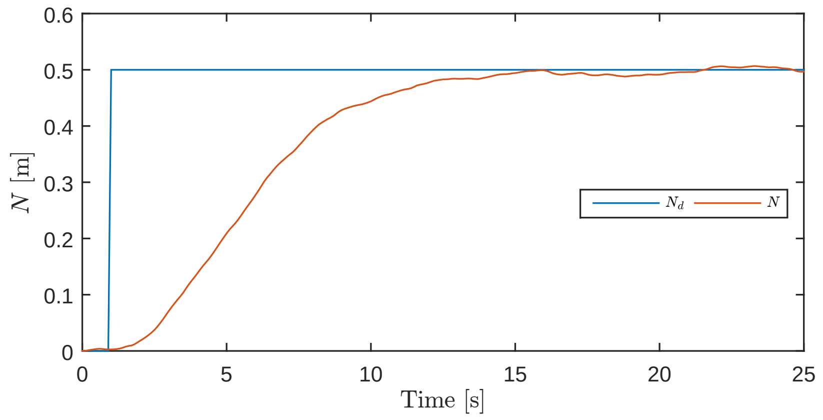


Figure 3.8: Time domain plot of desired north position, N_d, and actual north position, N, in simulation of step response. White noise is added in the process model.

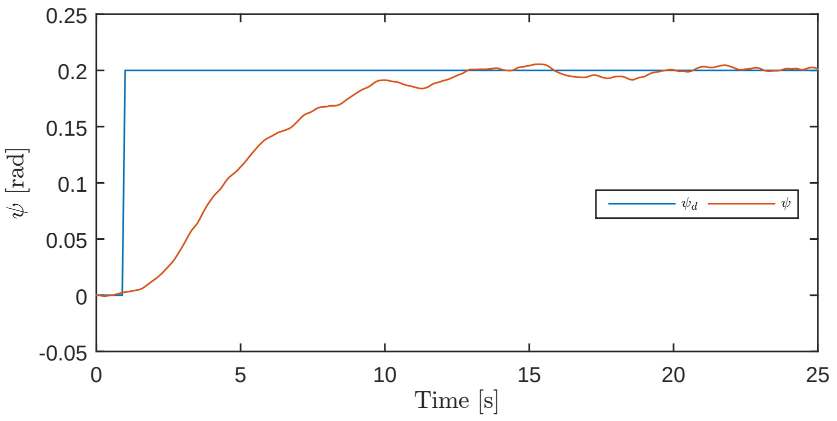


Figure 3.9: Time domain plot of desired heading, ψ_d, and actual heading, ψ, in simulation of step response. White noise is added in the process model.

We see that in both cases, with and without the process noise, the system takes about 15 seconds to converge to the desired positions. There were two main motivations for doing a simulation test; to confirm the control law, and to help tune the controller used in the real world implementation.

Chapter 4

Experiments

In this chapter results form experiments will be presented. All experiments and tests were carried out in the MCLab basin with the uDrone. MCLab is located at Tyholt and is a laboratory for testing and developing surface and underwater vehicles. The lab consists of a basin, sensor systems for above and under water positioning (Handbook, 2016). The experiments that were carried out are as follows:

- Laser range measurement validation.
- Stationkeeping capability in calm water.
- Stationkeeping capability when perturbed.
- Setpoint regulation capabilities in surge.
- Test of scaling factor found by range comparison.
- Test of scaling factor found by depth comparison.
- Comparison of the model based and kinematic EKFs.

First, the experimental setup is presented, then the results from the above mentioned experiments are presented along with a more comprehensive description of the experiment.

4.1 Experimental Setup

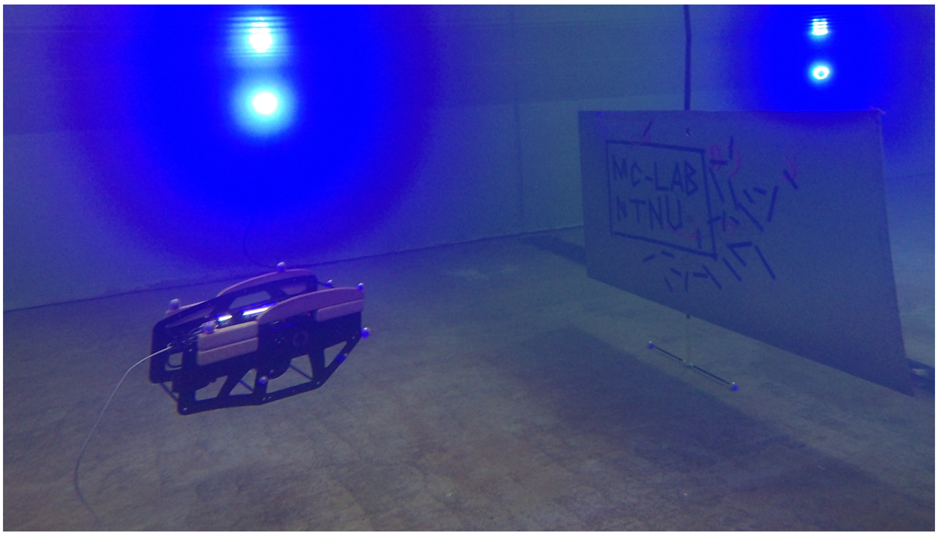


Figure 4.1: ROV uDrone in the Marine Cybernetics Laboratory basin during full DP using ORB-SLAM as position reference.

In order to test the sationkeeping capabilities of the system, we need an absolute measurement of the robot pose and compare it with the estimated pose from the implemented SLAM-algorithm and the EKF. To achieve this, a metal plate with high contrast features was lowered into the basin of the *Marine Cybernetics Laboratory*(MCLab) within the range of the Qualisys positioning system. Qualisys is a reliable and accurate positioning system, using 6 cameras emitting low wavelength light and detecting its reflection on reflectors visible on uDrone in figure 4.1 (Handbook, 2016). The plate served as the main source of features observed by ORB-SLAM. In order to know the positions of the plate and the drone, both were fitted with reflective balls visible to the positioning system.

4.2 Range Measurement

In order to verify the range measurement we need to compare the measurements made by the range estimator described in section 3.2.1. This is done by comparing the measurement of z_o to the distance from the Raspberry Pi camera to the front of the plate as measured by Qualisys, z_Q. In this section we will present the experiment done in order to verify the range estimator and its results.

4.2.1 Setup

The experiment was set up as depicted in figure 4.1, with both the plate and the drone visible to the positioning system. First, a measurement was made finding the offset, $\tilde{z}_o$, between the body-frames of the uDrone and the plate made in Qualisys when the range, z_o, was zero. The validity of the measurement was based on the following assumptions.

- When the front of the uDrone is in contact with the plate, and the optical axis is orthogonal to the plate surface, the range is zero.

- When the range is measured, the optical axis is aligned with the Qualisys x-axis.

- When the range is measured the plate surface is parallel to the Qualisys yz-plane.

Thus, we assume that the offset is only manifested in the direction of the optical axis, which is assumed parallel to the Qualisys x-axis. The offset was found to be $\tilde{z}_o = 0.30[\text{m}]$. We then calculated the range as measured by Qualisys, z_Q, by equation 4.1, where x_p and x_u is the x-positions of the plate and the uDrone, in the Qualisys reference frame, respectively.

$$z_Q = x_p - x_d - \tilde{z} \tag{4.1}$$

We will further present the results from the measurements made during stationkeping in calm water, comparing z_Q and z_o.

