NTNU

Norwegian University of
Science and Technology

Simultaneous Path-Generation and -Following for an ROV

Mads Sig Skinderhaug

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: Skinderhaug, Mads Sig

Field of study: Marine control engineering

Thesis title (Norwegian): Samtidig banegenerering og -følging for ein ROV

Thesis title (English): Simultaneous Path Generation and Following for an ROV

Background
For low-cost consumer-grade Remotely Operated Vehicles (ROVs) there is an increased requirement on autonomy to aid user-friendliness. While professional ROV systems are typically operated by skilled teams and technicians, whom are able to steer the vehicle manually and handle operational faults and unforeseen events, the general buyer of a low-cost ROV does not necessarily possess such skills. In addition, since a target consumer is typically not buying an ROV per se, but rather a platform for underwater experience – a telepresence into the deep, there is a need for the user to better perceive and orientate oneself in the underwater environment.

To this end, this project will investigate how controlling the ROV can be simplified using a path-generation and -following control mode, taking an online user input to generate the desired path with subsequent following control. We call the concept for "Simultaneous Path Generation and Following" (SPGF). In addition, it will look at Human-Machine Interfaces (HMIs) to facilitate these control modes.

The low-cost ROV uDrone was developed in autumn 2015 by a student team at the Department of Marine Technology at NTNU. The objective of the vehicle is to be a testbed for control, estimation, and user interface functions in the NTNU MC-lab. This vehicle will be used to implement a proof-of-concept for the path-following and SPGF control modes.

Work description
1) Perform a background and literature review to provide information and relevant references on:
   a) The ROV uDrone, MC-Lab, and BluEye Robotics drones.
   b) Path-generation and path following for marine vehicles, incl. Serret-Frenet equations and The Maneuvering Problem.
   c) User eXperience Interface facilitating motion control of ROVs.

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

2) The Serret-Frenet equations can be used to generate a path online, using the curvature as an input signal. Show the mathematical derivations of these equations based on the work note "ROV control modes and functions". Implement the equations and visualize how a path is generated with a certain horizon.

3) Derive a control law for the ROV uDrone, that ensures path-following of the generated path.

4) Propose SPGF alternative methods for:
   a) Generate path, freeze it, and follow it with online desired path speed from user controller. Discuss how the user set the desired heading and execute it.
   b) Generate and follow path simultaneously, with online desired path speed from user controller. What should be the desired heading? Make sure the online-generated path is feasible, e.g. by relating maximum allowed curvature to present desired speed (you could elaborate on a figure showing a curve relating feasible speed and curvature for the vehicle).

5) Propose a user interface, that is, the how the path and its data is projected on to the video stream from the ROV and how the path, heading, and speed is set from the controller.

6) Implement the system for ROV uDrone and demonstrate the concept.

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): Andreas Viggen Henriksen (BluEye Robotics), Espen Jørgensen (EGGS Design)

Trondheim,

_____
Roger Skjetne
Supervisor

Preface

This report is the result of a Master's thesis from the department of Marine Technology at NTNU. It is part of the program Marine Cybernetics, during the autumn semester of 2017.

This thesis is a continuation of my work in the project thesis *A prestudy on optimized User eXperience and Interface for low-cost ROV*, and builds upon this. Parts of this thesis are taken from that project thesis and expanded upon in this thesis.

The reader is assumed to have a background in marine cybernetics or vessel control theory, as well as a basic understanding of computer science.

Trondheim, June 11, 2017

_____

Mads Sig Skinderhaug

Acknowledgment

I would like to thank Professor Roger Skjetne for his invaluable help and guidance. The input for the people at BluEye Robotics has also been a great help. Lastly I would like to thank my fellow student for the great collaboration we had during the laboratory work.

M.S.S.

Summary

This thesis starts by introducing the facilities and systems used while conducting experiments. It then goes on to present the methods and techniques used for rendering and superimposition, to project a curved path onto the video feed from a Remotely Operated Vehicle.

Next, path-generation and path-following are discussed, and the maneuvering problem is introduced. The Serret-Frenet equations for curves are introduced. Then the controller, observer, and user input scheme used in this thesis are presented. The result from the tests on path-following are then presented and discussed.

The thesis then goes on to explain the implementation of simultaneous path-generation and -following. The results from these tests are presented and discussed.

From the laboratory experiment it was found that path-following and simultaneous path-generation and -following have a definite application for underwater ROVs, but some changes need to be made to optimize the performance. For simultaneous path-generation and -following, the algorithms need to be improved, both for computational performance and for physical performance.

Sammendrag

Denne oppgaven starter med å introdusere fasilitetene og systemene som brukes i utføringen av eksperimentene. Den fortsetter deretter med å presentere metodene og teknikkene som brukes for gjengivelse og overlegning av en buet bane på videoen fra et fjernstyrt kjøretøy.

Deretter diskuteres sti-generasjon og sti-følging, og manøvreringsproblemet blir introdusert. Serret-Frenet-ligningene for kurver blir introdusert. Deretter presenteres kontrolleren og observatøren som brukes i denne oppgaven, samt måten brukeren kontrollere systemet. Resultatene fra testene på banenfølging presenteres og diskuteres.

Avhandlingen fortsetter med å forklare implementeringen av samtidig banegenerering og -følging. Resultatene fra testene av dette presenteres og diskuteres.

Fra laboratorieforsøkene ble det funnet at banefølge og samtidig banegenerering og -følging definitivt har en anvendelse for undervanns-ROVer, men at endringer må gjøres for å optimalisere. For simultan sti-generasjon og -følging må algoritmene forbedres, både i beregningsytelse og fysisk ytelse.

Contents

List of Figures

Nomenclature

2D Two-Dimensional

3D Three-Dimensional

CSS Cascading Style Sheets

Distro Common name for Linux distribution

DOF Degrees-Of-Freedom

DP Dynamic Positioning

GNSS Global Navigation Satellite System

HMI Human-Machine Interface

HTML Hyper-Text Markup Language

IR InfraRed

LQR Linear-Quadratic Regulator

OS Operating System

ROS Robot Operating System

ROV Remotely Operated Vehicle

SLAM Simultaneous Localization and Mapping

SPGF Simultaneous Path-Generation and -Following

UAV Unmanned Aerial Vehicle

UUV Unmanned Underwater Vehicle

UI User Interface

Chapter 1

Introduction

This chapter presents the background for the problem explored in this thesis, and the problem formulation. The scope and limitations are also laid out. Some of the underwater ROVs that are possible targets for the application are also presented.

1.1 Background

In the last few years the popularity of consumer grade ROVs has shot up. Specially the aerial drones have experienced an explosive rise in popularity, with companies like DJI and Parrot creating UAVs, at a low enough cost, to capture the attention of the non-professional market.

The underwater drone has until now not had the same development, due to the difference in operating environment. The challenges of the underwater environment means that more specialized skills are required for development. This leads to higher costs of both development and production. As a result of this, focus has been on the professional market, where unit cost is less critical. How-

ever, with companies like BluEye Robotics and OpenROV, the low-cost, consumer grade underwater drone seems to be within reach.

When focus is shifted towards the non-professional market, user-friendliness becomes much more critical. In the professional sector highly trained operators control the ROVs. These operators do not require, and often prefer not to use, the provided features for autonomy. For home-users without prior training to be able to focus on the experience of exploring the underwater environment, rather then focusing on the technical aspect of controlling the ROV, a higher level of autonomy is required.

With this as motivation, the thesis will explore path-following and SPGF as possible control modes, simplifying the operation of an ROV by showing the user graphically where the ROV will go with a given input from the user.

The main literature used in this thesis is

Work Note: ROV control modes and functions by Roger Skjetne(Skjetne, 2017), which describes control modes for ROVs, and the equations used for path-generation. More on this can be found in section 3.1.1.

The Maneuvering Problem, the Ph.d. thesis of Roger Skjetne(Skjetne, 2005). This was used as a basis for the maneuvering problem as described in section 3.1.1.

Handbook of Marine Craft Hydrodynamics and Motion Control by Thor I. Fossen(Fossen, 2011), provided general theory for motion control of marine vessels.

In addition to these three main sources, the Master's thesis of Stian Skaalvik Sandøy(Sandøy, 2016) and Torkil Eide Solstad(Solstad, 2016) were used as references on the existing systems for the uDrone, mainly algorithms for the controller and observer, and user interface.

The concepts of path-following are well developed, and have been implemented for both surface and sub-surface vessels. For sub-surface vessels the most notable implementation if in AUVs, where the vessel executes a predefined mission with no, or minimal, input and corrections from an operator. In this thesis, path-following will be implemented for the uDrone, and to explore the concept of SPGF will be explored through laboratory experiments.

1.2 Objectives

The main objectives of this Master's thesis are

1. Implement a path-following algorithm for underwater ROV uDrone with generation of curved paths

2. Develop an HMI which facilitates path-following

3. Conduct experiments to explore the stability of the path-following algorithm

4. Extend the path-following algorithm with simultaneous path-generation and -following

5. Conduct experiments to explore the feasibility of simultaneous path-generation and -following

1.3 Limitations

The algorithm implemented in this thesis will be limited to 3-DOF, surge, sway, and heading. Depth will be controlled using a constant setpoint controller.

It is assumed that the ROV absolute position is available in the system. The report will therefore not focus in depth on the method used to acquire this data, only mention possible solutions, and explain conceptually how they work.

The existing systems for the uDrone will be used. This includes the controller and observer algorithms, and the existing HMI.

1.4 Approach

To verify that path-following and SPGF are viable control modes, experiments will be conducted on the ROV uDrone in the MC-lab. The implemented path-following algorithm will be kept simple, with paths based on the Serret-Frenet equations, as a proof-of-concept. The existing components of the uDrone, like the LQR controller and EKF algorithm created by Stian Skaalvik Sandøy in his Master's thesis (Sandøy, 2016), are utilized, so that focus can be kept on the main problem of the project.

1.5 Development Method

The software development done in this thesis was conducted loosely based on the V-model. The concept of the V-model is illustrated in figure 1.1, which also shows where the name of the method comes from. The basic theory is to start with the overall features, then progressively break these features down to pieces which are easily programmable. During the break down, tests are devised, which can ensure that each of the pieces work as they should, both alone and together.

When all the pieces and tests are layed out, the programming and testing starts, working backwards, ending up with a completed software.

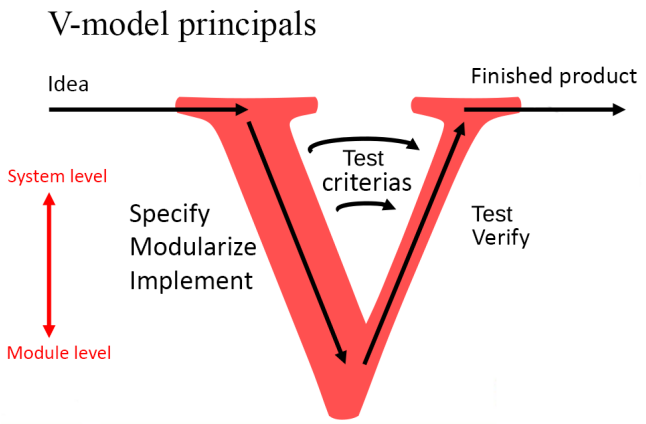

Figure 1.1: Illustration of the V-model software development method (Øyvind Stavdahl, 2016)

The V-model would work perfectly in an ideal world, but in the imperfect world we live in, there will always be something that has been overlooked. Consequently there will be some jumping backwards and forwards between the stages of development.

1.6 Consumer Grade ROVs

In this section a few of the ROVs that are possible targets for the SPGF control mode are presented.

