Marcus Aleksander Engebretsen
Kjetil Skogstrand Gjerden
Øystein Barth Utbjoe
Andreas Våge

Autonomous Navigation, Mapping, and Exploration for Underwater Robots

Master's thesis in Cybernetics and Robotics
Supervisor:
Annette Stahl
Edmund Førland Brekke
Kristin Ytterstad Pettersen
Marco Leonardi
Pål Liljebäck
June 2019

Marcus Aleksander Engebretsen
Kjetil Skogstrand Gjerden
Øystein Barth Utbjoe
Andreas Våge

Autonomous Navigation, Mapping, and Exploration for Underwater Robots

Master's thesis in Cybernetics and Robotics
Supervisor:
Annette Stahl
Edmund Førland Brekke
Kristin Ytterstad Pettersen
Marco Leonardi
Pål Liljebäck
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Abstract

This thesis proposes a system architecture for an autonomous system capable of context-based reasoning when performing autonomous mapping and inspection tasks with unmanned underwater vehicles. To simplify design and implementation, the system architecture was divided into five submodules: hardware, simultaneous localization and mapping (SLAM), classification, path planning, and control. Development of these modules went through four main phases: performance specification, design, implementation, and testing. Consequently, each submodule was verified, both individually and in pairs, to weed out bugs and weaknesses that could impair the performance of the complete system.

A custom stereo-visual-inertial sensor was constructed to allow for precise testing of state-of-the-art visual SLAM (VSLAM) methods underwater. Through the integration of a VSLAM method and an advanced object detection algorithm, semantic maps were generated for increased context-awareness in unknown environments. Using an informed, asymptotically optimal sampling-based approach, the planning module produced feasible, collision-free paths consistent with the C^2 continuity constraint imposed by the control system. By combining the planning module with an information-driven frontier exploration strategy, an autonomous exploration method was implemented. The control module consisted of a uniform global asymptotically stable cascaded guidance and kinematic control algorithm, which achieved full body convergence on the 3D curves provided by the planning module.

To solve the specific case of autonomous pipe inspection with an underwater swimming manipulator, the classification and planning modules were designed to plan routes based on semantic environment information, which, combined with the exploration module, could handle cases where the pipe was lost.

Sammendrag

Denne masteroppgaven legger frem, og implementerer, en arkitektur for et autonomt system som er i stand til kontektsbasert resonnering under inspeksjonsoppdrag utført med ubemannede undervannsfartøy. For å forenkle design- og implementasjonsprosessen har arkitekturen blitt delt inn i fem undermoduler: maskinvare, simultan lokalisering og kartlegging (SLAM), klassifisering, planlegging og kontroll. Utvikling av disse modulene undergikk fire hovedfaser: ytelsesspesifikasjon, design, implementasjon og testing. Som følge av dette har hver av undermodulene blitt verifisert, både individuelt og kombinert med hverandre, for å luke ut software bugs og svakheter som kunne forringe totalsystemets ytelse.

En tilpasset stereovisuell-treghetssensor ble konstruert slik at presis undervannstesting av toppmoderne visuelle SLAM-metoder (VSLAM) kunne gjennomføres. Ved å kombinere en moderne VSLAM-metode og en avansert objektgjenkjenningsalgoritme, ble semantiske kart generert for å gi økt kontekstbevissthet i uutforskede omgivelser. Gjennom bruk av en informert, asymptotisk optimal samplingbasert metode, produserer planleggingsmodulen gjennomførbare, kollisjonsfrie baner som er i overensstemmelse med C^2-kontinuitetsrestriksjonene pålagt av kontrollsystemet. Ved å kombinere planleggingsmodulen med en informasjonsdrevet randbasert utforskningsmetode, ble et system for autonom utforskelse av ukjente omgivelser implementert. Kontrollmodulen bestod av et uniformt globalt asymptotisk stabilt kaskadebasert styringssystem som oppnådde full konvergens på 3D-kurver forsynt fra planleggingsmodulen.

For å løse det spesifikke problemet med autonom rørinspeksjon med en svømmende undervannsmanipulator ble klassifisering og planleggingsmodulene designet for å kombinert kunne planlegge ruter basert på semantisk informasjon om omgivelsene som, sammen med utforskningsmodulen, kunne håndtere situasjoner hvor sporet av røret gikk tapt under operasjonen.

Preface

This thesis is submitted to the faculty of Information Technology and Electrical Engineering at the Norwegian University of Science and Technology in partial fulfilment of the requirements for the degree of Master of Science in the Department of Engineering Cybernetics.

The times we find ourselves in are heavily influenced by robotics making an entrance in all parts of society, and the maritime industry is no different. A common factor for the authors is an interest in robotics and the different challenges such complex systems presents. When the opportunity to work with Eelume AS - which develops snake robots for the subsea industry - presented itself, we were delighted to get a chance to work on such a unique robotic: the underwater swimming manipulator (USM). Using this robot as a focus point allowed us to draw inspiration for specific operational problematics, while also supporting our vision of a general autonomy framework. Developing such a system provides a multitude of challenges that cover a broad spectrum of research areas, with each sparking the individual authors' interests. These central themes include simultaneous location and mapping (SLAM), classification, path planning, and guidance and control. Our team of supervisors consisted of Annette Stahl, Edmund F. Brekke, Kristin Y. Pettersen, Marco Leonardi, and Pål Liljebäck; your combined knowledge helped us navigate through the fog of academia, thank you!

Due to this being a collaboration thesis, each of the supervisors acted as the main supervisor within their respective area. Annette Stahl helped with the development of the classification system, Edmund F. Brekke aided with the SLAM algorithm and sensor calibration, Marco Leonardi assisted with the planning and exploration modules as well as the SLAM module, and Kristin Y. Pettersen oversaw the guidance and control system. Additional thanks to Ida L. Borlaug, Marianna W. Kaminska, and Mathias H. Arbo, for feedback on the control system design.

To further assist our work, we were provided with certain gadgets and software, as well as economic support from the Department of Engineering Cybernetics at NTNU. Through cooperation with Eelume, we were lucky enough to be given access to testing facilities and equipment, not to mention the possibility to perform full-scale testing with on a USM. Additionally, an in-house simulator was made available to simulate the interface of the physical USM.

Seeing as the SLAM and classification modules are dependent on sensor input to do much of anything, a high-quality sensor rig was constructed. This sensor setup consists of a stereo camera, an IMU, and a pressure sensor, most of which were acquired through department funding. Additionally, in order to handle the computational efficiency the neural networks require, we were provided a Jetsson TX2 developer kit.

Finally, it is worth mentioning that a few parts of this thesis is inspired by some of the authors' previous works completed during the specialization project. Much of the background theory, as well as the introductory literature review sections, in part V is inspired by the work from TTK4550 [1] and is used as a stepping stone for the implementation of the planning and exploration modules performed in this thesis. This is also the case for some of the sections in part VI, based on the work by M. A. Engebretsen. In part IV some of the work from [2] have been used. The choice of SLAM algorithm, as well as background theory in part III, is inspired by the work from TTK4550 [3].

Contents

List of Tables

Nomenclature

${}_m\mathcal{P}$	SLAM map represented as a 3D point cloud relative to frame m (see chapter 7)
$\mathbf{F}_i$	Coordinate frame i (see chapter 2)
$\eta_{b/I}$	Robot pose of frame b with respect to frame I (see chapter 2)
$\vec{v}_{b/I}$	Linear velocity of frame b with respect to frame I (see chapter 2)
$\vec{\omega}_{b/I}$	Angular velocity of frame b with respect to frame I (see chapter 2)
$\vec{a}_{b/I}$	Linear acceleration of frame b with respect to frame I (see chapter 4)
$\mathcal{A}$	space occupied by the robot or vehicle (see chapter 11)
$\mathcal{C}$	configuration space (see chapter 11)
$\mathcal{C}_{\text{free}}$	free configuration space (chapter 11)
$\mathcal{C}_{\text{obst}}$	obstacle-occupied configuration space (chapter 11)
$\mathcal{O}$	space occupied by obstacles (chapter 11)
$\mathcal{U}$	the set of all possible actions (chapter 11)
π	path (section 11.1)
ϖ	path parameter (section 11.1)
$\mathcal{W}$	robot workspace (chapter 11)

Part I

Introduction

1 | Motivation

Autonomous robots have seen considerable advancement in recent years due to the escalating dependency of increasingly complex autonomous systems. An autonomous system is a system capable of performing tasks with little to no human interaction by doing intelligent reasoning based on sensory information. Such systems are then able to fulfil tasks unaided, even with loosely defined goals in dynamic or unknown environments. This results in a wide range of commercial, military and scientific applications, including underwater inspections [4], [5], oceanographic mapping and pipeline inspection [6]; planetary explorers [7], [8]; and aerial drone inspection [9].

To facilitate these kinds of autonomous operations, the robot itself needs to be able to perceive its environment and from that information plan actions to fulfil a goal. Good decision making requires sufficient knowledge of the environment, or *workspace*, the robot operates in. With state-of-the-art sensor technology, more robust perception systems are ever available, allowing progressively more potent motion estimation, tracking and localisation, even in underwater environments — albeit still challenging, particularly with underwater visual sensors [10]–[12].

Environments that require large effort and cost for human operators to reach are typical *harsh underwater environments*. In these situations, it is favorable to deploy an autonomous robot permanently on location. This is one of the main objectives of the *underwater swimming manipulator* (USM) [13], allowing for autonomous inspection, maintenance and repair (IMR) operations and thus reducing the need for costly IMR ship-based operations. USMs benefit from being highly actuated, as well as from their shape-changing capabilities and slim build, which facilitates operations in confined and unstructured spaces, contrasting more conventional remotely operated vehicles (ROVs).

While beneficial, energy- and cost-wise, deployment of these types of submarine robots results in strict robustness requirements. Possible tasks for such systems include autonomous submarine pipe inspection based on visual sensors and object detection, while simultaneously mapping the surroundings. Therefore, a complete system with well interconnected subsystems is vital, including subsystems such as computer vision, simultaneous localization and planning (SLAM), path planning and control. Much of modern research tend to focus singularly on a specific subsystem, e.g. concentrating on SLAM or control separately. In some cases, however, it is advantageous to look into creating a more combined system. On this front, modern literature and research is scarce, and this task will therefore be the main focus of this thesis; creating a system pipeline from perception to mapping, planning, and control.