4.2.2 Results

The results plotted in figure 4.2 are from stationkeeping in calm water, where the distance from the optical center to a flat plate is measured underwater. We see that the range estimation, z_o, has some outliers and is somewhat noisy otherwise. Disregarding the outliers and noise, it is visible by figure 4.2 that the range measurement is quite accurate. It may become less accurate at smaller distances due to refractive distortion, this was not tested due to the plate creating a blind spot for Qualisys where uDrone cannot be detected. A counter argument would be that since the Raspberry Pi camera has a more narrow field of view, it would not be as affected by refractive distortion as the usb-camera.

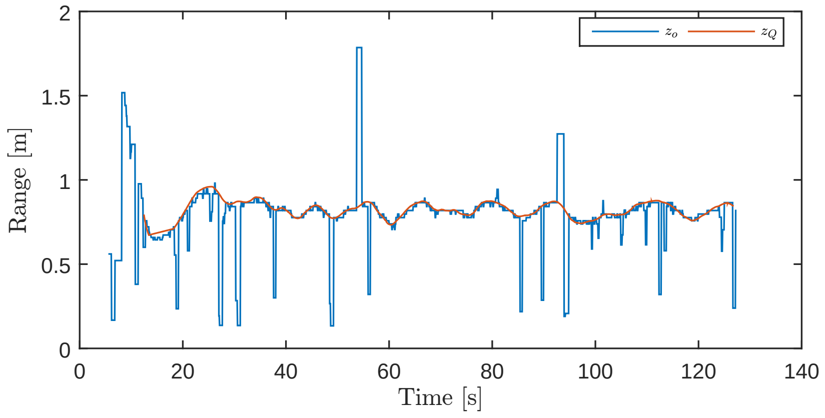


Figure 4.2: Comparison of the range measured by lasers, z_o, and the range measured by Qualisys, z_Q. Both measurements are made from the optical axis to the surface of a flat plate.

Based on the results presented in figure 4.2, we deem the laser range measurements to be accurate but in need of better outlier rejection and filtering.

4.3 Stationkeeping

In this section we will present the stationkeeping experiment. This experiment was set up the same way as the previous one. The desired position is set to $\boldsymbol{\eta_d} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^\mathsf{T}$, which corresponds to the initialization point of ORB-SLAM. The first experiment was done in calm water, with minimal external perturbation. In the second experiment, the drone was pushed away from its position with a rod. Currents and waves were not introduced due to safety for the drone.The PDF and PID controllers were tuned according to table 4.1. The PID controller in z-direction is tuned quite aggressively in to compensate for the buoyant force on uDrone. PDF-gains were found using $\alpha = 1$, $\zeta = 1.5$ and $\omega_0 = 0.7[rad/s]$.

Table 4.1: Tuning parameters for PDF and PID control of uDrone.

DOF	Controller	k_p	k_i	k_d	x_{max}
x	PDF	3.14	0.49	2.59	1
y	PID	0.6	0.002	0.04	-
z	PID	10	0.5	0.4	-
ψ	PID	0.3	0.001	0.02	-

For state estimation, the kinematic EKF was used. It provided position and velocity estimates to the controller.

4.3.1 Results

In this section the results from the stationkeeping tests are presented. We start by presenting the unperturbed system, and continue to present the results from when the system was perturbed.

Calm Water

We recorded the pose with Qualisys and ORB-SLAM and estimated the states using the EKF. These three measurements are now compared for the four degrees of freedom in our model. In figures 4.3, 4.4, 4.5, and 4.6, the position is plotted for the two measurements and the EKF estimate for x-, y-, z-, and ψ-position respectively.

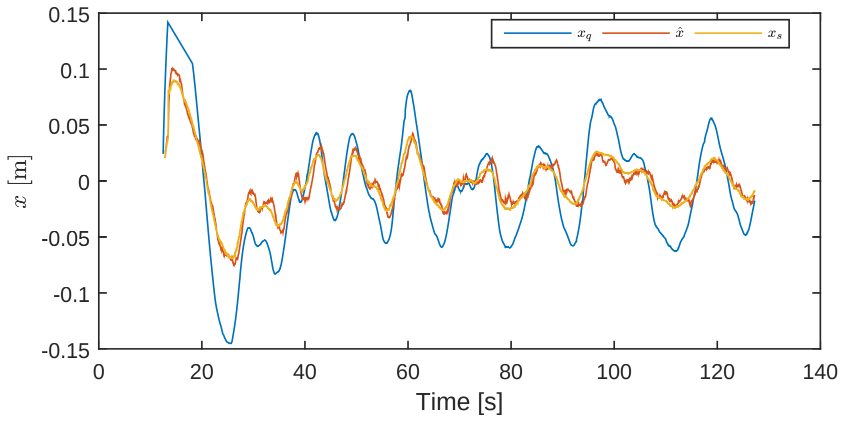


Figure 4.3: x-position as measured by Qualisys and ORB-SLAM, and the estimated x-position from the EKF.

In figure 4.3, we see the x-position of uDrone oscillating around its setpoint, 0. The oscillations has a period of about $T = 7s$ to $T = 11s$, this corresponds well with ω_0 used in tuning the PDF controller, which gives $T_o = \frac{2\pi}{\omega_0} = 8.976s$. In addition, we see that the Qualisys measurements has a larger amplitude than the position estimates from ORB-SLAM and the EKF, this indicates that the scaling factor is too small. Last, a time delay is visible as the peaks of x_s and $\hat{x}$ is shifted to the right compared to x_q. This time delay might be responsible for the oscillatory behavior of the system.

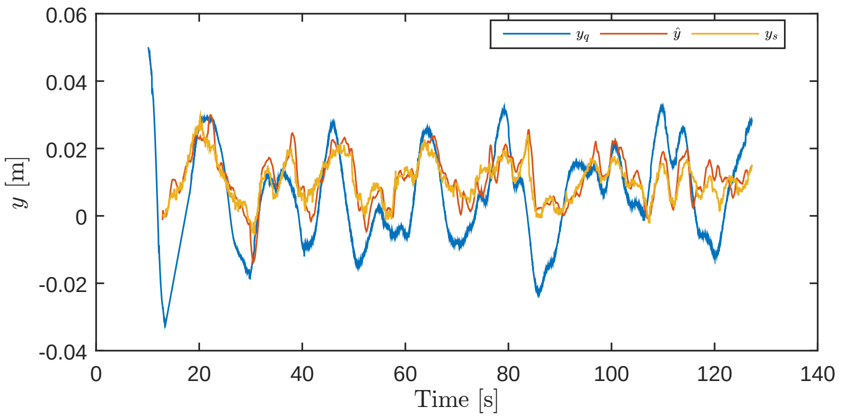


Figure 4.4: y-position as measured by Qualisys and ORB-SLAM, and the estimated y-position from the EKF.

Figure 4.4 shows the y-position of uDrone during stationkeeping, from y_q it is visible that the drone held its position within $\pm 3cm$ in contrast to x_q, which showed larger fluctuations. This might be caused by a less aggressive control algorithm for sway motion, leading to a smaller controller frequency, thus reducing the effect of time-delay.