Blueye is a start-up company producing high quality, consumer market underwater ROVs. As of early 2017, they are in the process of developing and testing their first production ready product, the Blueye/Pioneer, hereafter called Pioneer. The Pioneer has a planned releasing at the end of 2017.(Blueye, 2017)

The Pioneer, is a highly capable UUV, or ROV. The goal of Blueye is ultimately to create a product which is ready to go out of the box. This allows the end-user to focus on the experiences of underwater exploration. Although it mainly targets the consumer market, the Pioneer has enough power and is sophisticated enough that it has many applications in a more professional market. Markets where a large, professional UUV would not be financially viable, such as inspection of a cruise ship's hull and propellers for damage and fouling while in harbor (Dypkoren, Erik. Informational meeting, 8 Sept. 2016).

Figure 1.2: Blueye/Pioneer (source: blueye.no)

Blue Robotics produce the BlueROV2 (figure 1.3), the successor to the BlueROV which uDrone is based on. The BlueROV2 takes Blue Robotics away from kit based products, where electronics and software had to be provided by the buyer, to complete and ready to use ROVs. This widens the target group for their products, from researchers and advanced hobbyists, to anyone who wants to explore the briny deep. They do still sell components, giving the possibility of creating a custom ROV.

Figure 1.3: BlueROV2 (source: bluerobotics.com)

OpenROV has a business model much like Blue Robotics. They started out selling ROV kits, but provided more of the electronics then Blue Robotics used to do. OpenROV is currently selling both the kit based OpenROV, and their new, pre-built flagship *Trident* (see figure 1.4).

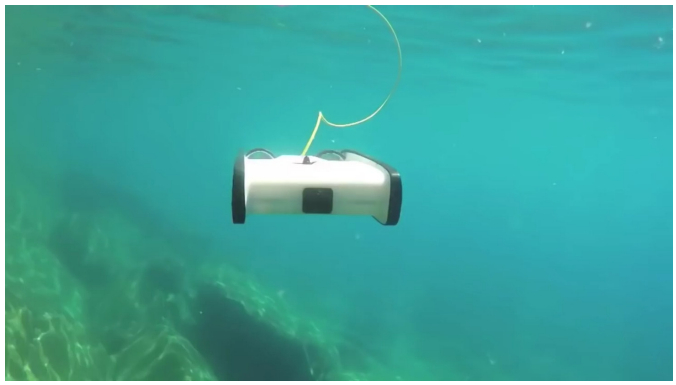

Figure 1.4: OpenROV Trident (source: openrov.com)

VideoRAY creates small, inspection class ROVs. They mainly focus on the professional inspection market, as well as researchers. But they are also trying to break into the high-end leisure markets, like sport-fishing, yachting, and wreck

diving.

Figure 1.5: VideoRay Pro 4 (source: videoray.com)

1.7 Structure of the Report

- Chapter 2 introduces the experimental equipment, laboratory setup, and the interfaces between the system and the operator
- Chapter 3 contains path-generation and -following, and introduces the maneuvering problem. The results from the path-following experiments are presented
- Chapter 4 introduces SPGF, and the results from the tests
- Chapter 5 contains last discussions, conclusions, and recommendations for further work

Chapter 2

Experimental Setup and HMI

In chapter 2 the experimental platform and implemented HMI system is presented. The graphical representation of the path is presented.

2.1 MC-lab

The MC-lab, located at *Marinteknisk senter*, is used to conduct experimental testing of marine control systems, as well as on other cybernetics systems like SLAM algorithms. It consists of a control room, and a $40m$ long, $6.45m$ wide, $1.5m$ deep basin. The basin has a towing carriage, wave and current generators, and a Qualisys motion capture system(Heyn et al., 2016). For more on the Qualisys system see section 2.1.1.

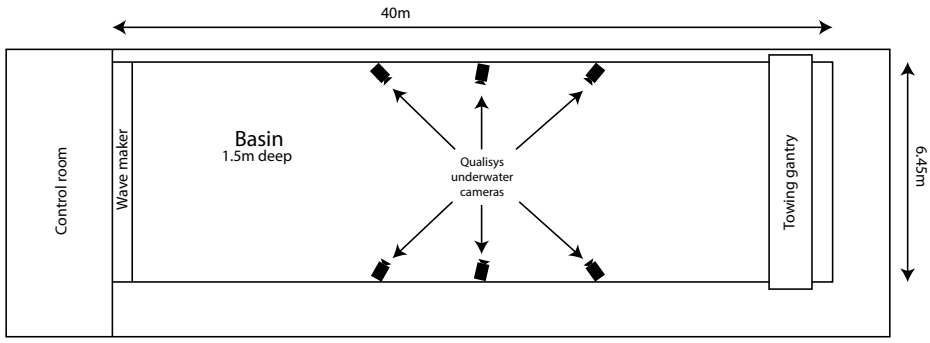

Figure 2.1: Drawing of the Marine Cybernetics laboratory

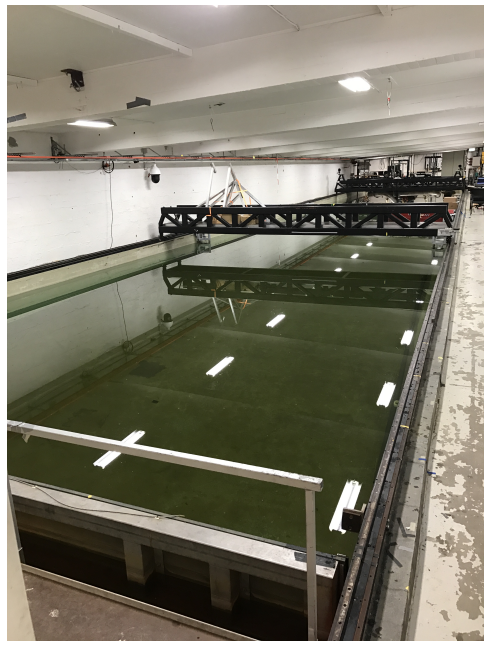

Figure 2.2: Drawing of the Marine Cybernetics laboratory

2.1.1 Qualisys

Since this project contains itself to laboratory experiments, the position reference system Qualisys will be utilized.

Qualisys is an IR camera-based positioning system, used for anything from engineering and medical applications, to entertainment. The Qualisys system consists of two or more Oqus IR cameras (see figure 2.3) and a host computer. The host computer publishes the vessel position and orientation through an ethernet connection. The vessel becomes visible to the IR cameras by placing reflective dots(see figure 2.4) on it, thus the Qualisys system can calculate it's position and orientation(Qualisys, 2015).

Figure 2.3: Qualisys Oqus camera for underwater applications (source: qualisys.com)

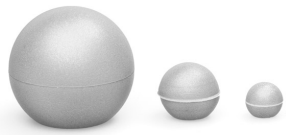

Figure 2.4: Examples of reflective markers (source: qualisys.com)

The precision of the Qualisys motion tracking system is great for testing concepts in the lab. In the real world however, it is unlikely that the ROV position will be know with equal confidence. Usually some sort of dead reckoning is involved. This means that when the Qualisys system is used for testing in the lab, another sensor suits needs to be designed and implemented for the application to work in the real-world.

The Qualisys system gives a position in Cartesian coordinates, and an orientation in quaternions. These are given in reference to an origin defined during the calibration of the system. In figure 2.5 this is illustrated.

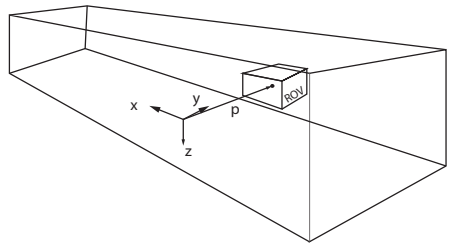

Figure 2.5: Coordinate and reference from Qualisys

The Qualisys coordinates are sent through the observer, and the heading is converted to Euler angles. These observer estimates are the positioning data presented in the results from the laboratory experiments.

2.2 uDrone

The ROV uDrone is a small underwater ROV , based on the BlueROV Kit from Blue Robotics. It was built by a team of students at the department of Marine Technology, NTNU, during the fall of 2015. Figure 2.6 shows the BlueROV. With it's 6 thrusters it is fully actuated. It has been developed to serve as a testbed for cybernetic systems and user interfaces(Solstad, 2016).

As it is based on open hardware and software, it is well suited as a development and research platform, and can interface with most systems through a topside computer.

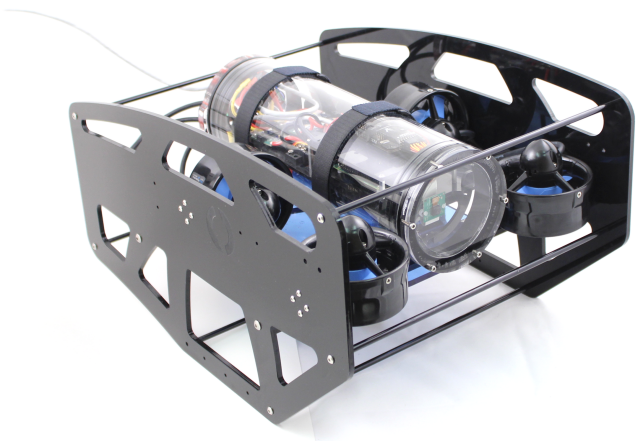

Figure 2.6: Blue Robotics BlueROV (source: bluerobotics.com)

Figure 2.7 shows the uDrone with Qualisys markers and some added buoyancy material, which makes sure the ROV has more buoyancy forces than gravitational force, making it positively buoyant.

Figure 2.7: The uDrone ROV with Qualisys passive reflectors

After testing it was found that fewer Qualisys markers resulted in a more stable calculation of the position. In the end, eight marker were used.

The main, on board computational power comes from a Raspberry Pi, which runs a version of Linux called Emlid Raspbian. The Raspberry Pi handles the communication with the topside computer, as well as the communication with the on board Arduino.

The Arduino Mega 2650 is a microcontroller prototyping platform based on an AVR microcontroller. The Arduino is used to control the thrusters, and to connect and communicate with most of the on board sensors.

2.3 ROS

The Robot Operating System, or ROS, is an open-source robotics software framework. Although it is called an operating system, it is really more of an OS extension. ROS is coded to run on UNIX systems, primarily Ubuntu-based Linux distros. There exists an experimental version for Mac OS X. There also exists libraries for use on multiple architectures, e.g. ARM and AVR(ROS-community, 2016).

ROS simplifies the development of robot systems by implemented the necessary networking facilities, and by providing a way for different sub-systems to interact. By implementing a central piece of software, called roscore, who's address is know to all nodes, this can serve as a phone book, of sorts, for the IP addresses of all the nodes in the system(ROS-community, 2016).

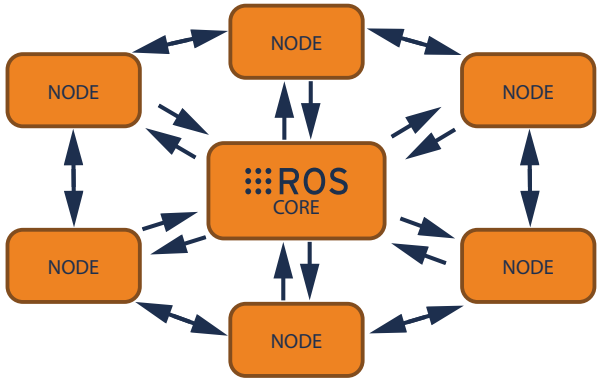

Figure 2.8: ROS core and nodes

The ROS communication architecture allows every part of a control system to be in direct contact with any other part. This results in a system that is very flexible and expandable, because of the modular nature.