1.1 Project Statement

The objective of this project is to implement underwater SLAM algorithms based on optical cameras and using the SLAM map to plan missions that are then to be executed. The aim is to implement this on an underwater robot, where an Eelume robot is considered as the target platform. The Eelume vehicle is a particularly relevant platform for SLAM since its long body allows for a good spatial distribution of the cameras and lights used for mapping, thereby enabling cameras and lights from different angles. The overall objective is to enable the robot to develop a 3D map of its environment (such as the seabed, structures on the seabed, or floating structures), to enable the robot to continuously determine its own location as it moves in the mapped environment and use this information to plan and execute its mission. The project also wants to provide context to the map through computer vision and/or machine learning techniques. The following are some examples of what a mission may look like:

- station keeping.
- following a pipeline on the seabed.
- return to base station.

The main objective of this thesis is to implement a system capable of performing autonomous context-based inspection tasks based on optical sensors, while simultaneously developing a 3D map of the environment to enable further mission planning and execution. The aim is to implement this on an underwater robot, where an Eelume robot is considered as the target platform. The Eelume vehicle is a particularly relevant platform for SLAM, since its long body allows for a good spatial distribution of the cameras and lights used for mapping, thereby enabling cameras and lights from different angles. This will be accomplished through the implementation of an object detection network, a VSLAM algorithm for underwater operation, a path planning module to provide flyable 3D paths, an exploration module to guide autonomous environment exploration, and a cascaded guidance and control system, as well as by proposing an overall system architecture and a hierarchical state machine to keep track of the system status.

1.2 Contributions

This thesis is a collaboration between four students and does, therefore, cover a lot of different concepts within robotics. To summarise this work, the main contributions are listed below.

- Design of a system architecture for a UUV capable of performing autonomous navigation, mapping, and inspection tasks.
- Development of a hierarchical state machine to keep track of the overall state of the total system.
- Development of an open source C++17 IMU communication driver.
- Design of a custom calibration boards for underwater camera calibration.
- Construction of a potent stereo-visual-inertial (stereo-VI) sensor capable of underwater operation.
- Gathering several datasets for testing underwater SLAM systems.
- Improving upon existing open source SLAM methods.
- Creation and labeling of a dataset to be used for training of neural networks for underwater operations.
- Presenting a strategy for making the neural network increasingly robust over time.
- Implementation of a planning and exploration system capable of exhaustive environment exploration in unknown three-dimensional environments.
- Implementation of a cascaded, uniformly globally asymptotically stable (UGAS) guidance and control system.
- Implementation of a C++11 simulation framework to verify correctness of the USM kinematic control system.
- Integrate a open source SLAM method and classification network to generate a semantic map.
- Several suggestions regarding future work to increase the robustness of the system, as well as to achieve better operational performance.

1.3 Thesis Outline

This thesis is a collaboration of four students, and does, therefore, cover a wide array of topics. To ease the reading flow and make for easier look-up, the text is divided into several parts, each detailing one of the main aspects of the project. These parts are structured as follows.

Part I introduces the main motivation behind this thesis as well as the essential preliminaries in chapter 2. The first part is concluded with chapter 3, giving an overview of the overarching architectural choices present in the system as a whole and its sub-modules, and provides a brief description of the workflow organization for this project. In part II, the work, theory, and design behind the applied hardware setup is presented. The theory, implementation, and calibration details of the sensors used is treated in chapter 4. Chapter 5 explains the electronic designs behind the hardware setup, whereas chapter 6 deals with the mechanical design. Part III discusses the SLAM module. In chapter 7, the theory behind SLAM is introduced. Then chapter 9 presents the experiments conducted and their results. The work regarding classification and object detection is explained in part IV, with chapter 10 presenting the theory behind neural networks for classification, as well as the specifics regarding dataset creation, network training, and results. Part V discusses the planning problem in 3D as well as strategies for autonomous exploration. In chapter 11, a theoretical planning framework is introduced, before different planning methods, and their related literature, are discussed in chapter 12. Chapter 13 treats the problem of autonomous environment exploration and examines different exploration strategies. This part is concluded with a description of the planning and exploration method, as well as their evaluation through simulation tests. In part VI, aspects regarding kinematic control of the USM are treated and a UGAS control law are described in chapter 15. Performance specification, design, experiments, and results for each of the respective modules are presented in their corresponding parts. However, part VII discusses some of the module connections in more detail, with respect to interfacing and information flow in real-life inspired situations through specific simulation cases. Simulations and results of these cases are presented in chapter 16, before the thesis as a whole is concluded in chapter 17.

2 | Preliminaries

This chapter introduces notation that will be used across different parts and chapters in this thesis. It is not uncommon that similar concepts are represented differently across various disciplines. As this is the case for computer vision, SLAM, path planning, and control systems theory, a coherent set of definitions are introduced. This chapter is intended to be a look-up chapter and will, therefore, be referenced extensively.

2.1 Geometry

To represent transformations between rigid bodies in 3D, Lie algebra is used, see e.g. [14]. The notation follows a combination of the marine systems standard from [15] and the USM literature, see e.g. [16].

2.1.1 Homogeneous Transformations

The homogeneous transformation between two frames of reference can be defined as follows

Definition 2.1. A homogeneous transformation from frame F_j to F_i will be represented by

$$\mathbf{T}_j^i = \begin{bmatrix} \mathbf{R}_j^i & \eta_{i/j,1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} , \; \mathbf{T} : \mathbb{R}^3 \longrightarrow \mathbb{R}^3 \, , \tag{2.1}$$

with inverse

$$\mathbf{T}_i^j = (\mathbf{T}_j^i)^{-1} = \begin{bmatrix} \mathbf{R}_i^j & -\mathbf{R}_i^j \eta_{i/j,1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} , \tag{2.2}$$

where $\mathbf{R}_j^i \in \mathbf{SO}(3)$ is a 3D rotation matrix, and $\eta_{i/j,1} \in \mathbb{R}^3$ defines the position of the body frame F_b relative to the inertial frame F_I.

Remark 2.1.1. For all practical purposes $\mathbf{T}_j^i \in \mathbf{SE}(3)$, i.e. $\mathbf{T}_j^i$ is a member of the group of rigid body transformations on $\mathbb{R}^3$.

Remark 2.1.2. The origin of F_i with respect to F_j is denoted $\eta_{i/j,1}$

A useful result is that the composition of multiple homogeneous transformations is

$$\mathbf{T}_j^i = \mathbf{T}_j^{(j+1)} \mathbf{T}_{(j+1)}^{(j+2)} \cdots \mathbf{T}_{(i-1)}^i \; . \tag{2.3}$$

The definitions presented here assume $\mathbf{R}_j^i = \mathbf{R}_j^i(\phi, \theta, \psi)$ to be any composite 3D rotation matrix, and operates on $\mathbf{SE}(3)$ which denotes the *Special Euclidean Group* of dimension three, defined by [17]:

$$\mathrm{SE}(3) := \left\{ \mathbf{T}_j^i \mid \mathbf{T}_j^i = \begin{bmatrix} \mathbf{R}_j^i(\phi, \theta, \psi) & \eta_{i/j,1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix},\ \mathbf{R}_j^i \in \mathrm{SO}(3),\ \eta_{i/j,1} \in \mathbb{R}^3 \right\}, \tag{2.4}$$

which defines the group of all 4×4 dimensional homogeneous transformation matrices.

2.1.2 D-H Convention

Following the framework defined in [18], the translations and rotations between the links of an underwater multi-body robot can be described using the D-H convention, see e.g. [19] for a description.

Definition 2.2. Assuming only revolute joints, the homogeneous transformation matrix becomes

$$\mathbf{T}_i^{(i+1)}(q_i) = \begin{bmatrix} s(q_i) & -s(q_i)c(\alpha_i) & s(q_i)s(\alpha_i) & a_i c(q_i) \\ s(q_i) & c(q_i)c(\alpha_i) & -c(q_i)s(\alpha_i) & a_i \sin(q_i) \\ 0 & s(\alpha_i) & c(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{2.5}$$

where $s(\cdot)$ refers to $\sin(\cdot)$.

The formulation in definition 2.2 simplifies the implementation between links of the robot considerably, making it a suitable choice for this project.

2.1.3 Euler Convention

To represent the orientation of the robot relative to the inertial frame, Euler angles are used. This is to simplify debugging, as they are *practically* much easier to comprehend than i.e. quaternions. The ZYX convention was used, giving a representation of the form

$$\mathbf{T}_I^b = \begin{bmatrix} c(\psi_{b/I})c(\theta_{b/I}) & -s(\psi_{b/I})c(\phi_{b/I})+c(\psi_{b/I})s(\theta_{b/I})s(\phi_{b/I}) & s(\psi_{b/I})s(\phi_{b/I})+c(\psi_{b/I})c(\phi_{b/I})s(\theta_{b/I}) \\ s(\psi_{b/I})c(\theta_{b/I}) & c(\psi_{b/I})c(\phi_{b/I})+s(\psi_{b/I})s(\theta_{b/I})s(\phi_{b/I}) & -c(\psi_{b/I})s(\phi_{b/I})+s(\psi_{b/I})c(\phi_{b/I})s(\theta_{b/I}) \\ -s(\theta_{b/I}) & c(\theta_{b/I})s(\phi_{b/I}) & c(\theta_{b/I})c(\phi_{b/I}) \end{bmatrix}. \tag{2.6}$$

Definition 2.3. F_I denotes the inertial frame.

Definition 2.4. $\eta_{b/I,2} = \begin{bmatrix} \phi_{b/I} & \theta_{b/I} & \phi_{b/I} \end{bmatrix}^T$ represents roll, pitch and yaw angles, respectively, of the body frame, F_b, with respect to F_I.