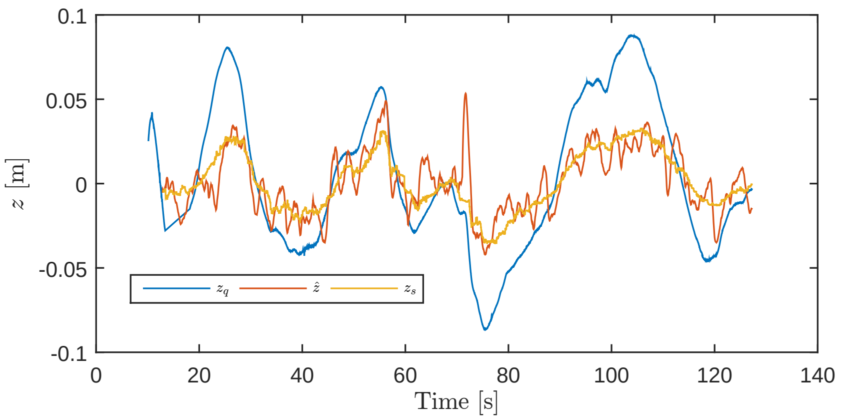


Figure 4.5: z-position as measured by Qualisys and ORB-SLAM, and the estimated z-position from the EKF.

By figure 4.5, we see that the z_q-position was held within $\pm 10cm$, and again, that the scaling factor was too small. Also, the estimate, $\hat{z}$, behaved more erratically than the other estimated states. Error sources for this are:

- Wrong *roll*- and *pitch*-estimates from the IMU, such that the acceleration measurement in z-direction was influenced by gravity

- Wrongful implementation of the EKF.

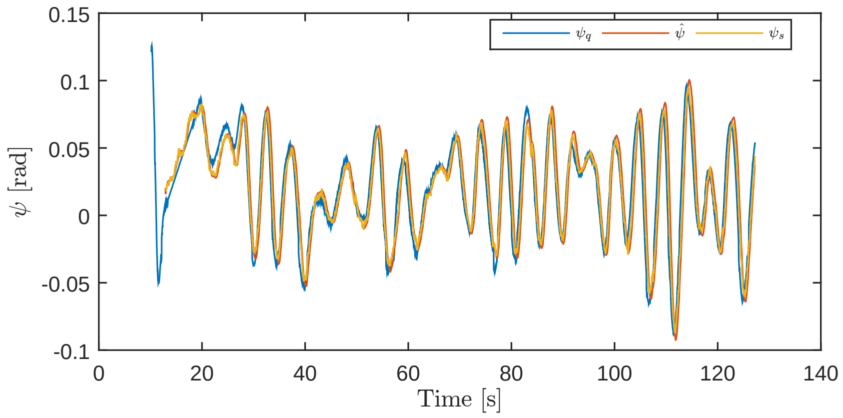


Figure 4.6: ψ-position as measured by Qualisys and ORB-SLAM, and the estimated ψ-position from the EKF.

The last figure from this experiment, figure 4.6, shows the heading, ψ, as estimated by ORB-SLAM and the EKF along with the measured heading from Qualisys. It shows a near perfect correlation of the estimates and measurements along with the time-delay between Qualisys, ORB-SLAM and the EKF. Oscillations occur in the range $\pm 0.1 rad$, and are more likely caused by poor controller tuning, wrong control law, asymmetrical thrust allocation or time-delay.

Perturbed System

As mentioned above, the system was perturbed during stationkeeping by using a rod to push the drone off its position. Experiments using waves or current were not done due to constraints in the experimental setup. If waves were to be introduced in the basin, the metal plate would oscillate and might damage the drone. If current were to be introduced in the basin, the plate would be dragged off to one end of the basin, due to it being mounted on a bridge on rails. Thus, experiments with waves and current were replaced with manual perturbation.

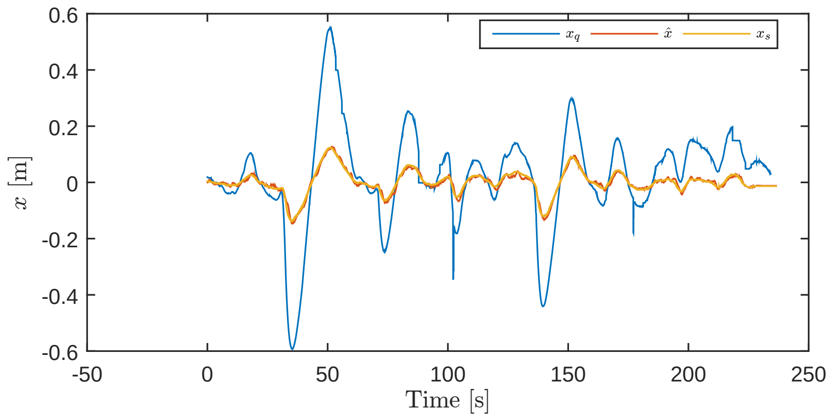


Figure 4.7: x-position as measured by Qualisys and ORB-SLAM, and the estimated position from the EKF, $\hat{x}$. The drone was pushed in the negative x-direction three times, at Time $= [30s, 72s, 140s]$

Figure 4.7 shows the x-positions of the drone as estimated by ORB-SLAM, x_s, the EKF, $\hat{x}$, and as measured by Qualisys, x_q. The drone was pushed in the negative x-direction three times, at Time $= [30s, 72s, 140s]$. The system responds to these perturbations similarly, being bushed back, overshooting the setpoint by almost the same distance as it was pushed back and settling on the oscillatory motion visible in figure 4.3.

4.4 Setpoint Regulation

In this section the results from setpoint regulation tests are presented. As the PDF controller is intended for setpoint regulation, we present results from changing sepoints along the x-axis. Controller tuning remains the same as in the previous experiments, as presented in table 4.1. The desired x-positions, x_d, used in this test are presented in 4.2. All other desired postions were kept at 0, such that $\boldsymbol{\eta_d} = [\ x_d \quad 0 \quad 0 \quad 0\ ]^\intercal$.

Table 4.2: Desired x-positions of uDrone in relation to the initialization point of ORB-SLAM.

Time	0	$18.7s$	$68.7s$	$98.8s$	$128s$
x_d	$0.0m$	$-0.5m$	$0.0m$	$-1.0m$	$0.0m$

We continue to present the results from this experiment.

4.4.1 Results

Results from the setpoint regulation test are plotted in figure 4.8. Again, the oscillatory behavior from the previous tests is visible and the scaling factor is also too small. Furthermore, we see that the plate creates a blind spot for Qualisys, as it drops out at $x_q >\approx -0.25m$, this is due to the plate blocking the view of the Qualisys cameras.

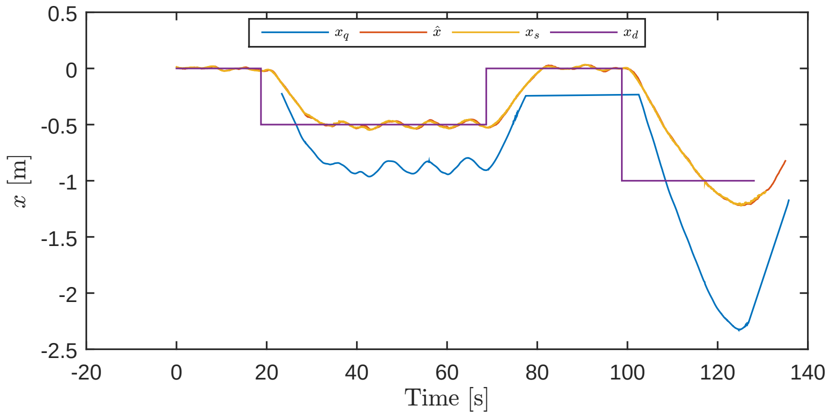


Figure 4.8: x-position as measured by Qualisys and ORB-SLAM, and the estimated x-position from the EKF along with the desired x-position.