There also exists a vast amount of packages which can be utilized to control and monitor just about anything. The ready built packages for video streams, makes

it simple to implement video in an ROV system.

For complicated robot projects the abilities of ROS are nearly indispensable, considering the speed of development it provides.

The individual pieces of software in a ROS application are called nodes. The uDrone software package consists of many nodes. Some of these nodes are the

- Controller
- Observer
- System manager
- Thrust allocation

To get information from the Qualisys system, an open source ROS package written and maintained by KumarRobotics(KumarRobotics, 2017) is used.

ROS also implements many functions for time keeping and timed loops. Using these functions the control system of the uDrone is set to run at a frequency of $10Hz$.

2.4 Graphical Representation

The graphical representation of the path, and the path-following controls will be a continuation and expansion of the work done by Torkil Eide Solstad on the uDrone user interface (Solstad, 2016). The existing user interface was extended with a path-following mode, which initializes the necessary, underlying processes.

The user interface is based on web technologies, i.e. HTML, CSS, and JavaScript. A browser is used to render the UI to the screen.

HTML describes the structure and contents of a web page, in this case the uDrone user interface. CSS defines the look of the HTML elements. The dynamic parts of the user interface are written in JavaScript, e.g. the communication with ROS.

To represent a three dimensional object on a two dimensional screen, the object has to be transformed from the real-world coordinates, to points on the 2D canvas. This is done using linear algebra and a technique called perspective projection. Since the path is a 2D object, defined in 3D space, this technique must be used.

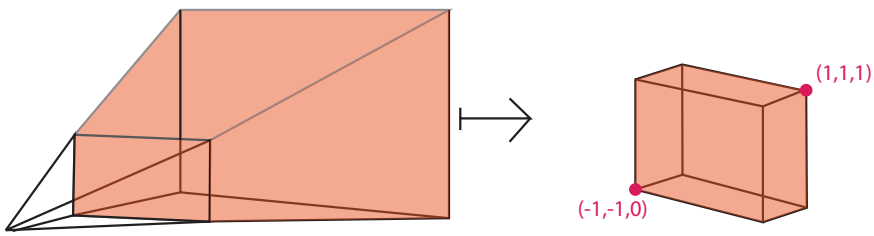

Figure 2.9: Illustration of the mapping done by the perspective matrix

To draw the paths, the WebGL JavaScript API is utilized. This API gives access to the capabilities of the graphic card, which makes rendering to the screen

faster.

Due to the fact that WebGL has a clipping space of ±1, meaning that any coordinate that lies outside of a box with sides ±1 on each side does not get rendered. This means that a projection matrix must be designed so that it scales the real-world coordinates, so all the desired parts of the scene is contained inside the clipping space, as illustrated in figure 2.9. (Danchilla, 2012)

Shaders are small pieces of software that run on the graphics card. These shaders are highly effective at rasterizing, that is, converting vector data to data for individual pixels. Passing points defined in 3D space to these shaders, result in a rasterized drawing on the screen.

WebGL has some predefined shapes it is capable of drawing, like points, lines, and triangles. For compatibility with older graphic cards, squares and other polygons are not supported. The shape used to draw the path is a triangle strip, see figure 2.10. (Danchilla, 2012)

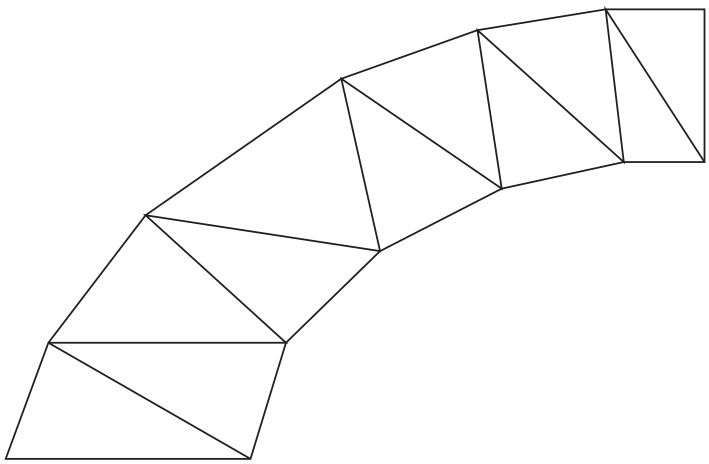

Figure 2.10: Concept sketch of the triangle strip element of WebGL

To get the vertices, or points, from the Serret-Frenet equations(see section 3.1.1),

a strip width is defined and (2.1) is used.

$$v = p \pm \frac{w_s}{2} * N \tag{2.1}$$

Here

- v - position of two vertices equally spaced, normal to the central path
- w_s - strip width
- N - unit normal vector of the path

This is illustrated in figure 2.11.

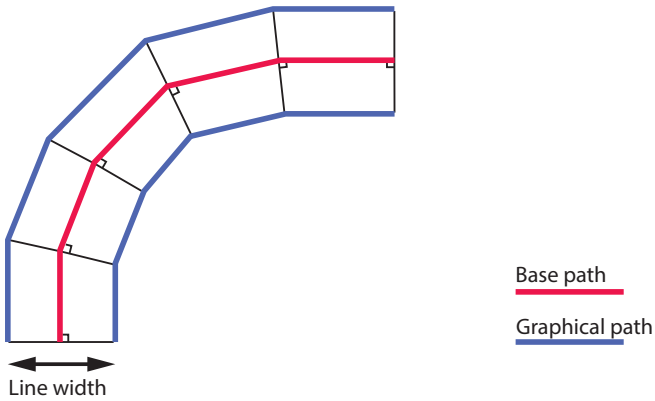

Figure 2.11: Principal sketch of the strip width

The generated path illustration can the be superimposed onto the video feed from the ROV by setting the background color to fully transparent, so that the feed is visible through the WebGL canvas.

Figure 2.12 shows a screen shot from the user interface during the development with a dummy background. Her the vessel is placed exactly at the point and heading of the path initialization. This results in the path being drawn starting in the center of the screen.

Figure 2.12: Graphical representation of path from development

Figures 2.13 and 2.14 shows screen shots from of the user interface during operation in the MC-lab. Her the path is drawn of center and slightly ahead of the vessel. This is because the vessel is not perfectly centered on the path, and the path is drawn with reference to the origin of the Qualisys coordinate system.

Figure 2.13: Graphical representation of path from laboratory experiments

Figure 2.14: Graphical representation of path from laboratory experiments

2.5 Discussion

During testing the implemented representation of the path gives a good impression of where the path will take the vessel. Perspective projection could be tweaked slightly. Basing the projection matrix solely on the camera parameters, seemed to rendered a path which did not give the optimal feeling from a user perspective. It was a good starting point for tuning.

This may also be because the camera behaves differently under water, so a way to calibrate the camera under water might help to ensure a good and accurate path reproduction.

A feature which would improve path-following mode would be to render the locked-in path, as well as the path being set. At the moment it is a bit hard to know where the vessel is heading after the path has been locked, and the path-following has started. A solution to this was attempted, however, rendering two paths made the drawing of them intermittent. This is assumed to be a property of the WebGL API, and that a different algorithm needs to be deviced to accom-

plish this goal.

For aesthetic reasons the path could preferably be extended backward past it's start so the end of the path is never seen.

A UI element showing the set desired velocity in plain numbers would help the operator gain a feel for the forward progress, as it is slightly hard to get a good feel just from watching the camera feed.

An alternative idea that came up, was to use the color of the path as an indicator for desired speed. This might be an interesting visual feedback, but a plain number indicator would give a better feel for the actual speed. The color path would give some indication, but not a precise one.

Chapter 3

Path-Generation and -Following

In this chapter the generation and following of curved paths by underwater ROV's will be explored. The maneuvering problem, Serret-Frenet equations, and test results are presented

3.1 Background and Theory

The reason for implementing path-following for underwater ROVs, is that it is thought to make control and maneuvering simpler for the operator.

3.1.1 The Maneuvering Problem

To define the desired behavior of a vehicle it is practical to divide the problem into two separate parts

- The Geometric Task
- The Dynamic Task

This method of control is called the Maneuvering Problem, and is detailed in the Ph.D. thesis of Roger Skjetne (Skjetne, 2005).

The advantage of this approach is that the geometry of the path, and the dynamical behavior along the path can be defined and controlled separately. This means that a path can be defined, then the velocity can be controlled without having to regenerate the path.

The Geometric Task

The geometric task can be stated mathematically as in (3.1)

$$\lim_{t\to\infty} |y(t) - y_d(s(t))| = 0 \tag{3.1}$$

Where

- $y(t)$ - vessel position
- $y_d(s(t))$ - desired position
- $s(t)$ - parametrization variable at time t

For path-following the desired position is dynamic, and given the by desired path. The paths generated in this thesis use the Serret-Frenet equations. These equations make it possible to describe curves with just a few parameters. The equations are easily discretizable making them easy to adapt for use in digital systems.

The parameters are

- κ - curvature of the curve
- τ - torsion of the curve
- s - arc length

Since this project is limited to two dimensional curves, the torsion is always zero and can be disregarded.

Using the arc length s as a parameterization variable affords a simple and natural way of controlling the desired position along the path, as it is in the real-world unit meters, and provides and easy link to the desired velocity.

The implemented equations (3.2) to (3.9), can be found in (Skjetne, 2017).

$$\dot{p}_d(s) = T(s) \tag{3.2}$$

$$\dot{\psi}_d(s) = \kappa(t_0) \tag{3.3}$$

$$\dot{T}_d(s) = \kappa N_d(s) = \kappa \begin{bmatrix} -sin(\psi_d(s)) \\ cos(\psi_d(s)) \end{bmatrix} \tag{3.4}$$

$$\dot{N}_d(s) = -\kappa T_d(s) \tag{3.5}$$

Where

- κ is the curvature
- $\kappa(t_0)$ is the curvature at the time of path initiation, t_0.
- p_d is the desired position

- ψ_d is the desired heading
- $T(s)$ is the tangential vector
- $N(s)$ is the normal vector

Discretized these equations look like

$$p_{d,(l+1)} = p_{d,l} + T_{d,l} ds \tag{3.6}$$

$$\phi_{d,(l+1)} = \psi_{d,l} + \kappa ds \tag{3.7}$$

$$T_{d,(l+1)} = T_{d,l} + \kappa N_{d,l} ds \tag{3.8}$$

$$N_{d,(l+1)} = N_{d,l} + \kappa T_{d,l} ds \tag{3.9}$$

Here

- ds - discrete steps in parametrization variable
- l - discretization variable

l is defined on

$$l =< 0, N_l >, l \in \mathbb{N} \tag{3.10}$$

where N_l is the number of line segments. N_l and ds define the length of the path.

Since the path is discretized and stored as an array, the current desired position is calculated from $s(t)$ using linear interpolation, as in (3.11).

$$x(s(t)) = x_0 + \frac{s - s_0}{s_1 - s_0} * (x_1 - x_0) \tag{3.11}$$

where

- $x(s(t))$ - current desired value (heading or position)
- x_0 - closest stored value below
- x_1 - closest stored value above
- s - current interpolation variable
- s_0 - closest stored interpolation variable below
- s_1 - closest stored interpolation variable above

The Dynamic Task

For this project, the dynamic task of the maneuvering problem will be to follow the path, defined in the geometric task, at a varying desired velocity set through user input. This can be stated as in (3.12).

$$\lim_{t \to \infty} |\dot{s}(t) - v(s(t), t)| = 0 \tag{3.12}$$

where $v(s(t), t)$ is the desired velocity. This requirement is fulfilled, for the discretized system, by implementing (3.13).

$$s_t = s_{t-1} + u_d(t) * dt \tag{3.13}$$

- s_t - parameterization variable at current time step
- s_{t-1} - parameterization variable at previous time step
- $u_d(t)$ - current desired velocity
- dt - update period of the system

3.1.2 Position Data

Path-following requires that the absolute position of the vessel is known. For underwater vessels this is more complex then for surface vessels, since high frequency electromagnetic waves do not propagate well in water. This means that GNSS cannot be used for underwater vessels.