It should be noted that the Euler angles introduce a singularity when mapping from the body velocities of the rigid body to the time derivative of the position variables. From chapter 3.3 in [14] it is derived that

$$
\begin{aligned}
\dot{\eta}_{b/I} &= \mathbf{J}_b(\eta_{b/I,2})\vec{v}_{b/I} \\
&= \begin{bmatrix} \mathbf{R}_{b/I}(\eta_{b/I,2}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{J}_{b/I,\mathrm{rot}}(\eta_{b/I,2}) \end{bmatrix} \vec{v}_{b/I} \\
&= \begin{bmatrix}
c(\psi_{b/I})c(\theta_{b/I}) & -s(\psi_{b/I})c(\phi_{b/I})+c(\psi_{b/I})s(\theta_{b/I})s(\phi_{b/I}) & s(\psi_{b/I})s(\phi_{b/I})+c(\psi_{b/I})c(\phi_{b/I})s(\theta_{b/I}) & 0 & 0 & 0 \\
s(\psi_{b/I})c(\theta_{b/I}) & c(\psi_{b/I})c(\phi_{b/I})+s(\psi_{b/I})s(\theta_{b/I})s(\phi_{b/I}) & -c(\psi_{b/I})s(\phi_{b/I})+s(\psi_{b/I})c(\phi_{b/I})s(\theta_{b/I}) & 0 & 0 & 0 \\
-s(\theta_{b/I}) & c(\theta_{b/I})s(\phi_{b/I}) & c(\theta_{b/I})c(\phi_{b/I}) & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & \frac{s(\phi_{b/I})s(\theta_{b/I})}{c(\theta_{b/I})} & \frac{c(\phi_{b/I})s(\theta_{b/I})}{c(\theta_{b/I})} \\
0 & 0 & 0 & 0 & c(\phi_{b/I}) & s(\phi_{b/I}) \\
0 & 0 & 0 & 0 & \frac{s(\phi_{b/I})}{c(\theta_{b/I})} & \frac{c(\phi_{b/I})}{c(\theta_{b/I})}
\end{bmatrix}
\begin{bmatrix} u_{b/I} \\ v_{b/I} \\ w_{b/I} \\ p_{b/I} \\ q_{b/I} \\ r_{b/I} \end{bmatrix}
\end{aligned}
\tag{2.7}
$$

.

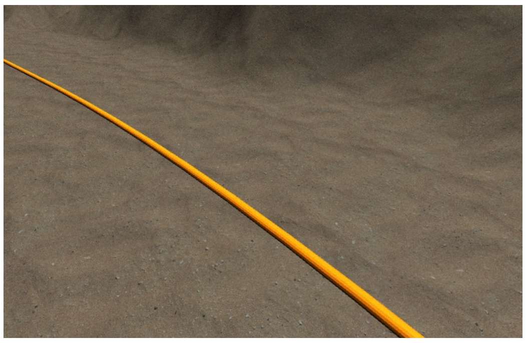
Definition 2.5. $\eta_{b/I} = \begin{bmatrix} \eta_{b/I,1}^T & \eta_{b/I,2}^T \end{bmatrix}^T = \begin{bmatrix} x_{b/I} & y_{b/I} & z_{b/I} & \phi_{b/I} & \theta_{b/I} & \psi_{b/I} \end{bmatrix}^T$ is the base link pose.

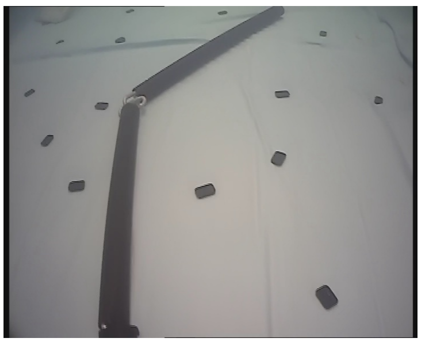
Definition 2.6. $\vec{v}_{b/I} = \begin{bmatrix} u_{b/I} & v_{b/I} & w_{b/I} \end{bmatrix}^T, \vec{\omega}_{b/I} = \begin{bmatrix} p_{b/I} & q_{b/I} & r_{b/I} \end{bmatrix}^T$ are the linear and angular velocities of F_b with respect to F_I.

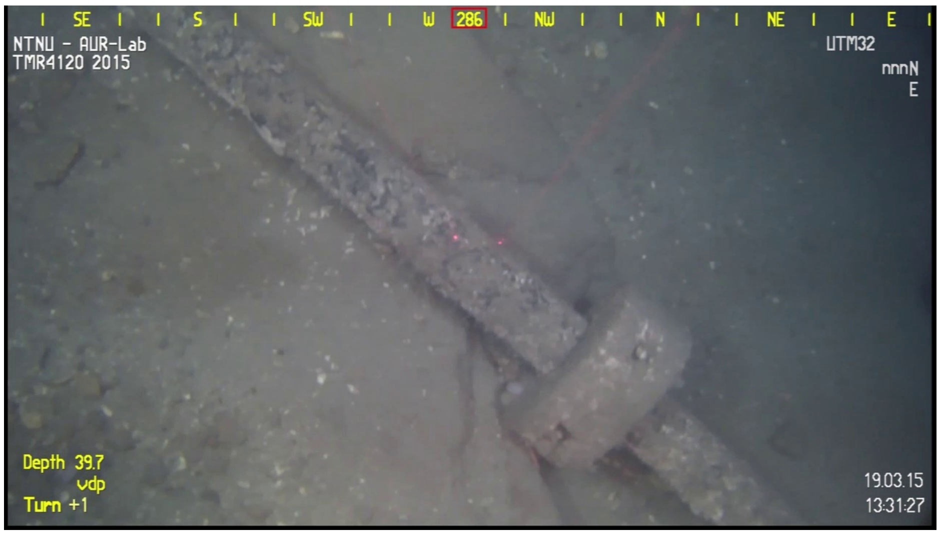
2.2 Operating Conditions

In this project there are three main conditions that are relevant for the system to operate in. The first is a simulated environment that have been used to test the path planning system. Figure 2.1a shows example conditions from the simulator. The simulator used is a UUV simulator [20] based on Gazebo. This simulator includes the equations of motion for underwater vehicles [15] and incorporates lift, drag, and current simulations, as well as a range of sensor plugins. A closer description of the simulator is given in appendix A.

Although the ground is textured, the simulated condition lacks many of the complexities that are present in underwater conditions — especially when it comes to how it looks through the camera. The environment is highly controlled, the visibility is very good — much better than can be expected from underwater condition — and the pipe is the only object of a known colour. In short, the simulated conditions are not representative of real


(a) Example simulated camera data.


(b) Image from the test setup.



(c) Example footage from an inspection mission.

Figure 2.1: Examples of each of the three main conditions relevant for this thesis.

operating conditions. Even though the simulated conditions deviate substantially from the real conditions, it can be a valuable platform to test and draw information on the performance of the system, so long as one is aware of its limitations.

To ensure that the system perform in an underwater environment, the second setting is introduced. The second setting is the testing facilities at Eelume AS as can be seen in fig. 2.1b. These conditions are a leap in complexity when compared to the simulated conditions. Complexities such as light fracturing as the light move from the water to the air in the camera housing are present. However, the conditions are highly controlled and not sufficiently complex to be comparable with a real operations.

To have an idea of the performance of the system in actual operating conditions, a third setting is introduced. This third scenario is comprised of video footage [1] from an inspection operation. In operating conditions, there are several factors contributing to making this scenario significantly more challenging. The biggest difference between the operating and test situations is, perhaps, the fact that the pool background is light blue,

[1]This footage was provided by Marco Leonardi and the autonomy group at NTNU

which sharply contrasts the pipe. This renders the detection problem in the lab nigh on trivial when compared to the actual operating conditions. Other differences are also present, among them are the lack of particles in the water that produce a backscattering effect and interference in the image. In operating conditions, substantial biofouling is usually present on any underwater structure [21]. This generally makes underwater objects harder to detect, compared to many terrestrial scenarios, as how obscured the object is can vary drastically. Examples of each of these conditions is shown in fig. 2.1.

3 | System Overview

To achieve the main goal of developing an interconnected autonomous pipeline solving the problem described in section 1.1, the required hardware and software have to be defined. To construct a scaleable architecture, the specific choice of components are not exclusively tailored to the problem at hand, even though they solve it. Consequently, this chapter presents an overall autonomous architecture that enables a USM to navigate and explore its environment (section 3.1). Additionally, the specific architectural choices towards underwater pipe following and exploration are specified to highlight the scalability. A hierarchical state machine controlling the overall state of the USM is subsequently described in section 3.2 to provide an overview of the control flow in the autonomous architecture. Finally, the collaboration tools are presented in section 3.3.

3.1 Architecture

The autonomous architecture is defined as all software and hardware enabling autonomous operation, see fig. 3.1. The most important hardware components used in this thesis includes a stereo camera setup, an IMU, and the magnetic angular encoders on the revolute joints. The stereo camera provides 2D images to the classification and SLAM modules to map environmental features and perform localization. The IMU and rotary encoders supply additional introceptive data to the SLAM module to enhance its performance. The overall software system design follows the perceive-think-act principle. Sensory input is decoded and contextualized to ensure that the robot knows its whereabouts and surroundings. With semantic understanding of the sensory input, the path planning algorithm ensures that the USM *wants* to move in a desirable direction depending on the task at hand. Finally, the control algorithm ensures that the USM follows this planned *path*.

3.1.1 Perception

The perception pipeline begins with synchronized sensor data acquisition where a stereo camera and an IMU synchronously sample measurements and transfer them to the SLAM algorithm. To achieve this, and to allow for easy independent testing of the perception system, a custom underwater stereo-VI sensor was built, including a micro-controller for sensor synchronization.

At the core of the perception system lies the SLAM algorithm. From the sensor mea-

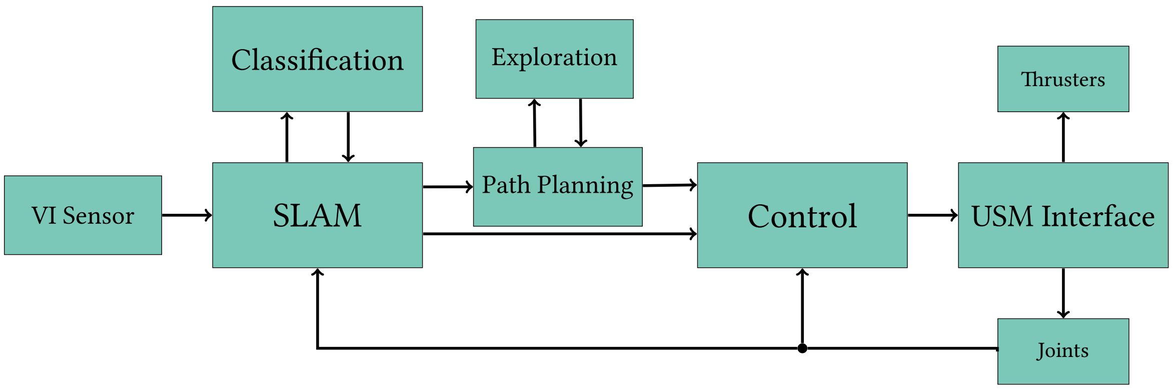


Figure 3.1: Block diagram of the overall autonomous architecture.

surements, it simultaneously creates a map of the observed environment and localizes itself in that map by continuously setting up and solving a nonlinear optimization problem. It is an advanced, yet general, algorithm which can incorporate information from any type of sensor or model prediction. The map is represented as a point cloud, where each point has a 3D position found by matching and triangulating *oriented FAST and rotated BRIEF* (ORB) features [22] recognized in multiple images over time.