Disregarding the error in scaling factor and oscillatory motion, we analyze the step response when x_d is changed from $-0.5m$ to $0.0m$.

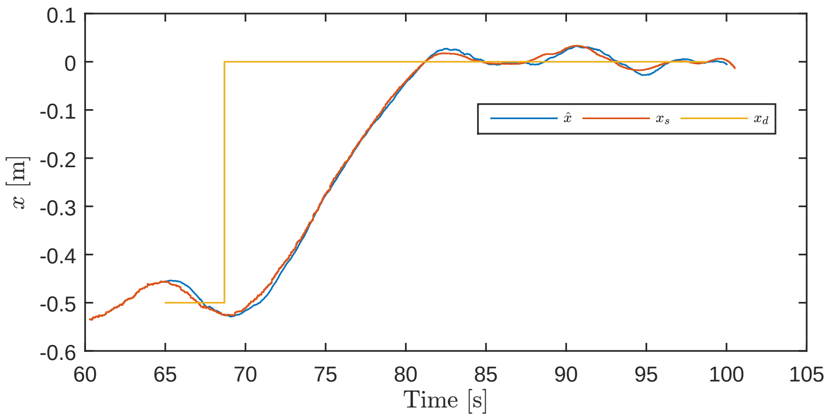


Figure 4.9: x-position as measured by Qualisys and ORB-SLAM, and the estimated x-position from the EKF along with the desired x-position.

From table 4.2, we see that the step in desired position is introduced at time $t = 68.7s$ and that the state $\hat{x}$ reaches the desired position at time $t \approx 81s$, this indicates that $\hat{x}$(and x_s) converges to x_d in $\approx 12s$.

4.5 Scaling Factor Comparison

The two scaling factor estimates were logged during the tests, in this section we will present the findings. In order to get an estimate of the scaling factor from depth comparison, the desired pose was set to $\boldsymbol{\eta_d} = \begin{bmatrix} 0 & 0 & 0.1 & 0 \end{bmatrix}^\intercal$. The true scale was estimated in post processing by comparing the positions in x-direction measured by ORB-SLAM and Qualisys. The experimental setup used in this section is the same as depicted in figure 4.1, where the drone is performing stationkeeping in front of a flat plate. First, we present the input data to the depth scale estimation, second, we present the laser range scaling factor along with the depth scaling factor and the true scaling factor.

4.5.1 Results

In this section, results and comparison of the scaling factors are presented. In figure 4.10, the inputs to depth scaling factor estimator are presented. These inputs are h, which is the offset from the initialization point to the current position along the world D-direction as measured by the pressure sensor, and z_s which is the ORB-SLAM z-position of the drone. As mentioned in section 3.2.1, this corresponds to the offset in z-direction in the ORB-SLAM frame from the initialization point.

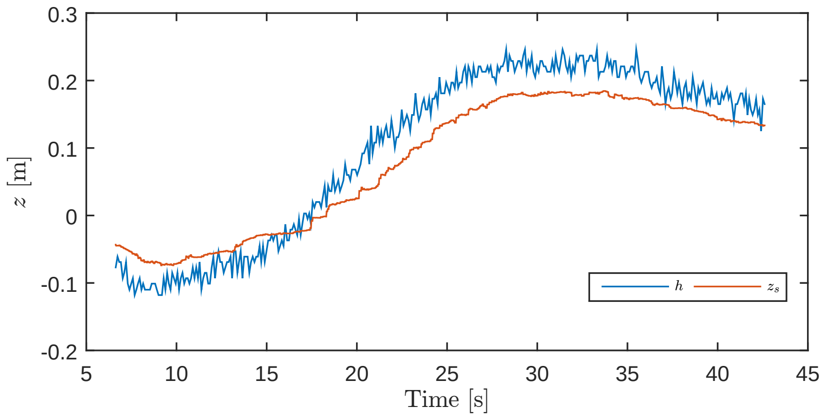


Figure 4.10: z-offsets from the initialization point as measured by ORB-SLAM, z_s, and pressure sensor, h.

We see by figure 4.10 that h has a larger amplitude than z_s, this should indicate that the depth scaling factor ought to be larger than one. Also, there is a zero-crossing around $Time = 18s$ which may cause problems for the scale estimator. Next we will present the inputs to the laser range scaling factor estimator for the same experiment.

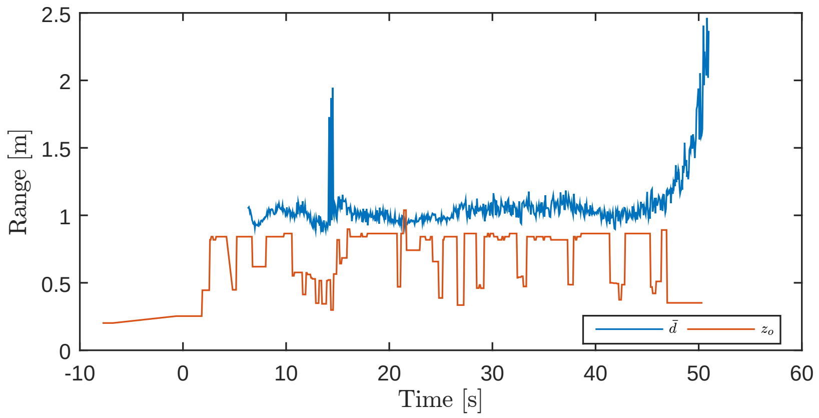


Figure 4.11: Ranges from the camera to a flat plate as measured by ORB-SLAM, $\bar{d}$, and laser range measurement, z_o.

Figure 4.11 shows the distance from the Raspberry Pi camera to the plate as estimated by laser range measurements, z_o, and the distance from the usb-camera estimated by

ORB-SLAM, $\bar{d}$. In this case, the ORB-SLAM range has an overall larger amplitude than the laser range measurements. Because of this, it is expected that the estimated scaling factor should be between one and zero. Next, we present the scaling factors as found by laser range and depth along with the estimated true scale.

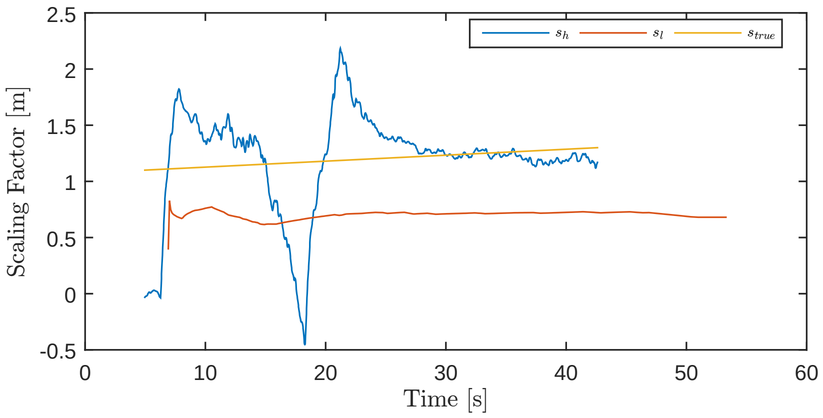


Figure 4.12: Scaling factors as found by laser range measurement, s_l, depth estimation, s_h, and in post processing, s_{true}.