One possible solution, is to use a combination of hydro-acoustical navigation systems and GNSS to get the absolute position. An example of this is the new system being launched by the company Blue Robotics, which they call Underwater GPS (BlueRobotics, 2017).

Other solutions based on hydro acoustics are available, however, most of these require expensive equipment, design for professional use. Some also need a network of transceivers on the seabed.

For the purposes of testing the path-following algorithm, the Qualisys system discussed in section 2.1.1 is used. In the real world it is improbable that such good position data would be available at this time, but for proof-of-concept of path-following this is assumed. With sensor fusion, a good observer, hydro-acoustics, IMUs, and other sensors, good estimates should be available in real-world application as well.

3.1.3 Observer

To filter the input data and estimate the unmeasured states of the system a Discretized Kalman Filter is used. The Kalman filter used for this project is a C++ implementation of the observer designed by Stian Skaalvik Sandøy in his Master's thesis (Sandøy, 2016). This reimplementation was done to simplify the conducting of the laboratory experiments. The original was made in Simulink.

(3.14) to (3.18) gives the implemented Kalman filter.

$$K_k = \bar{P}_k * H^T (H\bar{P}_k H^T + R_k)^{-1} \tag{3.14}$$

$$\hat{x}_k = \bar{x} + K_k[y_k - H\bar{x}] \tag{3.15}$$

$$\hat{P}_k = [I - K_k H]\bar{P}_k[I - K_k H]^T + K_k R_k K_k^T \tag{3.16}$$

$$\bar{x}_k = \phi\hat{x}_k + \Delta u_k \tag{3.17}$$

$$\bar{P}_k = \phi\bar{P}_k\phi^T + \Gamma Q\Gamma^T \tag{3.18}$$

(3.14) calculates the gain of the filter, K_k. In (3.15)-(3.16) the previous estimate is corrected based on the latest measurements and updated filter gain. (3.17)-(3.18) makes the prediction for the next time step.

- K_k - Filter gain
- $\bar{P}$ - Probability of the prediction
- $\hat{P}$ - Probability of the estimate

- H - Measured states matrices
- R - Sensor variance
- Q - Process variance
- $\hat{x}$ - State estimation
- $\bar{x}$ - State prediction
- Δ, ϕ, and Γ - System matrices from the mathematical model of the vessel

3.1.4 Controller

LQR is a model based solution to the regulator problem. It is part of the linear quadratic(LQ) optimal control theory, where the goal is to find a control law which optimizes the linear quadratic cost functional(Fossen, 2011). The cost functions is given in (3.19).

$$J = \min_u \left[\frac{1}{2} \int_0^T (y^T Q y + u^T R u) dt \right] \tag{3.19}$$

where

- Q - state weighting matrix
- R - output weighting matrix
- y - measured states vector, in out case y = $\hat{x}$
- u - output vector

The cost function is minimized by solving an algebraic Riccati equation (3.20), finding a gain matrix G, which can be used to find desired output(3.21).

$$P_\infty A + A^T P_\infty - P_\infty B_a R^{-1} B_a^T P_\infty + Q = 0 \tag{3.20}$$

$$u = Gx \tag{3.21}$$

$$G = -R^1 B^T P_\infty \tag{3.22}$$

Once again an algorithm for the uDrone has been found by Stian Skaalvik Sandøy (Sandøy, 2016), and the Simulink implementation has been rewritten in C++ for practical reasons.

3.1.5 User Input

For path-following the user input comes from an XBOX 360 hand controller. The controls are as defined in figure 3.1.

Figure 3.1: User input for path-following

During testing, a possible connection between path curvature and desired velocity will be explored. This is to see if there are limitations on the velocity, when following a path with high curvature. If this is so the input of the velocity controller might have to be changed from (3.23), to (3.24).

$$v = k_c x \tag{3.23}$$

$$v = k(\kappa) x \tag{3.24}$$

where

- $x \in < -1, 1 >$ - input from the controller

- k_c - constant scaling factor
- $k(\kappa)$ - scaling factor dependent on κ
- v - desired velocity.

3.2 Results

3.2.1 Low Velocity

The figures 3.2 to 3.5 show the ROV response at low desired velocity, approximately $0.05\frac{m}{s}$.

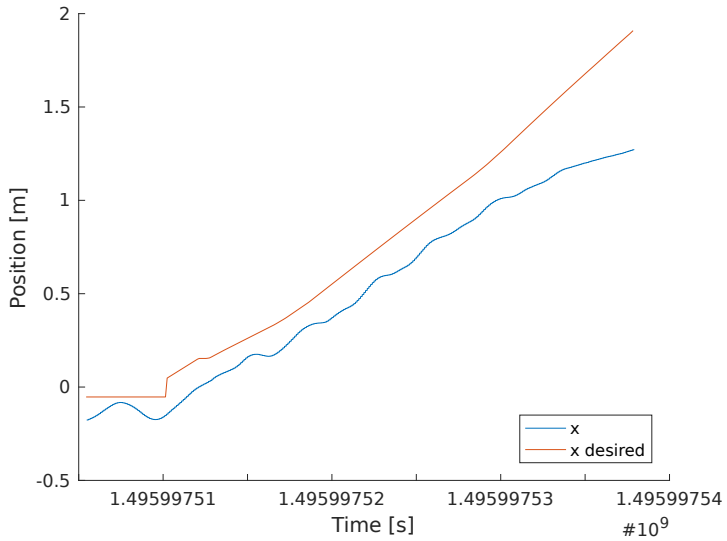

Figure 3.2: Comparison of x position and desired x position at low velocity

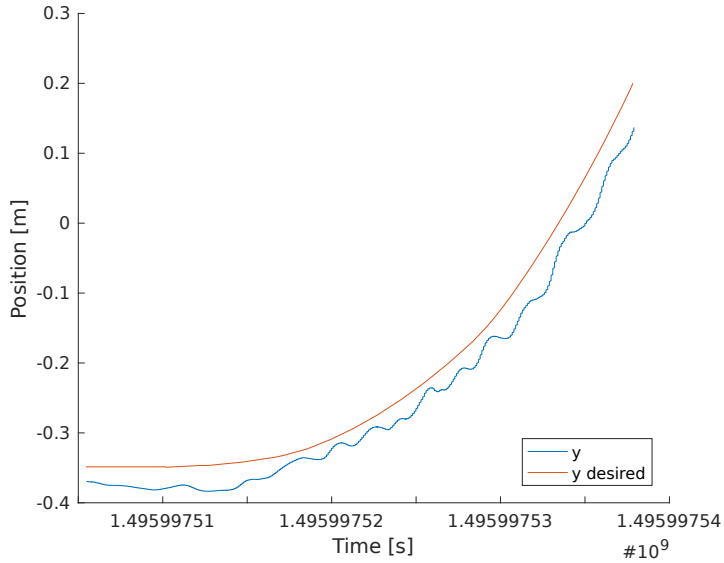

Figure 3.3: Comparison of y position and desired y position at low velocity

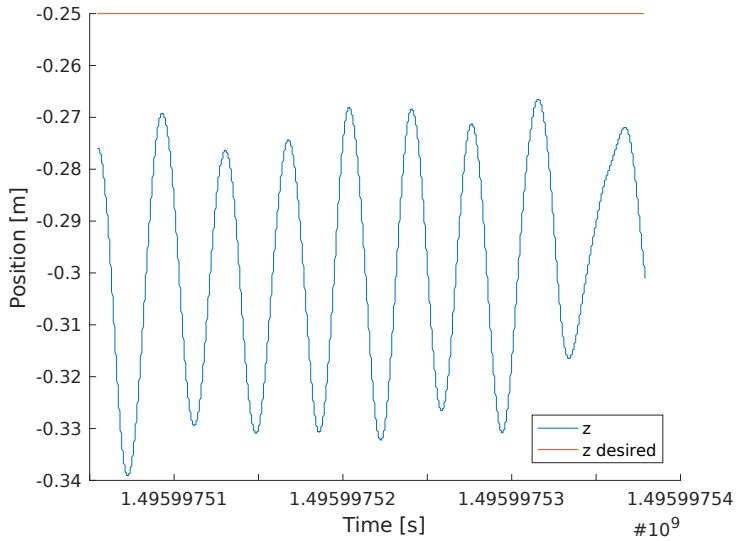

Figure 3.4: Comparison of z position and desired z position at low velocity

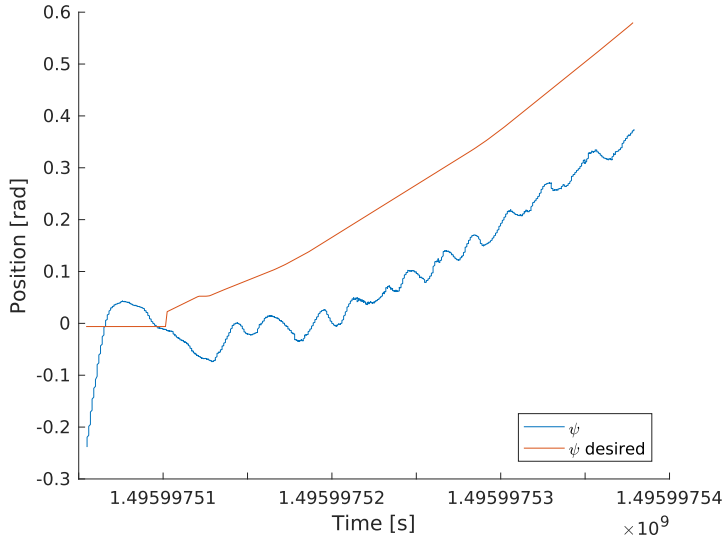

Figure 3.5: Comparison of heading and desired heading at low velocity

3.2.2 Intermediate Velocity

For this run the desired velocity started at about $0.14\frac{m}{s}$, and ended at slightly under $0.2\frac{m}{s}$. The figures 3.6 to 3.9 show the responses at intermediate velocity.