In order to interpret the environment this thesis will look into state-of-the-art solution that can be used to finding objects in an image. The focus of this thesis was pipe detection, but the proposed solution should have the capability to be expanded to detect multiple other objects. The map points — the point cloud constituting the SLAM map — will be labeled from this classification system to contextualize the map. The map points will be visible from multiple key frames — frames chosen by the SLAM system that have a good view of multiple map points — and will be classified by subjecting these key frames to the classification system. The map points will be given a labeling score based on what the result from the classification. This score will accumulate over time as the map point is observed from multiple key frames and the map point will be labeled as the class with the highest score.

3.1.2 Planning and Control

The planning module uses an asymptotically optimal informed sampling-based algorithm to calculate optimal paths subject to a given optimization objective — path length in most cases, but this can be extended to take additional objectives into account. Due to the large varieties of underwater vehicles, and their different degree of manoeuvrability, the planning sub-system was implemented with generality in mind, thus ensuring applicability to more kinematically constrained robots. Collision-free paths that satisfy the stated constraints and the optimization objective given the current state of the map, are then passed along to the control system.

To act as a layer above the path planner, an exploration module was implemented. This exploration module is responsible for deciding where to go next based on the current state of the robot, map, and mission objective. These mission objectives include exhaustive environment exploration and specific exploration based on semantic data from classification, among others. Exhaustive exploration refers to the act of mapping a given area until the coverage is maximized, or no more safe locations are left to explore. To achieve specific exploration, this module collaborates with the classification module. The idea behind this is to allow the robot to perform context-driven exploration tasks, such as pipe-following and -inspection in an unknown environment. To carry out this task, semantically labelled map information is shared by the classification to the exploration sub-system, from which new goals that further drive the inspection are calculated and passed to the path planner module.

The basis for the control system is a guidance algorithm which generalizes to any robot operating in 3D. In general, this is the part of the control system interfacing with the path, provided by the path planning module, to ensure that the robot tracks it. The kinematic controller, on the other hand, is specifically tailored to ensure convergence of a *floating base manipulator* to a curved 3D path, i.e. a robot with a base and manipulators attached to that base. Designing a generalizable system with the desired convergence and stability properties would be tedious, and is consequently avoided in the more low-level parts of the control system — meaning all abstraction layers below the guidance controller.

3.2 State Machine

As the autonomous architecture consists of a set of modules, all with a variety of *operating modes*, a state machine is introduced to keep track of the overall state of the USM. At the highest level, the state machine reflects the current task the USM performs. Example tasks include booting, dynamic positioning (DP), and moving (MOVE), see fig. 3.2. To encapsulate the internal behaviour of the software modules, and their interactions, task transitions are triggered by specific *events*. Events represent the state transitions of the underlying hierarchically layered state machine (HSM), see e.g. [23]. With a hierarchical ordering comes the possibility of encapsulating the internal state machines of the main software modules — i.e. SLAM, path planning and control — and at the top layer only handle their influence on the task at hand. See the arrows in fig. 3.2 for an illustration of this concept. This principle is similar to the Liskov substitution principle, see e.g. [24].

The tasks to be performed in this particular case is pipe-following and -inspection. Actually following the pipe is covered by the state MOVE. If the SLAM module loses track, the planning module detects a future collision, or the control system finds the current objective to be unattainable, the USM is requested to stop by entering state DP. Depending

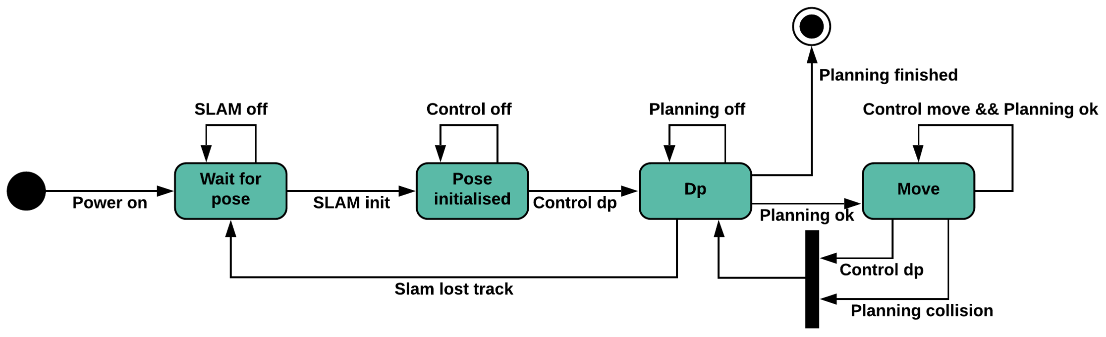


Figure 3.2: State diagram of high-level states of the autonomous architecture. The arrows represent events/state transitions of underlying software module state machine machines triggering a high-level state transition. For clarity, the booting procedure is divided into several steps. Standard UML notation is avoided primarily to promote simplicity.

on the environment and the internal states of the software modules, the pipe following task is either finished, or events are handled at a lower level to re-enter state MOVE.

The design favours modular development of the individual software components as long as they have a clear interface with peripheral software. For example, SLAM broadcasts its internal state such that the rest of the autonomous system knows whether pose and map information can be trusted or not. The explicit content of the lower-level state machines are presented in detail in part III, part V and part VI.

3.3 Organization

In any large project with multiple contributors, it is advantageous to work in a structured manner — e.g. to increase effectiveness and simplify system integration between collaborators. Coding in teams can quickly be challenging, as team members most likely develop on different versions of the same project. If one is not careful, these versions may come into conflict when trying to merge the work into a cohesive project. To solve this, the collaborators decided to use GitHub, an open source versioning tool. This way, each collaborator could work on features in their respective local branches and only merge the functionality into a master branch when ready. By requiring that code merged to the master branch is tested properly, the likelihood that the master branch is ready for operation at any given time increases. Since the project consists of multiple modules, it was decided to structure them in their own respective programs, or packages, within individual GitHub repositories (repos.). The result of this decision was several rub-repositories connected through a master repository, each with their own development history. As a consequence of this partitioning, version testing of the different modules became much simpler and more efficient, although at the cost of a slight increase in repository bookkeeping.

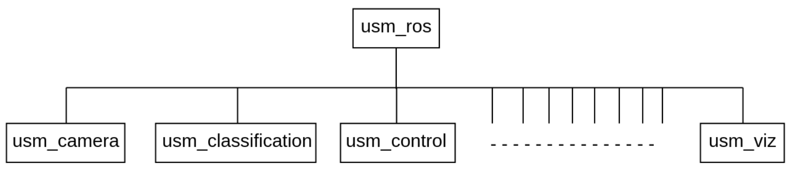
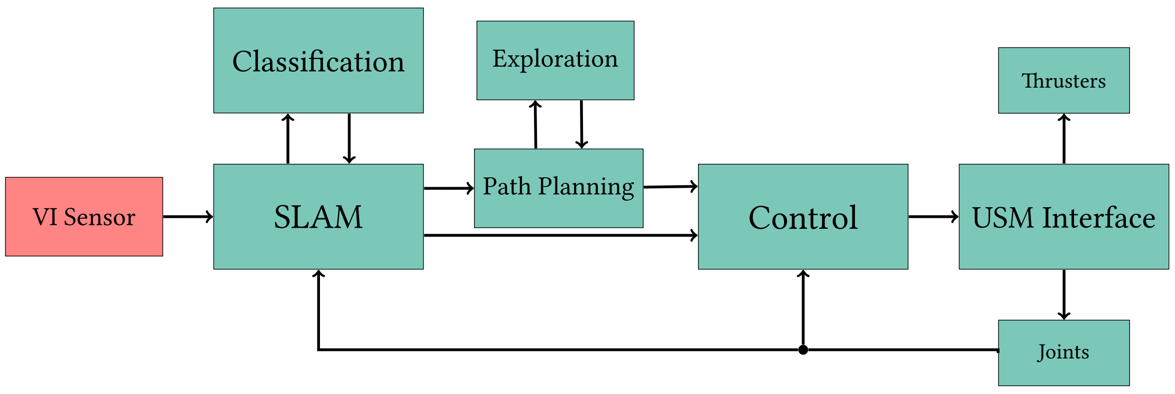


Figure 3.3: Git submodule organization.

To keep the repositories organized, *Git submodules* was used. Git submodules introduce a *root* repo. (*usm_ros* in fig. 3.3), which links up with a set of *leaf* repos. by tracking their versions. The root repository can be updated to track any desired branch of all the leaf repositories. In other words, any set of compatible versions can be maintained simultaneously.

Part II

Hardware



4 | Sensors

When implementing algorithms for real world applications, one of the most important factors for success is a lot of testing. However, underwater robots are expensive, complicated, and at times dangerous for equipment or personell. Therefore, performing a lot of testing is expensive and time consuming because it needs to follow safety protocols and requires trained personnel. Furthermore, start-up companies typically have very limited number of robots available for testing, and the trained personnel are extremely busy. Due to this, it became apparent that a test rig for the perception system(s) needed to be designed and constructed. The primary goal for the test rig was to provide the sensory input needed for testing underwater SLAM algorithms. Additionally, certain design choices were made to make the rig suitable for applications outside of the scope of this thesis. To accomplish this, several criteria needed to be met. Thus, the rig had to:

- Consist of the correct set of sensors for the given use-case.
- Be accurate with respect to each of the sensors' input quantity.
- Have sufficient waterproof rating to allow shallow-water operation.
- Support modular modifications, with the possibility to extend with additional sensors.
- Be small enough to be easily handled by one person.
- Allow for easy mounting onto robots or other objects.