In figure 4.12, the laser range scaling factor, s_l, being quite steady and having a near constant offset from the true scale. This offset is found to be approximately factor 1.8 of the true scaling factor. If the figures in the above experiments are examined, we see that the laser range scaling factor is too small. This indicates that something is wrong in the assumptions. We revisit the assumptions in section 3.2.4. Furthermore, figure 4.12 shows that the depth measurement scaling factor, s_h, is far more erratic than s_l. It also dips below zero during the zero-crossing, before returning to near s_{true}. This indicates that a more robust filtering and zero-handling approach is needed. It is also worth noting that in this experiment, the constraint on the sign of the depth offsets were not enforced. This is due to implementation errors in implementing the sign comparison.

4.6 EKF Comparison

Comparing the two EKFs ability to estimate position and velocity will give an indication of which one has the better performance. The comparison is done using as equal values for the $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices, in addition, the covariance propagation function $\boldsymbol{\Phi}$ is kept equal in both cases. First, the performance of the model based EKF is tested, second the kinematic EKF. In this experiment, the $\hat{z}$- and $\hat{w}$-estimates from both filters are compared to the z-estimate and w from ORB-SLAM, where w is found in post processing using a *Finite Impulse-Response*(FIR)-filter, presented in (Sørensen, 2012), on the vertical velocity found by Euler differentiation. Utilizing the FIR-filter the results from the differentiation was also low pass filtered in order to remove high amplitude noise. The reason for choosing the z and heave directions is that here we have an unmodeled bias, the buoyancy of uDrone, giving the model based EKF a disadvantage. In z-direction, the kinematic EKF will aslo be at a disatvantage, due to imperfect cancellation of gravitational acceleration. Thus, this will also be a test for the bias estimator.

4.6.1 Results

In this section, we present the z-positions as estimated by ORB-SLAM and the EKF, along with the positions from Qualisys from two different experiments, using the model based EKF in the first experiment, and the kinematic EKF in the second experiment. $\hat{x}$

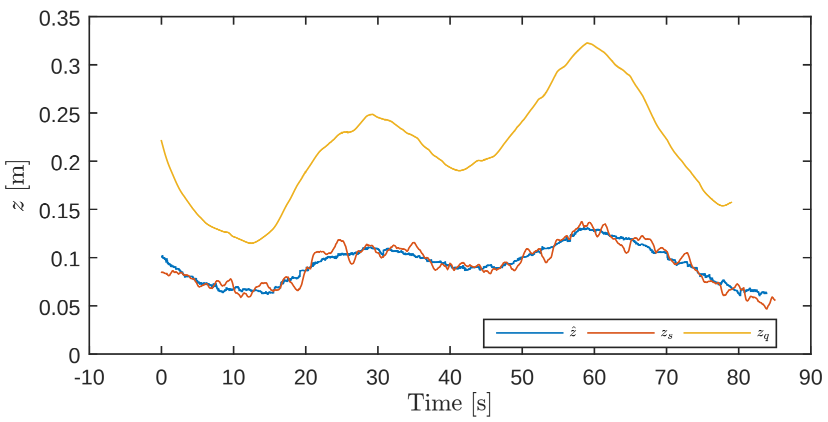


Figure 4.13: z-positions as estimated by scaled ORB SLAM, z_s, the kinematic EKF, $\hat{z}$, and measured by Qualisys, z_q.

Looking at $\hat{z}$ in figure 4.13, it fluctuates around z_s and behaves erratically indicating that the IMU-measurements are not filtered properly, and their covariance is underestimated. As it fluctuates, $\hat{z}$ behaves like it is bound to a the values around z_s, not diverging from it. We move on to the model based EKF.

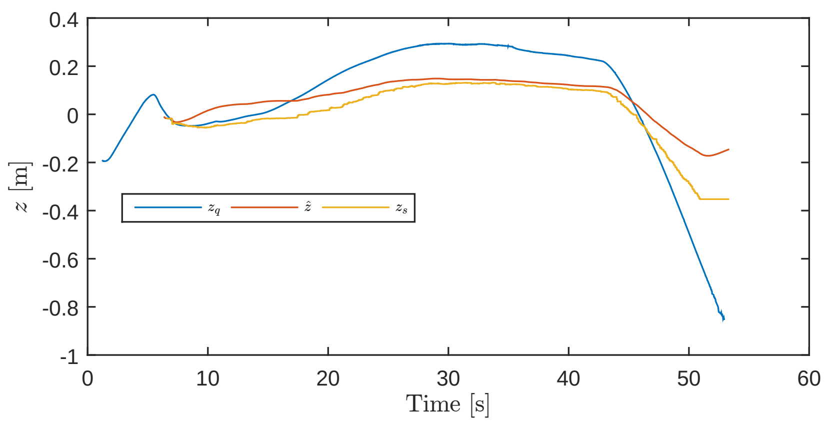


Figure 4.14: z-positions as estimated by scaled ORB SLAM, z_s, the model based EKF, $\hat{z}$, and measured by Qualisys, z_q.

In figure 4.14, $\hat{z}$ behaves much less erratically, and more smoothly than in figure 4.13. An initial divergence from z_s is also visible. This is most likely to the unmodeled buoyant force of the drone, acting in the negative z-direction, causing the controller to command a force in the positive direction, this force, as part of the model, will then push the estimate above the true value. It is also visible that $\hat{z}$ converges towards z_s after the initial divergence, before the position diverges again along with motion in the negative z-direction(towards the surface). The convergence before time $t \approx 45s$ is caused by the estimated bias. Next, the bias estimate for the same experiment is presented in figure 4.15.

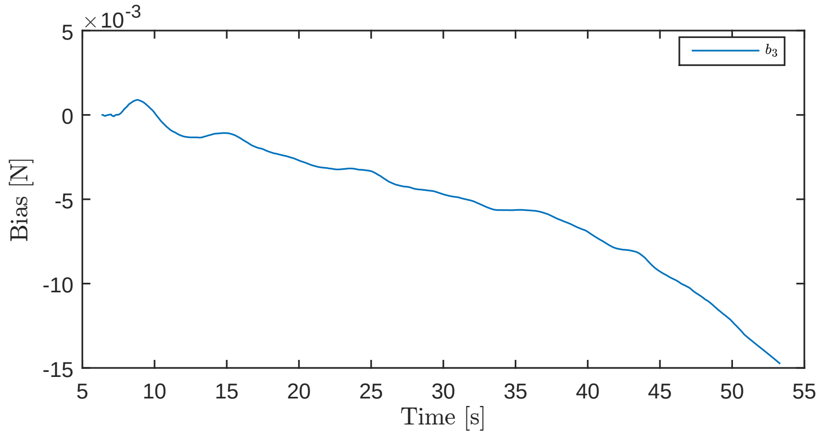


Figure 4.15: The bias of the model based EKF in z-direction, b_3, during the test presented in figure 4.14. Bias absolute value increases over time in order to compensate for unmodeled forces.

In figure 4.15, the bias for $\hat{z}$, b_3, is plotted. As $\hat{z}$ lies above z_s in most of figure 4.14, we would expect b_3 to be decreasing for the duration, which it does. We move on to examine the velocity estimates from the EKFs.