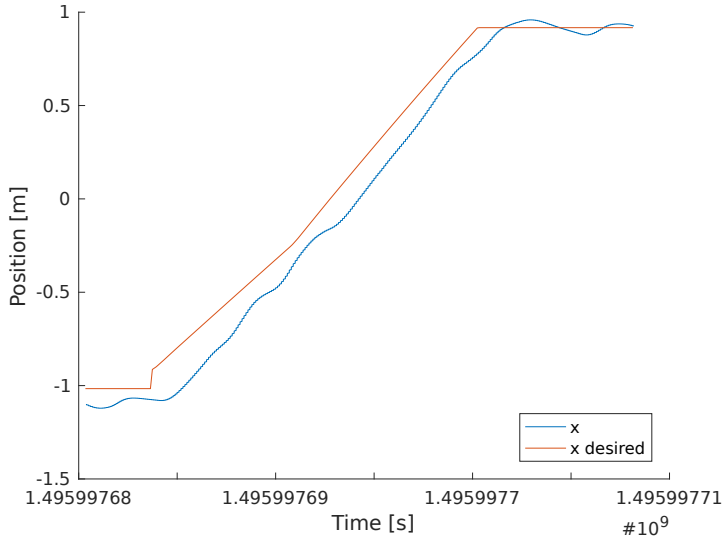

Figure 3.6: Comparison of x position and desired x position at intermediate velocity

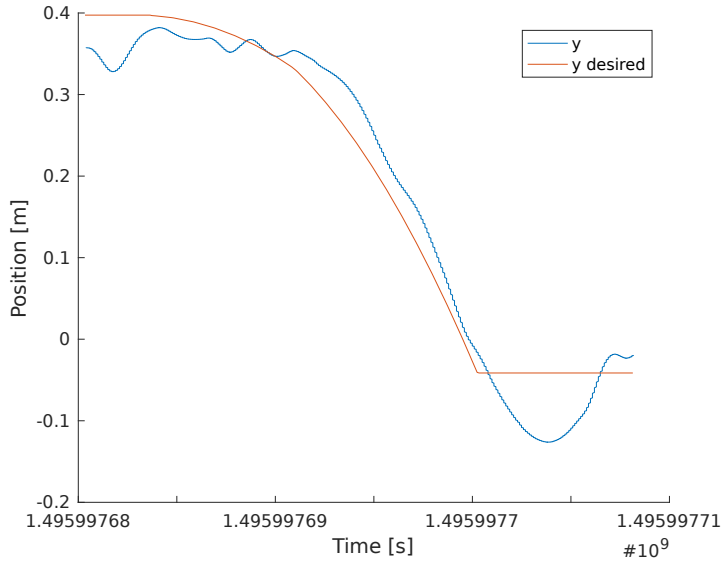

Figure 3.7: Comparison of y position and desired y position at intermediate velocity

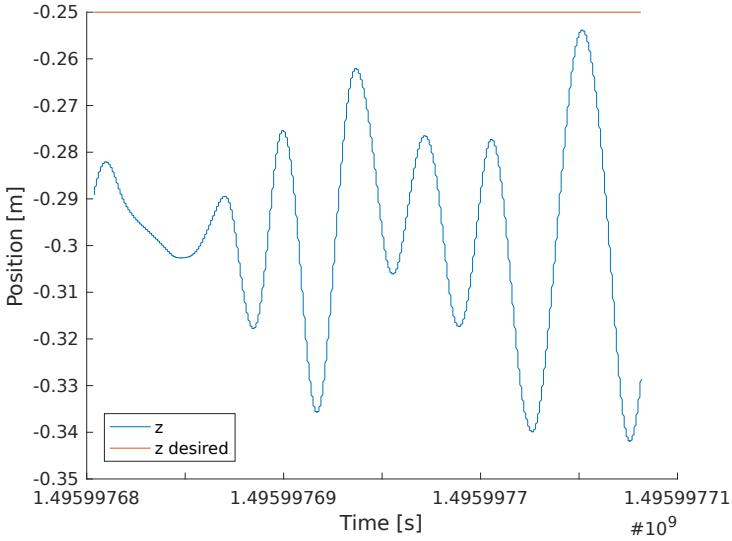

Figure 3.8: Comparison of z position and desired z position at intermediate velocity

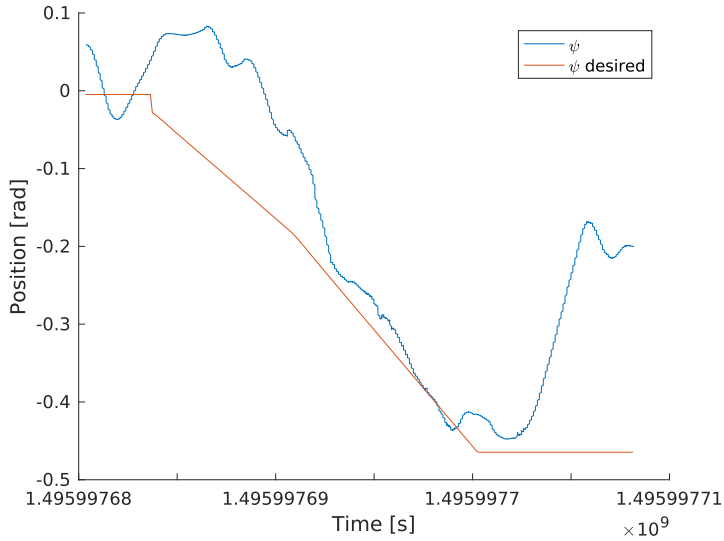

Figure 3.9: Comparison of heading and desired heading at intermediate velocity

3.2.3 High Velocity

The high velocity run was done with a desired velocity of about $0.25\frac{m}{s}$. At this speed, we see that the controller used starts to struggle with keeping up. $0.25\frac{m}{s}$ also seems to be a good limit to ensure the reliability of the Qualisys system, with this setup. In figures 3.10 to 3.13 these results are shown.

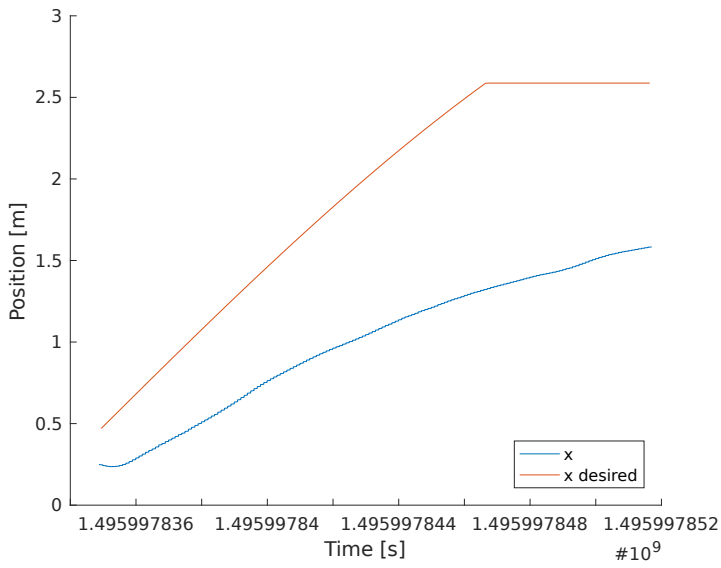

Figure 3.10: Comparison of x position and desired x position at high velocity

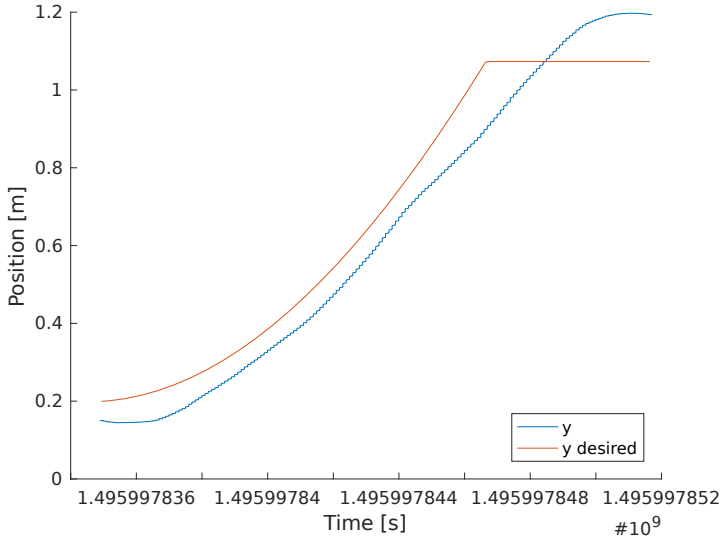

Figure 3.11: Comparison of y position and desired y position at high velocity

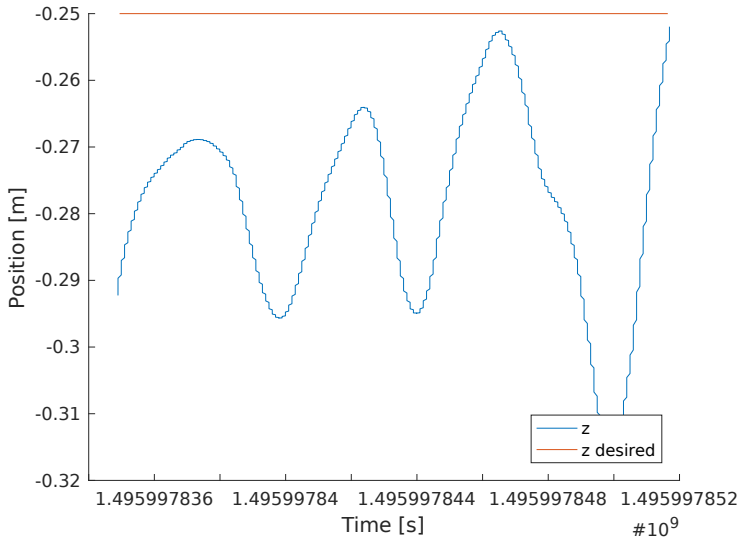

Figure 3.12: Comparison of z position and desired z position at high velocity

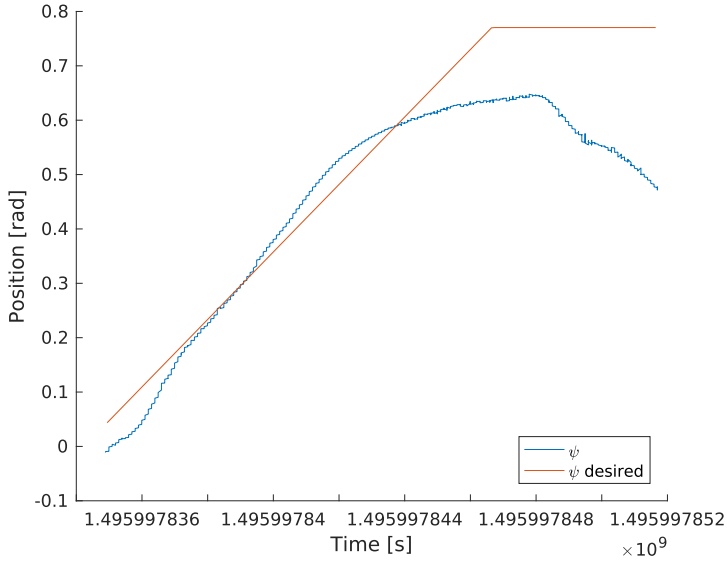

Figure 3.13: Comparison of heading and desired heading at high velocity

3.3 Discussion

Low Velocity

In the plots from the low velocity run, it can be see that the ROV follows the desired position quite well for all directions. At this low desired velocity, the desired position seems to change so slowly that the vessel has the time to oscillate around the setpoint. This is specially noticeable in the y-direction and heading.

The desired velocity in the x-direction is greater than in the other directions. This seems to give a smoother behavior, assumably due to the fact that the setpoint is less often overtaken by the vessel. Towards the end of the run the deviation in the x-direction increases. This seems to coincide with a slight increase in the desired velocity. This might be due to some property of the controller.

Increasing the cost of the position error in the controller tuning, should correct this problem making the controller more aggressive at correct an error in position.

It can also be seen that in all the directions, there is a constant offset from the setpoint. This is due to the fact that the controller does not have integral action, which could compensate for this steady-state error.

Intermediate Velocity

In section 3.2.2 we see that the response of the vessel, when the desired velocity is slightly higher, is much smoother for the x- and y-directions. The heading controller does however seem to be struggling a bit more, most likely due to the fact that the drone is turning the opposite direction when the desired heading starts to change.

Another contributing factor is probably fact that the uDrone has a lot of coupling between the forces in the various directions, as a result of the geometry and the mounting of the thrusters. Thus the rapid changes in forces in the x-, y-, and z-directions probably affects the heading. The thruster outputting most of the force in y-direction is mounted off-center, giving a momentum about the z-axis, which results in a change in heading.