In this chapter, the choice of sensors are briefly discussed before the individual sensors are described, modelled and calibrated. The implementation and synchronization of the sensors (chapter 5), as well as the mechanical design of the test rig (chapter 6), are described and discussed in the two subsequent chapters.

4.1 Hardware Platform Description

A stereo-visual-inertial sensor (stereo-VI) was implemented using two monocular cameras acting as one stereo camera, an IMU, and a pressure sensor. In order to discuss the choice of sensors, a camera is assumed to be the main exteroceptive sensor, i.e. the sensor which measures the external state and is used for mapping the environment — there are other options such as sonars available, but the cost of high quality cameras are comparatively low when compared to sonars One monocular camera does, in theory, provide enough information for a SLAM algorithm to map the environment and localize the 6 DoF pose of the camera *up to scale*. Up to scale means that the algorithm has no notion of the metrics, i.e how many meters it has traveled or how many square meters the map covers. Simply put, it can not detect if it is inside a miniature model of the environment or the actual environment. As the scale is not measured, a common problem is scale drift, where the estimated scale changes over time. Modern monocular visual SLAM algorithms overcome this problem by including the scale in the optimization when performing loop closures. However, this does not work during exploration when the robot is only detecting new areas, thus the scale drift leads to significant overall drift. A stereo camera can estimate the metric 3D position of the pixels relative to the camera, thus the scale is measured directly from each image pair. Similarly, a monocular camera in combination with an IMU can estimate the scale, because the acceleration and angular rate measured by the IMU is metrically correct. From this, the velocity and position can be estimated which in turn gives the scale. One thing to note, is that there have to be some acceleration or rotation for the IMU to give any measurements which can be used to estimate the scale. The same principle exists for the combination of a pressure sensor and a camera. The pressure sensor can estimate the scale directly, but the robot has to move vertically in the water for this to work. By adding all four sensors, the rig supports the testing of multiple sensor combinations, or with the full collection, creating a more robust system.

In addition to this, support for lights and an acoustic underwater positioning system were added. This allows for testing in dark environments, which is required for deep sea exploration, and the ability to generate ground truth position measurements to be included in the SLAM algorithm for increased robustness. However, due to time constrains, neither of these options are tested in this thesis.

4.2 Inertial Measurement Unit

An inertial measurement unit (IMU) typically consists of a accelerometer and a angular rate sensor. The accelerometer measures proper acceleration, that is acceleration in its own instantaneous rest frame. That is the frame where the accelerometer is receiving no external forces. For example, if the accelerometer is laying still on a table, one might think it would measure zero, but in fact it measures 9.81 m/s^2. This is due to the contact forces of the table acting on the accelerometer, opposing the gravity and keeping it at rest in our inertial frame. Thus to get the acceleration relative to the inertial frame, one would have estimate the gravity and subtract it from the measurements. One could also use the estimated gravity to determine the orientation of the accelerometer. The angular rate sensor measures the angular rate of the sensor relative to a inertial frame. The NED frame is used as the inertial

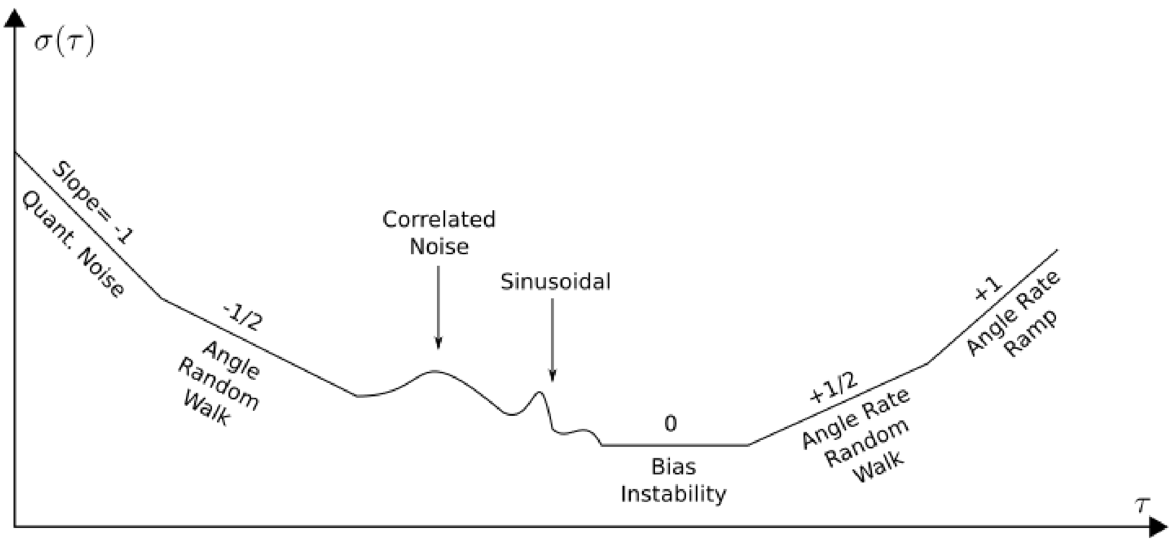


Figure 4.1: Allan (standard) Deviation plot recreated from [26]. Under the assumption that noise sources are independent, one Allan Deviation Plot can be used to estimate several sources of noise.

frame, which is, strictly speaking, incorrect as the NED frame is fixed on earth, while the earth is rotating. However, for low-speed applications on the surface, this is a common and justified simplification [15].

4.2.1 Sensor Model

The IMU is modeled using a linear model, as in [25], with two error sources:

- $\gamma(t)$ - a random component modelled as white noise.
- $\mathbf{b}(t)$ - a slowly varying bias modelled as a Brownian motion process.

$$\tilde{\boldsymbol{\omega}}^s_{\text{imu}}(t) = \boldsymbol{\omega}^s_{s/n}(t) + \mathbf{b}^s_{\text{gyro}}(t) + \boldsymbol{\gamma}^s_{\text{gyro}} \tag{4.1}$$

$$\tilde{\mathbf{a}}^s_{\text{imu}}(t) = \mathrm{R}^s_n(\theta)\left(\dot{\mathbf{v}}^n_{s/n}(t) - \mathbf{g}^n\right) + \mathbf{b}^s_{\text{acc}}(t) + \boldsymbol{\gamma}^s_{acc}(t) \tag{4.2}$$

$$\dot{\mathbf{s}}^s(t) = \mathbf{w}^s(t)\ , \tag{4.3}$$

where s represents the sensor frame, n represents the North-East-Down (NED) frame, and $\gamma(t)$ and $\mathbf{w}(t)$ represent white noise with standard deviation σ_γ and σ_w respectively. For simplicity we assume the measurement frame of the IMU is equal to the body frame.

It is not trivial to estimate the values of the standard deviation σ_γ and the bias standard deviation σ_w needed for our sensor model as they are difficult to tell apart. The most used approach is to do Allan Variance experiments. Allan Variance is a time domain analysis technique and can be applied to any signal to determine the character of the underlying noise process. The reader is referred to "An introduction to inertial navigation" [27] and "An overview of the Allan variance method of IFOG noise analysis" in appendix C of [26] for a complete definition. The Allan Variance, AVAR(τ), is basically the variance of the signal as a function of averaging time. Figure 4.1 displays an example of a square root Allan Variance plot, also called Allan Deviation AD(τ). For small averaging times, AD(τ) is dominated by the random noise component of the signal. As the averaging time increases the effect from the random noise decreases. While the effect from the slowly varying bias remains. The bias instability parameter, $\tilde{\sigma}_{\text{instability}}$, is the minimum of the Allan (standard) Deviation.

$$\tilde{\sigma}_{\text{instability}} = \min_{\tau} \text{AD}(\tau) = \min_{\tau} \sqrt{\text{AVAR}(\tau)} \tag{4.4}$$

The bias instability is often used by manufacturers to describe the quality of the sensor, and represents the best bias stability that could be achieved for a given signal (assuming that the bias averaging takes place at the interval defined at the Allan Variance minimum).

The Angle- (Velocity- for accelerometer) Random Walk, $\tilde{\sigma}_{\gamma}$, is the standard deviation of the white noise γ in eq. (4.1). The notion of random walk comes from the fact the if the white noise effecting the angle rate or acceleration it becomes a random walk in angle or velocity. In [26] it is shown that the white noise effect on the Allan variance plot results in a slope of -1/2. Further the angle random walk can be found by evaluate the value of a -1/2 slope fitted to the Allan deviation plot at $\tau = 1$s.

The angle rate (acceleration for accelerometer) random walk $\tilde{\sigma}_w$ is the standard deviation of the white noise driving the biases in eq. (4.1). Here the name is quite fitting as it describes the random walk of the angle rate bias as a result of integrating the white noise in the Brownian motion process. In [26] it is shown that the white noise effect on the Allan variance plot results in a slope of +1/2. Further the angle random walk can be found by evaluate the value of a +1/2 slope fitted to the Allan deviation plot at $\tau = 3$s.

4.2.2 Implementation

The STIM300 IMU used consists of 3 high-accuracy MEMS-based gyros, 3 high stability accelerometers, 3 high stability inclinometers, internal temperature sensors, and the possibility for connecting a auxiliary sensor. It is factory calibrated and completely insensitive to magnetic fields. The inclinometers are not utilizing, but the analog pressure sensor is connected as a auxiliary sensor to the IMU. This allows the pressure measurements to be pre-processed together with the inertial measurements internally in the IMU. Three internal compensation and filtering modes were utilized:

- Low pass filtering of all measured signals with a low pass filter with the -3dB frequency equal to 16 Hz (default is 262 Hz).

- The angular rate sensor is internally compensated by the accelerometer to reduce the effect of linear acceleration in the angular rate measurements.
- As measuring angular rate is faster than measuring acceleration, angular rate is internally stored until the acceleration measurement is ready.