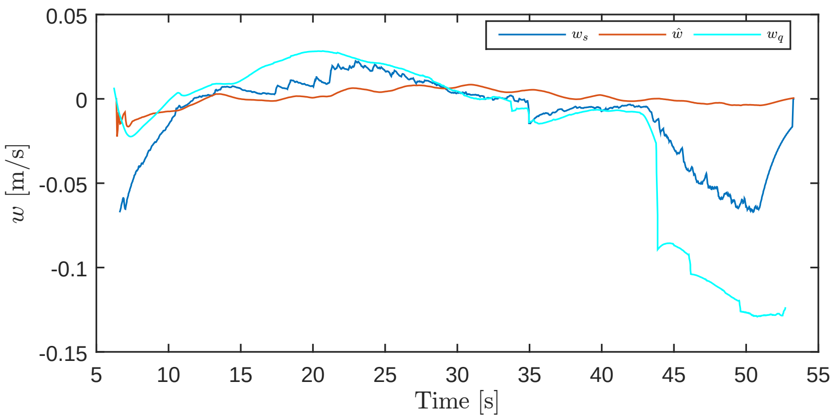


Figure 4.16: w-velocities as estimated by scaled ORB SLAM, w_s, the model based EKF, $\hat{w}$, and measured by Qualisys, w_q.

By figure 4.16, we see that the w-velocity estimate of the EKF is too slow compared to the filtered Euler differentiated z_s.

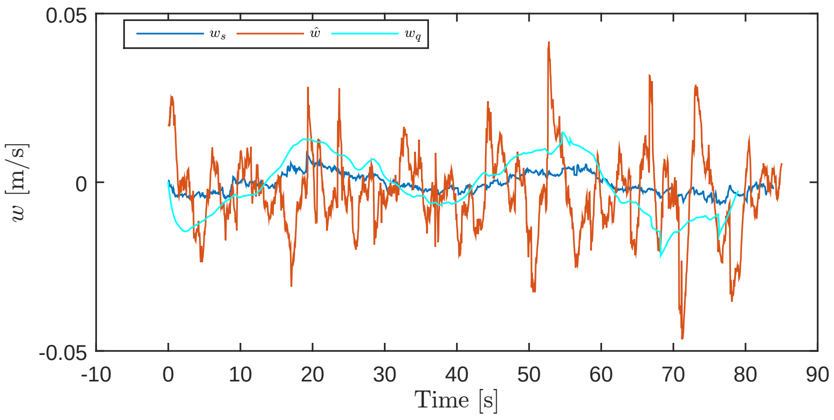


Figure 4.17: w-velocities as estimated by scaled ORB SLAM, w_s, the kinematic EKF, $\hat{w}$, and measured by Qualisys, w_q.

The above figures show that that neither the model based nor the kinematic EKF is any good at estimating the velocity in heave. The model based EKF is underestimating the velocity, and the kinematic EKF velocity estimate is erratic. The white-noise like behavior of the kinematic EKF is most likely caused by noise from the IMU. The poor performance can be attributed to poor tuning, wrongful implementation or a large difference between the model and the true behavior of the drone.

4.7 Comments on the Results

It is worth noting that the results presented in this chapter exhibit survivorship bias in regards to successful initializations and runs. In about 50% of the cases, the difference in scale between ORB-SLAM and the real world was too great for the control algorithms to be effective. As the initial guess of the scaling factor is 1, the convergence to true scale needs to be faster for it to be more robust. In some cases, the scale also drifted rapidly, oftentimes in conjunction with large movements in surge. This drift might be caused by poor camera calibration together with light scattering in the water. Scattering makes long time tracking of the same points a challenge when the drone moves through water, since the points in space change their appearance as a function of distance. This calls for a more robust feature descriptor, more suitable for the underwater environment. For more results, video and pictures from the experiments, see attachment.

Chapter 5

Discussion

In this chapter we will discuss the main topics and results found in this thesis, starting with the laser range scaling factor, continuing to the depth scaling factor, the fitness of ORB-SLAM in the underwater environment, the performance of the controllers and estimators, finishing with a discussion about the thesis as a whole.

5.1 Scaling factors

5.1.1 Laser Range

In the results chapter, especially figure 4.7, is is visible that the scaling factor is too small compared to what it should be. This could have several error sources, comparing the range estimates from figures 4.2 and 4.11, we see that the latter has more outliers, and they last for longer periods. We can also see that almost all of the outliers in figure 4.11 are below the true range, driving the average of the range estimate down. This will in turn drive the laser range scaling factor down. However, faulty laser range measurements are not the only error source. In order to examine the main suspect for the underestimation of the scaling factor we need to revisit the assumption made in section 3.2.4; that the points in the ORB-SLAM map have a Gaussian distribution around the range from the usb-camera to the surface in front of it if that surface is flat and orthogonal to the optical axis. Points initiated with sufficiently low parallax can be initiated at any depth between 0 and ∞, thus making it likely that the points are initiated too far away from the camera in the map. This in turn, drives the ORB-SLAM range up, decreasing the range laser scaling factor.

5.1.2 Depth Range

As it is presented in this thesis, the scaling factor from depth offset comparison is more a suggestion of how scale can be inferred, rather than a fully functioning algorithm. In order to compare the offsets from the initialization point accurately, both filtering and an offset is needed. We see by figure 4.10, that the depth estimate is quite noisy, thus, it will have several zero-crossings close to the initialization depth. In order to be less erratic than in figure 4.10, the depth scale estimate is in need of more active filtering and a larger rejection zone, or a new approach altogether.

5.2 Underwater ORB-SLAM

ORB-SLAM has proved satisfactory performance underwater, albeit poorer performance than above water. The reason for performance reduction could be a combination of a more complex image distortion and the poor light penetration in water compared to air. As ORB-SLAM needs features to detect in order to function, we can predict that the algorithm will have better performance in a real ocean environment compared to the relative uniformness of the MCLab basin.

5.3 Controller performance

Looking at the results presented in sections 4.3 and 4.4, we see that all DOFs exhibit some oscillatory motion, and the controllers could use more tuning. Another error source for poor controller performance is that both the PDF and PID controllers are dependent on velocity estimates. From figures 4.17 and 4.16, we see that the velocity estimates are inaccurate. Furthermore, it is known that there is a time-delay in the video pipeline, causing ORB-SLAM to having a delay. There will also be some inherent delay in the thrust allocation, causing the overall process delay to increase. If this delay is large enough it could cause sufficient phase shift, causing oscillation.

5.4 Estimator Performance

As discussed in section 4.6 and above, the estimation capabilities of bot EKFs are quite poor. Both filters are in need of proper tuning, and at their current state they are not properly estimating the velocity. However, they have served the purpose of a great learning experience in implementing EKFs in discrete time, and insight has been gained.

5.5 Overall Discussion

Looking at the results presented in the previous chapter, we can argue that ORB-SLAM can work for dynamic positioning of a low-cost ROV. We can argue that it is not necessary for the ROV to go 1 meter when given the command to go one meter forward. It is however necessary that the user can direct the ROV to the position he or she desires. In turn, this means that the scaling factor does not need to be perfect, although in its current state, it is underestimating the scale.

Chapter 6

Conclusion

6.1 Conclusion

In this thesis we have shown a proof of concept for using camera-based slam to perform dynamic positioning of a low-cost ROV. We have highlighted challenges, methods and results relevant for implementation of scale-aware ORB-SLAM in the underwater environment. In summation, it is possible for a low-cost ROV with sufficient computing power to successfully perform stationkeeping and dynamic positioning within a small area without any external position reference other than ORB-SLAM.