At the end of the path, when the desired position becomes static, there is a somewhat large overshoot in the y-direction, which is quickly corrected. However, once again due to coupling of the directions, the heading is strongly affected. The heading did recover, but the recording stopped before this occurred.

High Velocity

At a desired velocity of about $0.25\frac{m}{s}$, the vessel starts to struggle to keeping up with the desired position. This is clearly seen in the plots in section 3.2.3. It is most clear in the x-direction, where the desired velocity was greatest. At one point the position is almost 1.5 m from the desired position.

It is probable that a more aggressive controller would correct this. The limits on the output of the thrusters are also quite low from the early testing. This was a safety precaution to preventing accidental and undesired behavior, due to undiscovered errors in the code. The thrust reserves of the uDrone should be more than sufficient to overcome this problem.

As with the intermediate velocity case, we see that also at high velocity an overshoot in y-direction results in unwanted behavior in heading.

For all these runs the problem of coupling between the forces imposes a problem for the path-following, since the vessel struggles to reach a steady-state. However, the vessel does follow the path laid out. With some tuning of the observer and controller, the performance should improve.

Implementing a bank of controllers, which can be activated based on desired velocity, might improve the positioning further. Using multiple controllers would allow for a less aggressive controller for static desired position, limiting jerky and erratic movements. One or several more aggressively tuned controllers, automatically selected based on desired velocity, would give better response at higher desired velocities.

User Input

The control of the desired velocity is simple, and works intuitively.

When it comes to setting the path length and curvature, it was found that it is slightly tricky to control both with one joystick. The problem became less pronounced when the sensitivity was tuned and turned down.

One possible option would be to have a fixed path length, and only control the curvature. However, this would reduce the flexibility of the path-generation.

The method of having a fixed path length was implemented for SPGF, and for this it worked very well.

Another possibility is to use one of the trigger buttons on the front of the controller to control path length, thus only one variable needs to be controlled with the thumb.

Chapter 4

Simultaneous Path-Generation and -Following

In this chapter simultaneous path-generation and -following(SPGF) will be explored as a control mode for underwater ROVs, and results from the experiments are presented.

4.1 Background and Theory

If a path can be easily control by the operator, and the vessel is automatically set to follow this path, the theory is that controlling the vessel will be easier and more precise. The thought is that this will eliminate much of the experience needed to estimate where the vessel will end up.

By generating a new path when the user input is changed, setting the start point for this path to the current desired position and heading, a continuous path with varying curvature can be generated.

Figure 4.1 shows the concept of SPGF, where new paths are initialized in the middle of the previous one. The resulting is a smooth, continuous path, with variable complexity, which the ROV can follow.

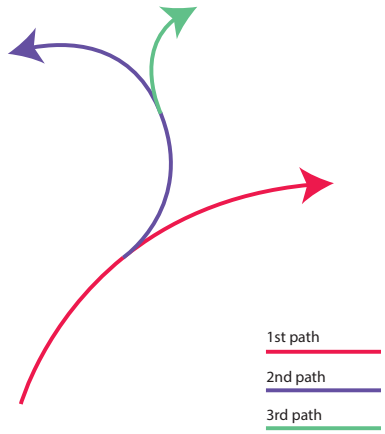

Figure 4.1: Concept sketch of SPGF

As a proof-of-concept, the algorithms from the path-following section were adapted for SPGF.

The main alteration is that a message to update the path is sent when the curvature is changed, without waiting for a button press.

4.1.1 User Input

The control input scheme from the path-following was largely kept for the SPGF implementation. The main difference is that the path length for SPGF is constant. This is because it was found during development that it was difficult to control both curvature, path length, and velocity at the same time, without constantly changing the path.

The implemented controls are shown in figure 4.2.

Figure 4.2: User input for SPGF

4.2 Results

In this section the results from the experiments on SPGF are presented.

4.2.1 First Run

The first run kept the same configurations as for path-following. The only alteration is the automatic sending of new paths.

Figures 4.3 to 4.6 show the results. It can be seen that towards the end of the run the positioning data became unstable.

One thing to note is the small change in y-direction in 4.4.

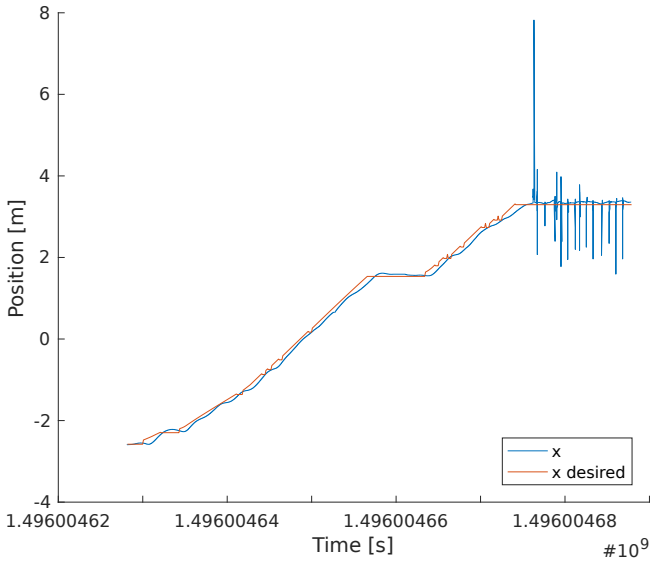

Figure 4.3: Comparison of x position and desired x position

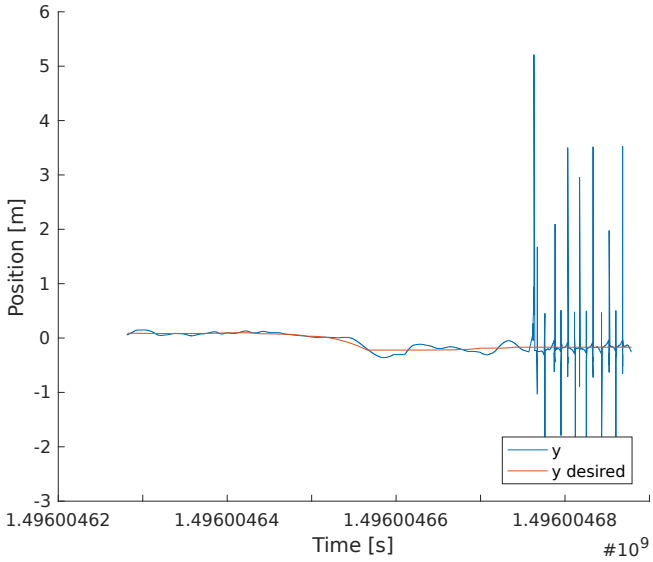

Figure 4.4: Comparison of y position and desired y position

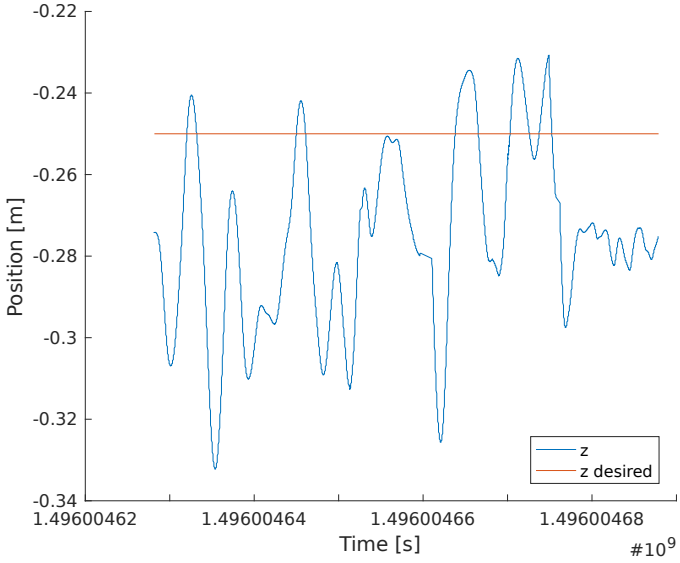

Figure 4.5: Comparison of z position and desired z position

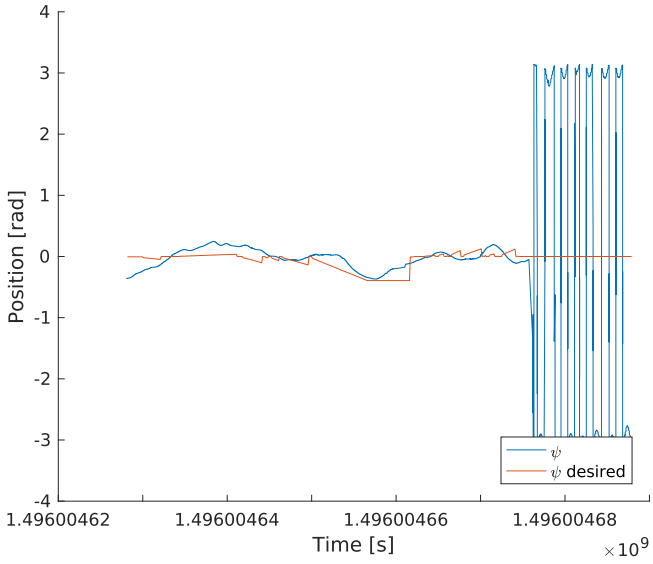

Figure 4.6: Comparison of heading and desired heading

4.2.2 Second Run

For the second run a limit on the path updating was imposed, and the gains of the controller were increased. This gave larger movements in y-direction desired position, and a larger overshoot. The results are shown in 4.7 to 4.10.

Notice also the behavior of the desired heading in 4.10, this is explained in the discussion later in this chapter.