Having such a low cutoff frequency on the low pass filter adds 29 ms latency and will ignore measurements at higher frequencies. However, fast movements are not very relevant, as the robot will move underwater and, thus, moves very slowly. Furthermore, the inertial measurements are to be fused in the SLAM system together with images. As the SLAM algorithm only outputs new results after receiving a new image, there are no gains in having a small delay on the inertial measurements, as long as it is less than the delay of the images (see fig. 5.2b). If, in the future, the system is being used in a high speed setting with an algorithm which updates the position for every inertial measurement, one could consider using a higher cutoff frequency.

In this thesis, a novel open source driver[1] for communicating with the IMU, using modern C++17 techniques, was implemented. It includes interfaces for communicating with the STIM300 directly, using the standard USB driver of Ubuntu, or through a micro controller for better synchronization control. It is also supports easy addition of other interfaces. The driver is wrapped in a ROS package, but the driver can act as a standalone executable, or easily be integrated into other C++ projects.

Table 4.1: Noise parameters of the STIM300 IMU specified by the manufacturer. Determined using the Allan Variance technique at 25 ° C.

Parameter	Symbol: σ	Value	Unit
Angular random walk	γ_{gyro}	$1.2 \times e^4$	$\frac{\text{rad}}{\text{s}} \frac{1}{\sqrt{\text{Hz}}}$
Velocity random walk	γ_{acc}	$9.2 \times e^4$	$\frac{\text{m}}{\text{s}^2} \frac{1}{\sqrt{\text{Hz}}}$

4.2.3 Calibration

To verify that the IMU is working properly, and to estimate the noise parameters of the IMU model with respect to the IMU configured as described above, an Allan variance experiment was performed. Better noise characteristics than what is presented in the data sheet is to be expected, as the current operating cutoff frequency is lower on the internal low-pass filter.

The IMU was placed on a rigid surface for five hours. The first hour of readings was ignored to ensure the IMU was in a steady state. The remaining four hours of measurement

[1]`https://github.com/AndreasVaage/usm_stim300_driver`

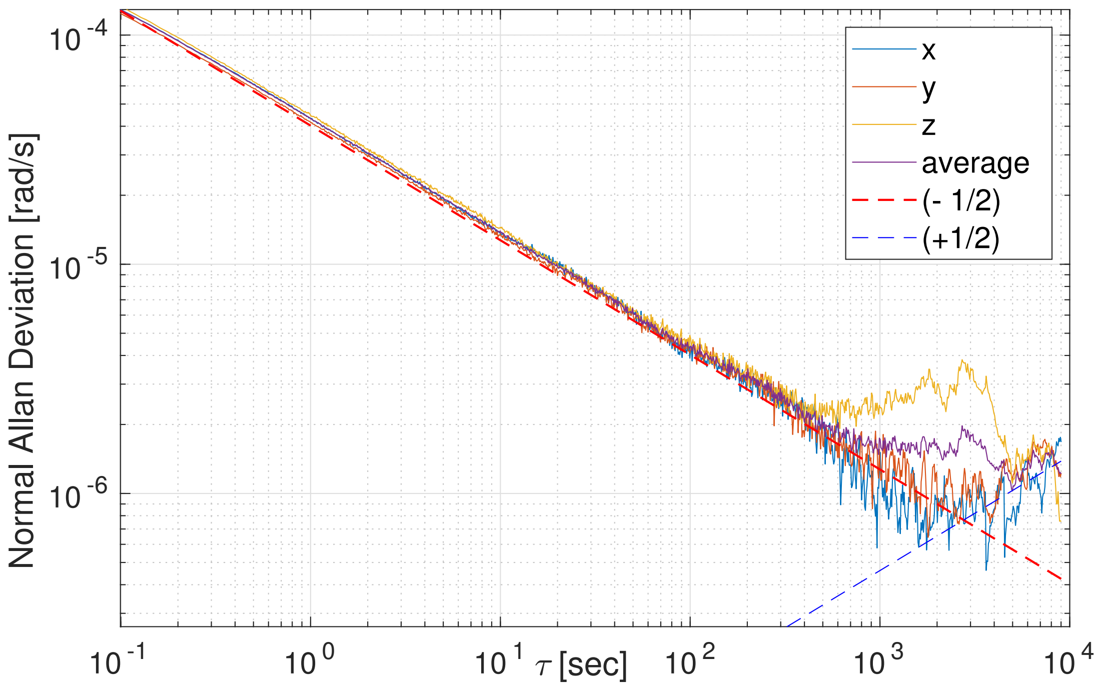


Figure 4.2: Allan (standard) Deviation plot of the angular rate sensor on the STIM300 IMU.

readings was collected at 125 Hz into a ROS bag, resulting in 1 802 442 readings. Then the rosbag were converted into a .mat file and used a matlab script[2] to calculate the Allan Variance plot. See fig 4.2 and 4.3. By evaluating the +1/2 slope and the -1/2 slope fitted to the Allan Plot at τ = 3s and τ = 1s respectively we can estimate the standard deviation for the white noise components of the

Table 4.2: Estimated noise parameters of the STIM300 IMU. Determined using the Allan Variance technique at 20 ° C.

Parameter	Symbol: σ	Value	Unit
Angular rate random walk	w_{gyro}	$1 \times e^6$	$\frac{\text{rad}}{\text{s}^2}\frac{1}{\sqrt{\text{Hz}}}$
Angular random walk	γ_{gyro}	$4 \times e^5$	$\frac{\text{rad}}{\text{s}}\frac{1}{\sqrt{\text{Hz}}}$
Acceleration random walk	w_{acc}	$3 \times e^5$	$\frac{\text{m}}{\text{s}^3}\frac{1}{\sqrt{\text{Hz}}}$
Velocity random walk	γ_{acc}	$4.7 \times e^4$	$\frac{\text{m}}{\text{s}^2}\frac{1}{\sqrt{\text{Hz}}}$

By comparing the estimated noise parameters with the ones from the data sheet, it became apparent that all achieved noise parameters are lower. This is most likely due to the fact that a lower cutoff frequency was used, at 16 Hz rather than the standard 262 Hz used in the data sheet estimation.

[2]`https://github.com/rpng/kalibr_allan`

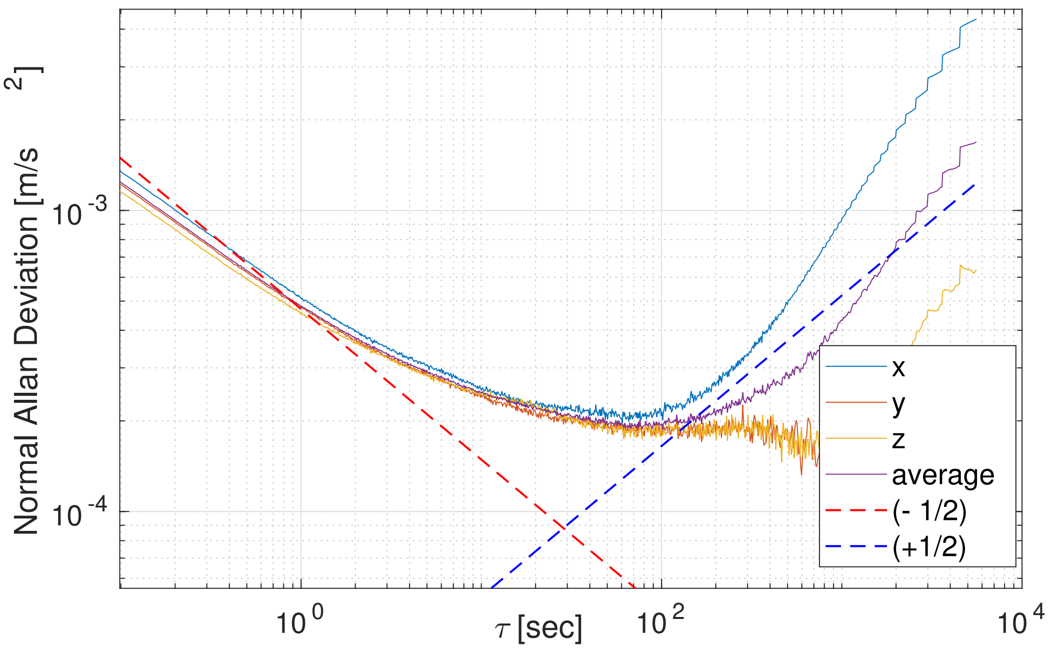


Figure 4.3: Allan (standard) Deviation plot of the acceleration sensor on the STIM300 IMU. The standard deviation of the underlying white noise, σ

4.3 Pressure sensor

A pressure sensor measures the pressure difference between the measured volume and a reference volume. Because of gravity the pressure of fluids or gasses generally increases the closer to the earths core one gets. Thus one can utilize a pressure sensor to measure the altitude above water or depth below water in a earth fixed frame such as NED.

4.3.1 Sensor Model

The measured pressure P is directly proportional to the depth D in the NED frame by the following equation:

$$P = \rho g D + \nu_{\text{pressure}} \tag{4.5}$$

Where ρ is the fluid density and g is the standard gravity and μ_{pressure} is white noise with standard deviation σ_{pressure}.

4.3.2 Implementation

For this thesis, a Honeywell PX3 pressure sensor was used. The PX3 is a fully calibrated and temperature compensated pressure sensor. It measures the pressure difference between the outside water and the atmospheric internal pressure inside the watertight enclosure. The sensor sends out an analog ratiometric voltage between 0.5 and 4.5 V. It is connected directly to the STIM300 aux input and is converted using a 24 bit ADC, filtered using the same 16 Hz lowpass filter as the IMU measurements, all inside the STIM300. Lastly the pre-processed pressure measurements are transferred together with the inertial measurements to the STIM300 driver. In table 4.3, *Accuracy BFSL* refers to the maximum deviation in output

Table 4.3: Pressure parameters

Parameter	Value
Producer	Honeywell
Product number	PX3AN2BS050PAAAX
Range	50 psi (24 m)
Total error band	$\pm 1.0\%$ FSS from $-20\,^\circ$C to $85\,^\circ$C
Accuracy BFSL	$\pm 0.25\%$ FSS
Ratiometric output	0.5 to 4.5 V_{DC}
Current consumption	3.5 mA max

from a Best Fit Straight Line (BFSL) fitted to the output measured over the pressure range at 25 °C. includes all errors due to pressure non-linearity, pressure hysteresis and pressure non-repeatably. Additionally, the *Total Error Band* refers to the maximum deviation from the ideal transfer function over the entire compensated temperature and pressure range. In addition to the error sources of Accuracy BFSL, it include errors due thermal effect on offset, thermal effect on span and thermal hysteresis.