It is not essential for the user experience to be able to control the position accurately in the sense of scale, however scale needs to be more accurately estimated for the sake of the estimators and control algorithms. As mentioned in section 4.7, the rate of successful initializations is at it current sate 50%. In conclusion, the work presented in this this has served as a proof of concept for camera-based SLAM on a low-cost ROVs. We have explored ORB-SLAM and it applicability for underwater operations, computer vision for laser range measurement, state estimation and feedback control based on position feedback from ORB-SLAM. There is still work to be done before this is a viable solution for any commercial or industrial application, thus we present our suggestions for further work.

6.2 Further Work

In this section we propose suggestions on further work as suggested by the author. The goal of this section is guiding the work in the direction of a robust implementation of the approach presented in this thesis.

- Revisit and improve the ORB-SLAM range estimate $\bar{d}$, in order to reject possible outliers.
- Make an ORB-vocabulary based on underwater images, preferably from the same ROV it is to be implemented on.
- Develop a smarter and more robust depth comparison scale estimator.
- Develop a more robust outlier rejection for laser range measurements.
- Develop a dynamic thruster model for use in process models.
- Develop a method for accurate time delay estimation, and a predictor to compensate for the delay.

Bibliography

Blueye, 2017. Blueye robotics home page. `https://www.blueye.no/`, accessed: June 11th, 2017.

Bonarini, A., Burgard, W., Fontana, G., Matteucci, M., Sorrenti, D. G., Tardos, J. D., 2006. Rawseeds: Robotics advancement through web-publishing of sensorial and elaborated extensive data sets. In: In proceedings of IROS. Vol. 6.

Davison, A. J., Ried, I. D., Molton, N. D., Stasse, O., 6 2007. Monoslam: Real-time single camera slam. IEEE Transactions on pattern analysis and machine intelligence 29 (6).

Faltinsen, O., 1993. Sea loads on ships and offshore structures. Vol. 1. Cambridge university press.

Fossen, T. I., 2011. Handbook of Marine Craft Hydrodynamics and Motion Control, 1st Edition. Wiley, Trondheim, Norway.

Handbook, M., 2016. Ntnu marine cybernetics laboratory handbook. NTNU Trondheim Norwegian University of Science and Technology Department of Marine Technology, available at `https://github.com/NTNU-MCS/MC_Lab_Handbook`.

Hartley, R., Zisserman, A., 2003. Multiple View Geometry in Computer Vision. Cambridge University Press, Cambridge, U.K.

Heikkilä, J., Silvén, O., 1997. A four-step camera calibration procedure with implicit image correction. Proc. IEEE Conf. on Computer Vision and Pattern Recognitio, 1106–1112.

Henriksen, A. V., 2016. Camera-assisted dynamic positioning of rovs. Master Thesis, NTNU Trondheim Norwegian University of Science and Technology Department of Marine Technology.

Hunt, R. W. G., 1967. The reproduction of colour.

Jenkins, F. A., White, H. E., 1957. Fundamentals of optics. Tata McGraw-Hill Education.

Kjerstad, i. K., Skjetne, R., Calabro, V., Værnø, S. A., 2017. Full-scale experiments with a robust dynamic positioning tracking control law including acceleration feedforward.

Klein, G., Murray, D., 11 2007. Parallel tracking and mapping forsmall ar workspaces. In: Proc. IEEE ACM Int. Symp. Mixed Augmented Reality. Nara, Japan, pp. 225–234.

Kunz, C., Singh, H., 2008. Hemispherical refraction and camera calibration in underwater vision. In: OCEANS 2008. IEEE, pp. 1–7.

Kwon, Y.-H., Casebolt, J. B., 2006. Effects of light refraction on the accuracy of camera calibration and reconstruction in underwater motion analysis. Sports biomechanics 5 (2), 315–340.

Mo-Bjørkelund, T., 2016. A feasibility study on camera-based slam for control of low-cost rov, unpublished project thesis.

Moré, J. J., 1978. The levenberg-marquardt algorithm: implementation and theory. Numerical analysis, 105–116.

Mur-Artal, R., Montiel, J. M. M., Tardós, J. D., 2015. ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE Transactions on Robotics 31 (5), 1147–1163.

Mur-Artal, R., Tardós, J. D., 2014. Fast relocalisation and loop closing in keyframe-based slam. In: Robotics and Automation (ICRA), 2014 IEEE International Conference on. IEEE, pp. 846–853.

OpenCV, 2017a. Opencv hough circle feature detection documentation. `http://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html?highlight=houghcircles`, accessed: June 2nd, 2017.

OpenCV, 2017b. Opencv inrange documentation. `http://docs.opencv.org/java/2.4.9/org/opencv/core/Range.html`.

OpenCv, 2017. Opencv librairy home page. `http://opencv.org/`, accessed: June 8th, 2017.

ORB-SLAM, 2017. github page of orb-slam. `https://github.com/raulmur/ORB_SLAM`, accessed: June 2nd, 2017.

Phelan, R. M., 1971. Pseudo-derivative-feedback (pdf) control. Tech. rep., California Univ., Livermore. Lawrence Radiation Lab.

Processing, M. I., 2015. 3d-modeling of seafloor structures from rov-based video data. `http://www.mip.informatik.uni-kiel.de/tiki-index.php?page=3DBlackSmoker`, accessed: May 31th, 2017.

Quigley, M., Gerkey, B., Smart, W. D., 2015. Programming Robots With ROS - A Practical Introduction to the Robot Operating System, 1st Edition. O'Reilly, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, USA.

ROS, 2017. Ros camera calibration package documentation. `http://wiki.ros.org/camera_calibration`, accessed: June 8th, 2017.

Rublee, E., Rabaud, V., Konolige, K., Bradski, G., 2011. Orb: an efficient alternative to sift or surf. In: IEEE International Conference on Computer Vision. ICCV, IEEE, pp. 2564–2571.

Sandøy, S. S., 2016. System identification and state estimation for rov udrone. Master Thesis, NTNU Trondheim Norwegian University of Science and Technology Department of Marine Technology.

Smith, A. R., 1978. Color gamut transform pairs. ACM Siggraph Computer Graphics 12 (3), 12–19.

SNAME, 1950. Nomenclature for treating the motion of a submerged body through a fluid. The Society of Naval Architects and Marine Engineers, Technical and Research Bulletin No., 1–5.

Sørensen, A. J., 2012. Marine control systems propulsion and motion control of ships and ocean structures lecture notes.

Thrun, S., Burgard, W., Fox, D., 2005. Probabilistic Robotics, 1st Edition. MIT Press.

Thrun, S., Leonard, J. J., 2008. Simultaneous localization and mapping. In: Springer handbook of robotics. Springer, pp. 871–889.

Wikipedia, 2017a. Hsl and hsv wikipedia page. `https://en.wikipedia.org/wiki/HSL_and_HSV`, accessed: June 8th, 2017.

Wikipedia, 2017b. List of refractive indices. `https://en.wikipedia.org/wiki/List_of_refractive_indices`, accessed: June 2nd, 2017.

Wöhler, C., 2012. 3D computer vision: efficient methods and applications. Springer Science & Business Media.

Yuen, H., Princen, J., Illingworth, J., Kittler, J., 1990. Comparative study of hough transform methods for circle finding. Image and vision computing 8 (1), 71–77.