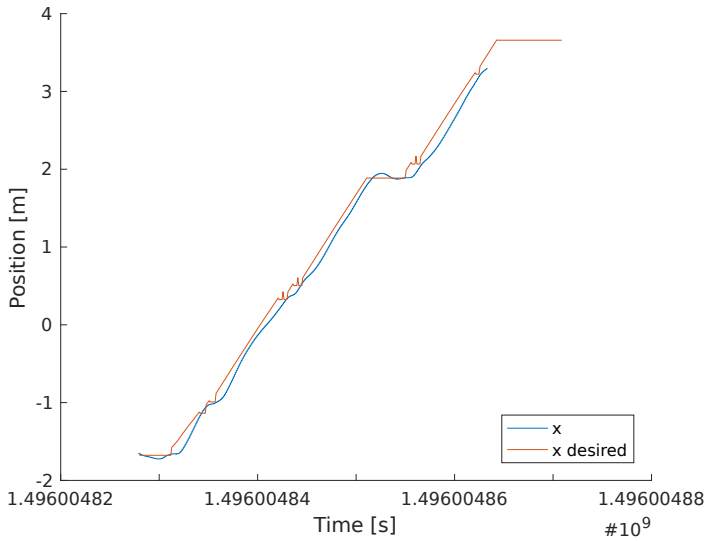

Figure 4.7: Comparison of x position and desired x position

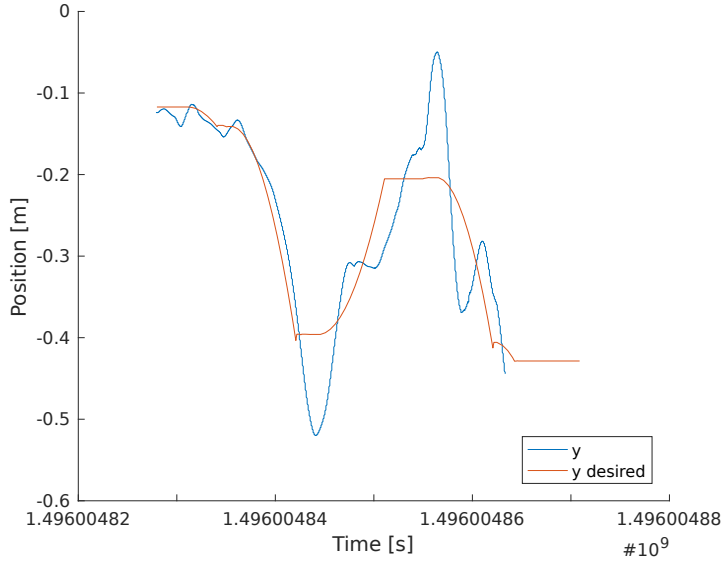

Figure 4.8: Comparison of y position and desired y position

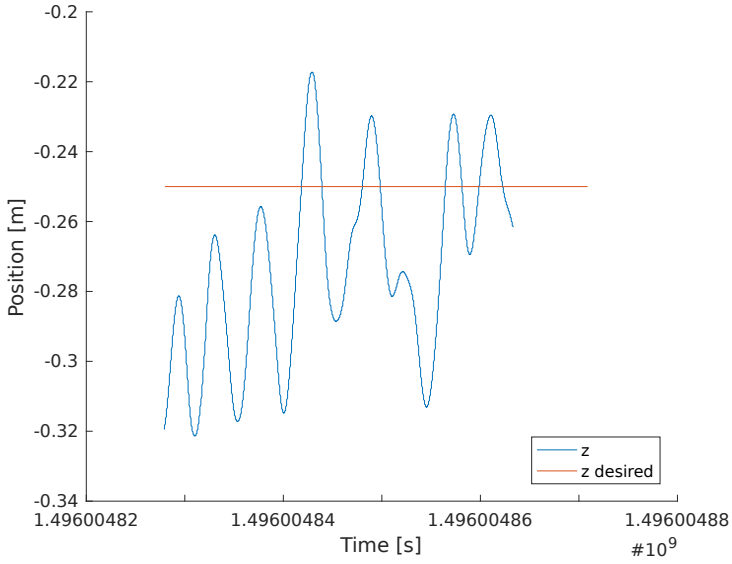

Figure 4.9: Comparison of z position and desired z position

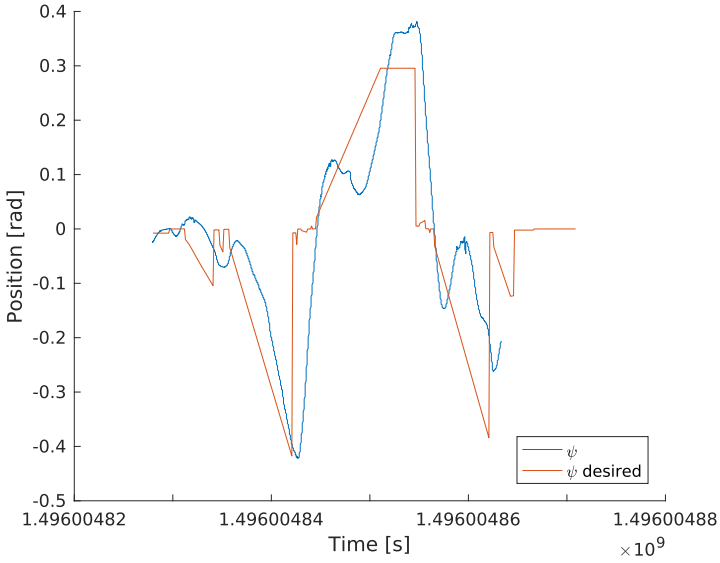

Figure 4.10: Comparison of heading and desired heading

4.2.3 Third Run

The third run was conducted in the same manner as the second run. The controller gains were slightly lowered again. See figures 4.11 to 4.14 for the results.

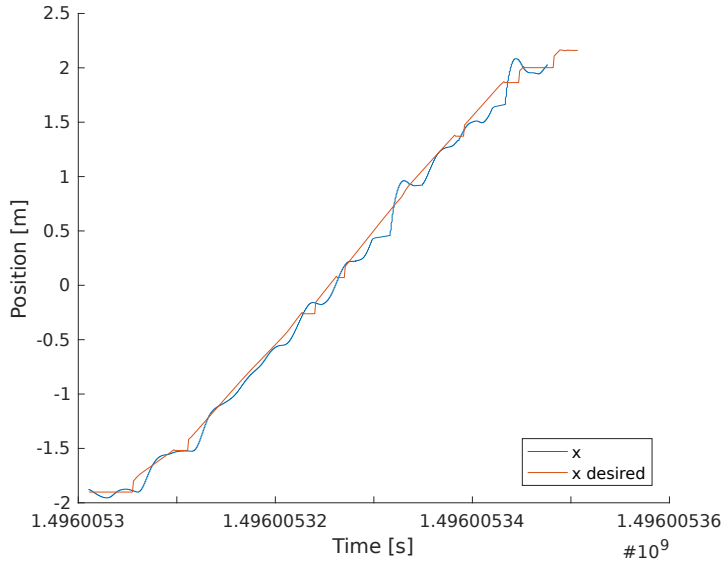

Figure 4.11: Comparison of x position and desired x position

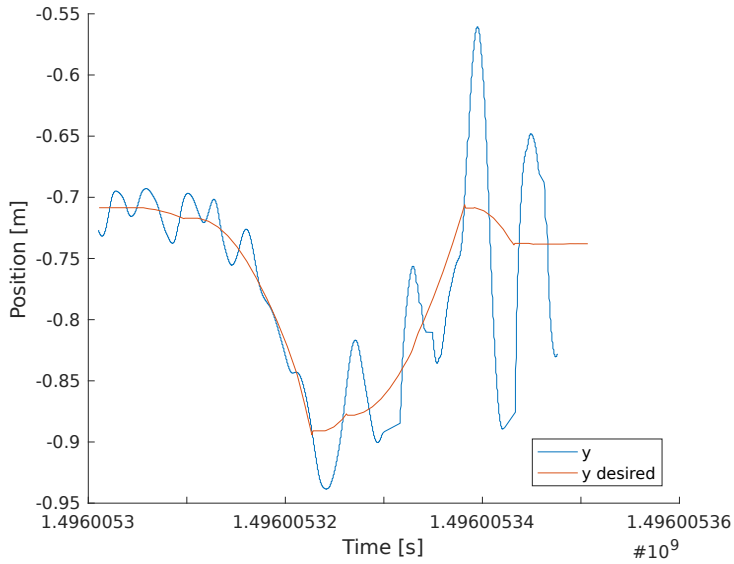

Figure 4.12: Comparison of y position and desired y position

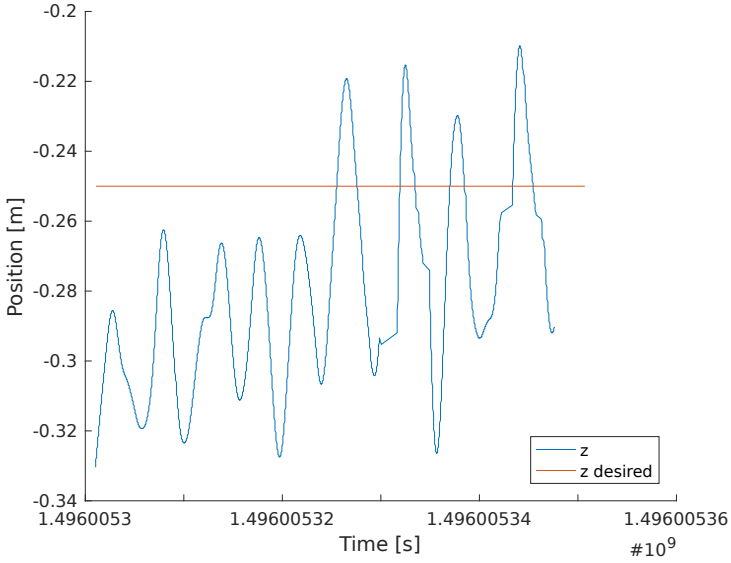

Figure 4.13: Comparison of z position and desired z position

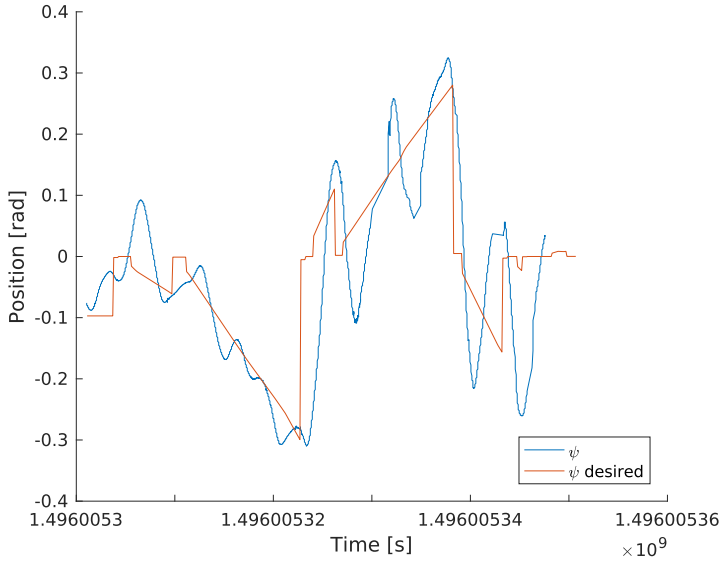

Figure 4.14: Comparison of heading and desired heading

4.3 Discussion

4.3.1 First Run

The first run revealed that a further adaption in the path update was needed for SPGF to work.

A limit was impose on when the path was updated. Due to delays in the calculation of a new path, it seemed the desired position often was not changed since a new path was ordered before the following of the previous had the time to start.

This was overcome by adding a dead-zone around the current curvature, so that a new path is only requested if the change in curvature is greater then a set limit.

At the end of the run a recurring problem with the Qualisys position data is revealed. Due to the system picking up some noise, the position data was rapidly mirrored about the y-axis. In hindsight this might have been improved by trying to place the markers less symmetrically.

4.3.2 Second and Third Run

For the second run the dead-zone workaround and a slight increase in controller gains were implemented.

The changes resulted in better path-generation, with the desired position being updated as desired.

The run revealed that the y-control has significant overshoots upon sudden changes in path direction. This might be, at least partially, due to the increased controller gain.

Due to an error in the code which went undiscovered until the post-processing of the data was done, the initial heading of the path was always set to zero. The initial path heading is supposed to be equal to the previous desired heading. This error probably caused additional instabilities, since it introduces significant discontinuities in the generated path.

For the third run the controller gain was decreased somewhat, this seems to reduce the overshoot in the y-direction. This run also suffers from the same error in initial heading assignment.

Both the second and third run show a problem with the way the path-generation was implemented. The problem is specially noticeable in the x-direction and the heading. Whenever the path is updated, there is a pause in the updating of the desired position, before it is suddenly resumed. It is likely that this is due to the fact that the entire path is generated as an array upon path change. This worked fine for path-following, since the frequency of path updating is much less. For SPGF however, this results in many small discontinuities, generating extra oscillations and instabilities.

A possible solution to this problem would be to redesign the algorithm, so that the desired position and heading could be calculated by a simple function, rather than numerical integrating the entire path in one go. If this calculation took less the one cycle of the control system, $0.1s$ in the case of the uDrone, the problem would be eliminated.

Using the Serret-Frenet equations, and calculating the change in desired position and heading on the fly, might solve the problem. The step in parametrization variable s would be variable. However, as long as the steps do not become to large, the discrete integral should approach the integral of the continuous function. Figure 4.15 shows the difference in the calculation results for 100 line segments and 1000 line segments.