4.3.3 Calibration

In order to estimate the depth from the voltage output of the pressure sensor, a linear regression fit was performed, finding the best line to match the output from the pressure sensor and the measured depth using a traditional meter (see fig. 4.4).

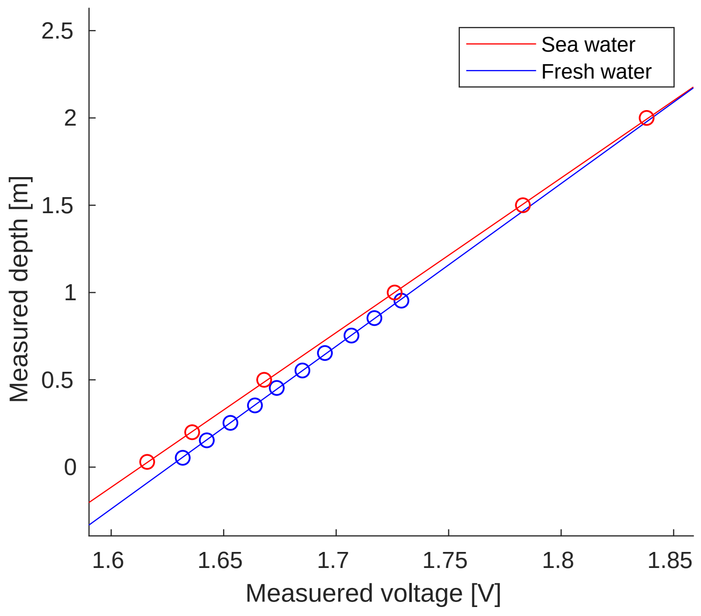


Figure 4.4: Calibration of the depth measurement using linear least square regression to fit the output from the pressure sensor to the actual depth in seawater.

The best fitted line had an r-square value of 0.9999 and is given by:

$$D = 8.86P_{\text{output}} - 14.29$$

Where P_{output} is the output from the pressure sensor given in volt and D is the depth in the NED frame.

4.4 Camera

Image capturing is, essentially, the act of projecting 3D data into a 2D plane — in other words, one dimension of information is lost. The dimension lost contains the depth information in the image. Through the use of multiple measurements of the same object(s) from different viewpoints, however, this information can be regained — given that the relative pose between the measurements is known.

4.4.1 Sensor Model

Before one can utilize the cameras to their fullest potential, it is necessary to have some sort of mathematical representation of the process of creating an image. As the USM is more interested in the world the image depicts than the image itself, it is useful to know how the process of capturing the 3D world in an image can be represented.

A common representation of a camera is the pinhole model. This representation views the rays of light reflecting off the objects in the world to be filtered through a small opening with a plane to capture the ray on the other side. See fig. 4.5 for a visualization.

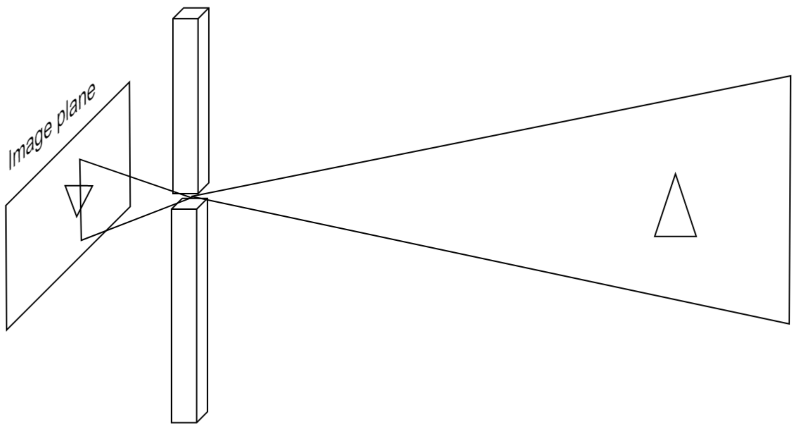


Figure 4.5: Pinhole camera model

A common factor for the model representation is how one moves between the normalized image plane, the pixel plane, and the 3D world. This can be represented as four composite transformations:

- The 3D world to the normalized image plane.
- The normalized image plane to the pixel plane.
- The pixel plane to the normalized image plane.
- The normalized image plane to the 3D world.

These transformations are in fact two operations and their respective inverse. These transformations are represented by the extrinsic and intrinsic matrix. The extrinsic matrix projects the 3D world to the normalized image plane by using three rotational and three translational parameters that can be described by a 4×4 homogeneous transformation matrix section 2.1.1. The intrinsic matrix projects the normalized image plane into the pixel plane.

The various models is determined by how the intrinsic matrix is structured. The calibration of the cameras refers to the process of finding actual values for the intrinsic matrix. This calibration is done by utilizing an easily recognized item with known dimensions in the 3D world. A common tool used for this, is a checkerboard pattern printed on a flat surface. The dimensions of the checkerboard squares are known. The corner points at the junctions between the white and black squares are easily detected. With the expected output known, one can calculate how the camera has deviated from this expectation. In other words, one can calculate how the camera renders the 3D world into the pixel plane. To perform camera calibration, the calibration tool Kalibr [3] was utilized. Kalibr supports the following four camera models: Pinhole, Omnidirectional, Double sphere and Extended unified camera model.

The pinhole camera model is the simplest and most common used camera model. It is the default camera model for most opens source SLAM methods, [28]–[30]. As the camera lens has low distortion and the domed window is supposed to reduce distortion effects by the water, the standard pinhole model is a natural choice. The pinhole model contain the following intrinsic parameters in the intrinsic matrix, eq. (4.6).

$$K_i = \begin{bmatrix} f_u & 0 & p_u \\ 0 & f_v & p_v \\ 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

Radial and tangential distortion is added to handle the normal pinhole model-errors such as the lens non-linearities and image-sensor miss-alignments [31]. As well as the additional model misfits that will arise due to non-perfect alignment between camera and glass dome and non-perfect water-glass-air distortion by the dome [32]. In order to represent the distortion effect, Kalibr support the following distortion models: Radial-tangential, Equidistant and FoV.

In the process of capturing the image the depth information is lost. For this thesis, a stereo camera setup is chosen to recover this depth information. Reconstruction of the 3D information usually consists of the steps: camera calibration, stereo rectification, stereo matching and 3D reconstruction [33]. Camera calibration refers to computing the camera parameters [33]. Stereo rectification refers to reducing the matching problem between the left and right camera view from 2D to 1D by aligning the epipolar lines [33]. Stereo matching refers to finding pixel-wise correspondence between the left and right camera view [33]. 3D reconstruction refers to recovering 3D information from the disparity image [33]. With a point located in both the left and right camera, the 3D position can be triangulated using the camera parameters [34].

[3]https://github.com/ethz-asl/kalibr

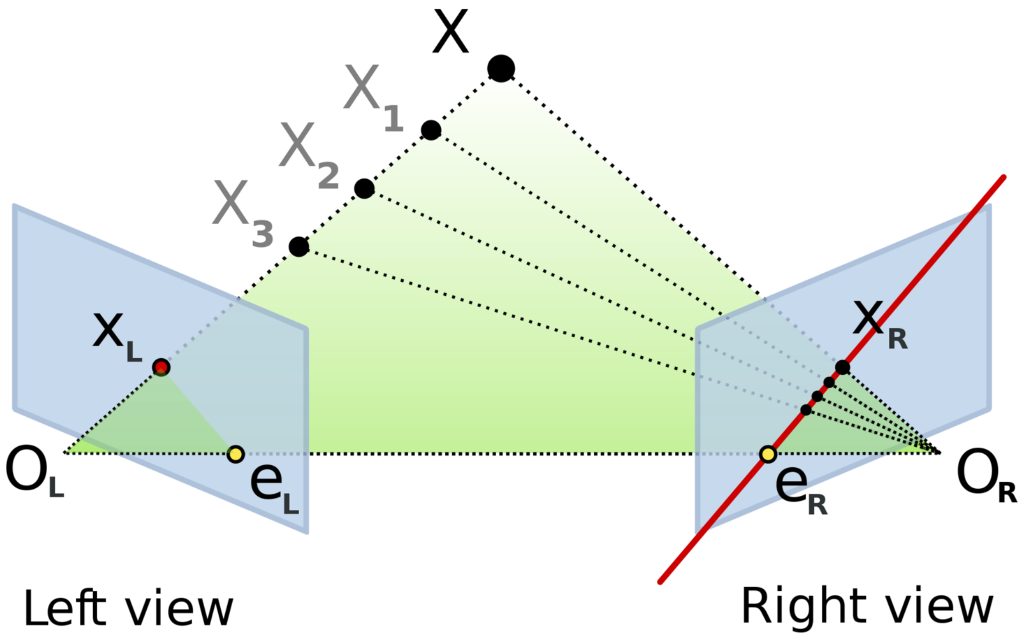


Figure 4.6: Illustration of triangulating the 3D position with a stereo camera setup [4].

Figure 4.6 illustrates how the second view can be utilized to remove the ambiguity of where on the line spanning from O_L through X_L the point X is located. The red line — i.e. the epipolar line — denotes where in the right image one can expect to find the point X. However, before trying to do any calculations the images are rectified remove distortion and aligning the epipolar lines [33], an illustration of this can be seen from fig. 4.7.

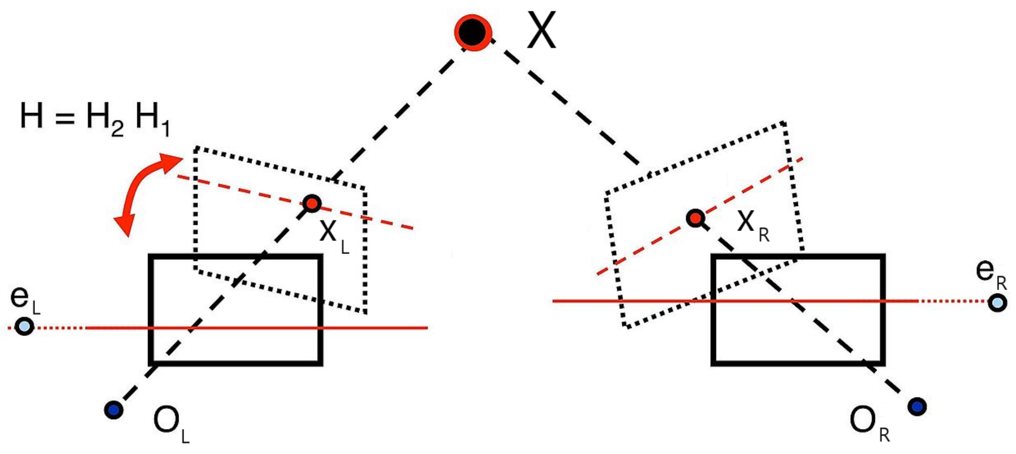


Figure 4.7: Illustration of stereo rectification [5].