Appendix A

A.1 Proofs

A.1.1 Proof 1

We seek to prove $\boldsymbol{J(\eta)}^2 = \boldsymbol{J(2\eta)}$ fo our particular $\boldsymbol{J(\eta)}$. We start by presenting $\boldsymbol{J(\eta)}$ in (A.1)

$$\boldsymbol{J(\eta)} = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 & 0 \\ sin(\psi) & cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.1}$$

We then calculate $\boldsymbol{J(\eta)}^2$ in (A.2).

$$\boldsymbol{J(\eta)}^2 = \begin{bmatrix} cos(\psi)^2 - sin(\psi)^2 & -2sin(\psi)cos(\psi) & 0 & 0 \\ 2sin(\psi)cos(\psi) & cos(\psi)^2 - sin(\psi)^2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

Rewriting gives (A.3).

$$\boldsymbol{J(\eta)}^2 = \begin{bmatrix} cos(2\psi) & -sin(2\psi) & 0 & 0 \\ sin(2\psi) & cos(2\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \boldsymbol{J(2\eta)} \tag{A.3}$$

Q.E.D.

A.1.2 Proof 2

We seek to prove that $\boldsymbol{J(\eta)}$ has full rank for all ψ, thus for all $\boldsymbol{\eta}$. We check the determinant of (A.1) in (A.4).

$$det(\boldsymbol{J(\eta)}) = \begin{vmatrix} cos(\psi) & cos(\psi) & 0 & 0 \\ -cos(\psi) & cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \tag{A.4}$$

$$det(\boldsymbol{J(\eta)}) = cos^2(\psi) + sin^2(\psi) = 1 \tag{A.5}$$

Since $det(\boldsymbol{J(\eta)}) = 1$ for all ψ, the matrix $\boldsymbol{J(\eta)}$ has full rank for all $\boldsymbol{\eta}$.

A.2 EKF Matrices

A.2.1 Differentiation of f

$$\frac{\partial \boldsymbol{f}(\boldsymbol{x}(k), \boldsymbol{u}(k))}{\partial \boldsymbol{x}(k)} = \begin{bmatrix} \frac{\partial \boldsymbol{f_1}}{\partial \boldsymbol{\eta}} & \frac{\partial \boldsymbol{f_1}}{\partial \boldsymbol{\nu}} & \frac{\partial \boldsymbol{f_1}}{\partial \boldsymbol{b}} \\ \frac{\partial \boldsymbol{f_2}}{\partial \boldsymbol{\eta}} & \frac{\partial \boldsymbol{f_2}}{\partial \boldsymbol{\nu}} & \frac{\partial \boldsymbol{f_2}}{\partial \boldsymbol{b}} \\ \frac{\partial \boldsymbol{f_3}}{\partial \boldsymbol{\eta}} & \frac{\partial \boldsymbol{f_3}}{\partial \boldsymbol{\nu}} & \frac{\partial \boldsymbol{f_3}}{\partial \boldsymbol{b}} \end{bmatrix} \tag{A.6}$$

$$\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{\eta}} = \frac{\partial}{\partial \boldsymbol{\eta}} \left(\boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu}\right) = \frac{\partial}{\partial \boldsymbol{\eta}} \begin{bmatrix} u \cdot cos(\psi) - v \cdot sin(\psi) \\ u \cdot sin(\psi) + v \cdot cos(\psi) \\ w \\ r \end{bmatrix} \tag{A.7}$$

$$\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{\eta}} = \begin{bmatrix} 0 & 0 & 0 & -u \cdot sin(\psi) - v \cdot cos(\psi) \\ 0 & 0 & 0 & u \cdot cos(\psi) - v \cdot sin(\psi) \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{A.8}$$

$$\frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{\eta}} = \frac{\partial}{\partial \boldsymbol{\eta}} \left(-\boldsymbol{M}^{-1}\boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}\right) = 0_{4\times4} \tag{A.9}$$

$$\frac{\partial \boldsymbol{f}_3}{\partial \boldsymbol{\eta}} = \frac{\partial}{\partial \boldsymbol{\eta}} \left(-\boldsymbol{T}_b^{-1}\boldsymbol{b}\right) = 0_{4\times4} \tag{A.10}$$

$$\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{\nu}} = \frac{\partial}{\partial \boldsymbol{\nu}} \left(\boldsymbol{J}(\boldsymbol{\eta})\boldsymbol{\nu}\right) = \frac{\partial}{\partial \boldsymbol{\nu}} \begin{bmatrix} u \cdot cos(\psi) - v \cdot sin(\psi) \\ u \cdot sin(\psi) + v \cdot cos(\psi) \\ w \\ r \end{bmatrix} \tag{A.11}$$

$$\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{\nu}} = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 & 0 \\ -sin(\psi) & cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.12}$$

$$\frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{\nu}} = \frac{\partial}{\partial \boldsymbol{\nu}} \left(-\boldsymbol{M}^{-1}\boldsymbol{D}(\boldsymbol{\nu})\boldsymbol{\nu}\right) \tag{A.13}$$

$$\frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{\nu}} = diag\left(\begin{bmatrix} -\frac{X_u+2\cdot X_{u|u|}u+3\cdot X_{uuu}|u|u}{m-X_{\dot{u}}} \\ -\frac{X_v+2\cdot Y_{v|v|}v+3\cdot Y_{vvv}|v|v}{m-Y_{\dot{v}}} \\ -\frac{Z_w+2\cdot Z_{w|w|}w+3\cdot Z_{www}|w|w}{m-Z_{\dot{w}}} \\ -\frac{N_r+2\cdot N_{r|r|}r}{I_{zz}-N_{\dot{r}}} \end{bmatrix}\right) \tag{A.14}$$

$$\frac{\partial \boldsymbol{f}_3}{\partial \boldsymbol{\nu}} = \frac{\partial}{\partial \boldsymbol{\nu}} \left(-\boldsymbol{T}_b^{-1}\boldsymbol{b}\right) = 0_{4\times4} \tag{A.15}$$

$$\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{b}} = 0_{4\times4} \tag{A.16}$$

$$\frac{\partial \boldsymbol{f}_2}{\partial \boldsymbol{b}} = 0_{4\times4} \tag{A.17}$$

$$\frac{\partial \boldsymbol{f}_1}{\partial \boldsymbol{b}} = -\boldsymbol{T}_b^{-1} \tag{A.18}$$

A.2.2 Q and R matrices

The $\boldsymbol{Q}$ and $\boldsymbol{R}$ matrices used in the EKFs are presented here.

$$
\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{I}_{4\times4} & 0 & 0 \\ 0 & 0.1\boldsymbol{I}_{4\times4} & 0 \\ 0 & 0 & \boldsymbol{I}_{4\times4} \end{bmatrix} \cdot 10^{-3} \tag{A.19}
$$

$$
\boldsymbol{R} = \boldsymbol{I}_{4\times4} \cdot 10^{-4} \tag{A.20}
$$

A.3 Attachments

In the attachment folder the following are included:

- uDrone Simulink model.
- Raw data from all tests presented in this thesis.
- Videos and pictures from some of the tests.
- All written code used, without libraries or ROS. The contents are both results of the work of previous students and the author. The posref package is written solely by the author, while the EKF.cpp and slamAutoPos.cpp files in the udrone package are altered by the author.
- The altered ORB-SLAM code, including camera calibration file.
- Master's Thesis presentation poster.