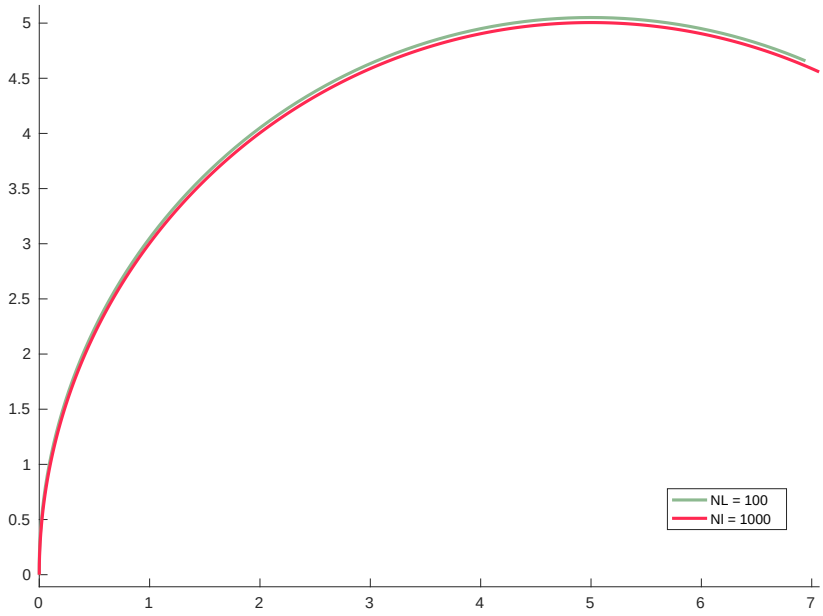

Figure 4.15: Comparison of integration step sizes

It is also probable that part of the problem will be rectified when the heading

initialization is corrected. Since the heading always start out being 0, the movement in the y-direction also starts out being 0, generating the sharp corners in the y-direction path.

Chapter 5

Discussions and Conclusions

In this chapter final discussions are presented, and conclusions on the work are drawn. At the end are the recommendations for further work.

5.1 Discussion

5.1.1 Experimental Difficulties

Since the heading always got initialized to 0, the smooth path desired for SPGF could not be created. The problem was fixed in software, but upon returning to the laboratory to test this, the uDrone did not work. The problem was with the thrusters, which stopped responding to input after the first command. Due to time restrictions the error was not resolved, but it is reasonable to assume that the problem lies with the Arduino. It outputs the thruster command signal. It may also be a problem with the voltage regulator. It is unlikely that all the thruster controllers would fail at the same time, so the problem probably does not lie with them. Consequently it was not possible to test the software fix.

Another problem with the experiments were the stability of the Qualisys measurements, and the size of the work area. During testing the uDrone kept passing outside the area. This meant that the desired velocity had to be relatively low to keep experiments running, but this might not be as big a problem when the heading initialization is fixed.

Even inside the area of recognition the readings were unstable during movement. This was not a big problem when the desired position was near static, so for DP the Qualisys system works great. One solution for use with changing desired position would be to use active markers. These markers emit light on their own, and thus are easier for the cameras to identify, increasing the range of detection.

Implementing some sensor fusion, for example with Qualisys and an IMU, would also increase the accuracy of the state estimation.

5.1.2 Guidance

The lack of a real guidance law results in a lot of lateral, and sub-optimal movement. A guidance law which corrects the desired heading based on the of the velocity of the change in desired position, would result in more of the movement happening in the surge-direction of the ROV. This would allow for a smoother, more efficient movement. This is visualized in figure 5.1.

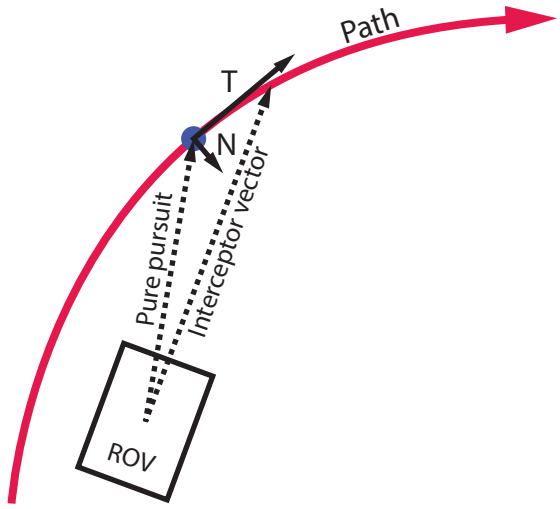

Figure 5.1: Concept sketch of discussed guidance law

Another possible improvement would be to give priority to the heading controller, so that heading is corrected before the linear motion start. This might not give a large performance gain for continuous paths, since it would have the biggest impact when there are sudden, larger jumps in the desired position.

5.1.3 Thrust Allocation

Due to the shape of the uDrone and the placement of the thrusters, distributing the desired forces to the different thrusters is difficult. The thrust allocation implemented on the uDrone is based on geometry, then manually tuned. Although it works reasonably, forces in the x-direction does tend to send the uDrone downwards, forces in the y-direction affect the heading, and so on.

All these coupled forces makes it difficult to get the best positioning, since correcting one direction, throws another one off.

Improving, and possibly making a non-linear thrust allocation algorithm would help both manual and automatic control become more accurate.

5.1.4 Smooth Path

As seen in figure 4.8, abrupt changes in the path direction creates a lot of oscillation. One solution to this problem might be to impose a maximum rate of change on the path, resulting in a path similar to the conceptual sketch in figure 5.2.

This could however take the operator by surprise, resulting in a collision, so the behavior would have to be predictable and graphically represented to the operator.

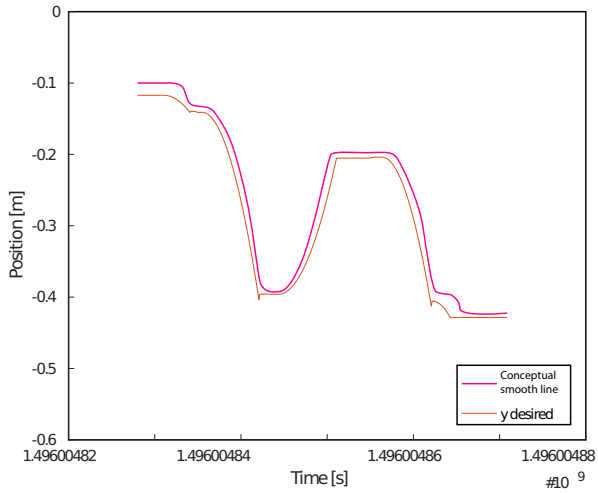

Figure 5.2: Conceptual sketch of the smoothing of the path

The sharp corners of the path should disappear when heading is corrected, so this may not be a necessary step.

5.1.5 Multiple Controllers

As mentioned in section 3.3, some type of adaptive control algorithm might be necessary for an application with such wide span of operational velocities as path-following.

A simple solution would be implementing an adaptive control algorithm, with multiple controllers, and a switching algorithm to switch between the controllers. The switching could be bases on the desired velocity. More advanced multiple-model algorithms, with both multiple observers and controllers, could also be a good solution.

5.1.6 Curvature vs. Velocity

For the tests done during the work with this thesis, it was found that the maximum linear velocity was reached before any maximum curvature was found. This may be due to the tested curvatures not being large enough. Additionally the uDrone is fully activated, with a lot of thrust. This meant that the expected problem of curvature vs. velocity did not manifest itself noticeably.

For an ROV with less thrust capabilities, or when running at higher velocities, this anticipated problem might show up.

5.2 Conclusion

In chapter 2, a solution for the graphical representation of the path was proposed. The proposed solution is highly functional, however the visual aspect of the implementation needs some work. This is important for the hobby and non-professional markets.

This chapter also presented the laboratory facilities, the testbed uDrone, and the systems used for the testing.

Chapter 3 presented the equations for path-generation. It also introduces the maneuvering problem as the method used to solve the path-following problem in this project. The solution to the geometric and dynamic tasks are presented. Then it goes on to present the results from experiments conducted to explore the stability and viability of the path-following control mode.

SPGF was introduced in chapter 4, and the necessary adaptions to the path-following algorithm were presented. The chapter goes on to present the results of the experiments in the laboratory, and discuss the findings. It was found that SPGF does work. However, it was discovered the it is sensitive to sudden changes in path direction, in other words that the derivative of the path needs to be smooth. It was also discovered that a different tuning of the controller is necessary for SPGF and path-following, than for stationary DP.

Some tests on tuning were conducted. However, more are needed to get an optimal response.

An error in the code for path initialization was discovered during inspection of the results, and fixed in software. This correction was not tested in the laboratory due to an error with the thruster system of the uDrone upon returning to the lab. Due to time constraints, this could not be corrected. The creation of a continuous, smooth path was impossible with the initialization error. The concept of SPGF was, never the less, shown to be viable with some tuning. Even with the sudden corners in the path, the controller did not become unstable.

It was also discovered that a more effective method of path-generation is needed for smooth operation in SPGF mode, and to prevent discontinuities in the updating of the desired position.

5.3 Recommendations for Further Work

In this section, recommendations for further work are presented. The recommendations are split up into value-categories based on the amount of time and effort needed versus the gain of implementing them.

5.3.1 High Value

The improvements suggested in this section should give a high performance gain, with little effort required.

- Run test with the fixed heading initialization
- Tuning the controller and observer for path-following
- New algorithm for path-generation as mentioned in section 4.3.2
- Implementing a guidance law

5.3.2 Medium Value

The further work suggested here requires some more time and effort.

- Acceleration and deceleration of desired velocity
- Study the effects of control system frequency on system response
- Implement a battery management system for the uDrone to avoid damage to batteries

5.3.3 Low Value

The suggestions following are more in the class of "nice-to-have" rather than necessary improvements

- Desired heading normal to the path, for object circling and inspection
- Curvature κ as a function of s, allowing for more flexible paths
- Improve the uDrone code base, eliminating performance drains and clutter

Appendix A

HMI source code

The source code for the HMI implementation is attached in the archive file *Attachments.zip*, in the folder *Touch-HMI*.

This file contains all the source code for the entire HMI, not only that of this thesis. This is because the changes are intertwined with the pre-existing code.

Appendix B

Control system source code

The source code for the implemented control system is attached in the file *Attachments.zip*, in the folder *SourceCodeDrone*.

This archive file contains only the files wholly written for this thesis.

Bibliography

BlueRobotics (2017). Underwater gps, Online. Accessed 30.05.2017.
URL: *http://www.bluerobotics.com/store/electronics/underwater-gps/aps-wl-11001/*

Blueye (2017). Blueye, Online. Accessed 06.05.2017.
URL: *http://blueye.no*

Danchilla, B. (2012). *Beginning WebGL for HTML5*, The expert's voice in Web development Beginning WebGL for HTML5.

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

Heyn, H.-M. et al. (2016). Marine cybernetics laboratory handbook.

KumarRobotics (2017). Qualisys ros package, Online. Accessed 03.04.2017.
URL: *https://github.com/KumarRobotics/qualisys*

Qualisys (2015). *Qualisys: Marine applications*.

ROS-community (2016). Ros documentation. Accessed 16.11.2016.
URL: *http://wiki.ros.org/*

Sandøy, S. S. (2016). *System Identification and State Estimation for ROV uDrone*, NTNU.

Skjetne, R. (2005). *The Maneuvering Problem*, PhD thesis, NTNU.

Skjetne, R. (2017). Work note: Rov control modes and functions. Unpublished.

Solstad, T. E. (2016). *Improved user-experience for control of rovs*, Master's thesis, NTNU.

Øyvind Stavdahl (2016). Systemutvikling i henhold til "v-modellen", ttk4235 tilpassede datasystemer.