[4]This image is downloaded from `https://www.digitalbridge.com/blog/can-we-use-stereo-vision-with-arkit-to-estimate-floor-plans`, stated to be courtesy of `https://commons.wikimedia.org/wiki/Main_Page`

[5]This image is downloaded from `https://www.wikiwand.com/en/Image_rectification`

After the stereo rectification, the depth from O_L and O_R to the point X will be the same. The distance between O_L and O_R is called the baseline and will be necessary to determine the epipolar lines. This baseline will be determined when calibrating the stereo camera setup.

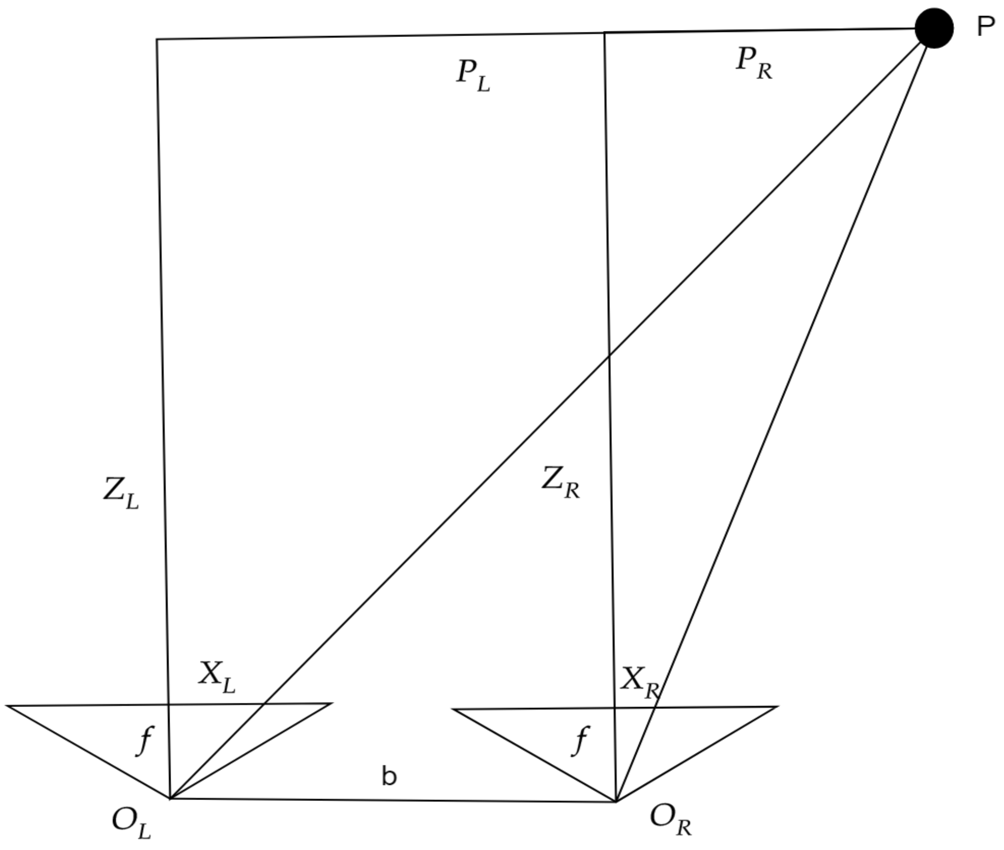


Figure 4.8: Figure to clarify variables for eq. (4.7) through eq. (4.13).

$$\frac{X_L}{f} = \frac{P_L}{Z_L} \tag{4.7}$$

$$\frac{X_R}{f} = \frac{P_R}{Z_R} \tag{4.8}$$

$$Z_L = Z_R = Z \tag{4.9}$$

$$P_L = P_R + b \tag{4.10}$$

$$d = X_L - X_R \tag{4.11}$$

$$d = f \cdot \frac{P_L - P_R}{Z} = f \cdot \frac{(P_R + b) - P_R}{Z} = f \cdot \frac{b}{Z} \tag{4.12}$$

$$Z = f \cdot \frac{b}{d} \tag{4.13}$$

When calculating the depth information from the rectified images, similar triangles are used to establish eqs. (4.7) and (4.8). Equation (4.9) is a consequence of the rectified images having the same depth away from the point X. The distance between the camera positions are given by the baseline b, see eq. (4.10). Equation (4.11) show the difference in pixels of X from the left and right camera image. Equation (4.12) shows the various steps of substituting eq. (4.8) through eq. (4.10) into eq. (4.11), which culminates into a formula for the depth information, see eq. (4.13).

The projection functions for monocular and rectified stereo vision are then given by eqs. (4.14) and (4.15), respectively. These can be used to project a 3D point in the camera coordinate system into the image plane. For stereo camera the depth is included.

$$\Pi_m\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \cdot \frac{X}{Z} + c_x \\ f_y \cdot \frac{Y}{Z} + c_y \end{bmatrix} \tag{4.14}$$

$$\Pi_s\left(\begin{bmatrix} X \\ Y \\ Z \end{bmatrix}\right) = \begin{bmatrix} f_x \cdot \frac{X}{Z} + c_x \\ f_y \cdot \frac{Y}{Z} + c_y \\ f_x \cdot \frac{X-b}{Z} + c_x \end{bmatrix} \tag{4.15}$$

4.4.2 Implementation

Two Pointgrey (now FLIR) Blackfly GigE cameras are used. They give complete control over highly sensitive sensors with global shutter and possibility for external hardware triggering or software triggering. They are connected directly to the Ethernet swithc through one separate Ethernet cable each. On the computer the open source point-grey camera driver[6] is used to process and foreword the images to the ROS network. The lens and camera specifications are listed in table 4.4. The cameras are very customizable, with two GPIO pins which can be used to trigger or synchronize the cameras. Further they have global shutter, which means the entire image will be taken at once, unlike rolling shutter cameras, where one line is captured at a time. They are equipped with some pretty special lenses, as they are so called wide angle low distortion lenses. They promise 125° FoW, while maintaining less than 3% distortion. The cameras and lenses are to be placed in a water thigh cylinder, with a domed window. The domed window will remove most of the water-glass-air distortion, and since the lens is also low distortion, the camera setup should be pretty close to a ideal pinhole camera, even under water.

[6]`https://github.com/ros-drivers/pointgrey_camera_driver`

Table 4.4: Camera parameters

Parameter	Value
Product number	BFLY-PGE-13S2C-CS
Resolution	1288x964
Frame rate	30
Readout Method	Global shutter
Triggering	External and Software
Synchronization	GPIO pins
Meta-data	Timestamp, img count, img parameters
Sensor	Sony ICX445
Sensor Format	1/3"
Lens Focal Length [mm]	1.28
Field of View	125°
Distortion [%]	< 3

4.4.3 Calibration

There are several aspects to calibrating a camera. Firstly, the physical lens, as well as the camera settings, can be tuned. The camera lenses have two physical adjustable knobs, one for tuning the focus and one for tuning the aperture. In addition the choice of shutter time of the camera is strongly dependent to the lens settings. If the shutter time is too long the camera sensor will receive a lot of light, but if the camera is moving during this time there will be motion blur as a result of the change in scenery being merged into one image. There are also digital setting such as gain, which can be applied, but increasing these setting will reduce the signal-to-noise ration in the image. The focus and aperture has to be set ones, and can not be changed during run-time. A large aperture sends through a lot of light thus allows for a shorter shutter time, which again allows for faster movements without motion blur. However a larger aperture also results in a more blurry image, or at least reduces the range in which the scene is in focus. Since the cameras are moving underwater, the damping from the water reduces fast movements. Thus a bigger shutter time than for example used in UAVs where there are a lot of shaking and rotation can be applied. After some trial an error the shutter time was set to 5 ms. This shutter time was short enough to not induce significant motion blur with the motions one could expect underwater. Then the aperture and focus was tuned using an entropy based method[7]

To estimate the lens and sensor parameters, calibration boards were used. Calibration boards for underwater use were custom designed and printed directly on aluminum frames by a commercial manufacturer[8]. The printing was cheap, but there were no guarantee for

[7]https://github.com/ethz-asl/kalibr/wiki/camera-focus
[8]https://www.japanphoto.no/plakater-og-fotolerreter/alu-plate.html

the print to handle water. However after several days of testing, where one of the boards have been submerged underwater for approximately a total of 4-6 hours, there are no signs of loose or deformed paint. Another problem with the commercial manufacturer was that they cut the edges of the uploaded image and then scaled the image to the original size, without knowing exactly how much they cut and scale. The calibration board images was created by modifying a open source python/latex script[9] to support custom sized white edges around the calibration patterns and pre-scaling to account for the scaling during the manufacturing. The resulting calibration boards can be seen in fig. 4.9. The used pre-scale was an educated guess based on discussion with customer support, for creation of future calibration boards the following settings were used:

- 6×4 Checkerboard printed on 60×80 cm aluminum frame with 5cm white space. Wanted 10 cm squares. Used pre-scale factor of 0.984. Resulting squares: 10.1 cm. Correct pre-scale factor 0.973.
- 5×5 April-grid printed on 60×60 cm aluminum frame with 2.28 cm white space. Wanted 8 cm squares with 0.3 square-space ratio. Used pre-scale factor of 0.975. Resulting squares: 8.03 cm. Correct pre-scale factor 0.971.

The standard checkerboard pattern is accepted by most common camera calibration software, but the entire board has to be visible in each image for it to be usable. The april-grid calibration pattern is usable even if only a small portion of the board is visible in the image because each square is unique. Another advantage is that the april-grid is asymmetric, thus there is always only one solution when calculating the relative position between the board and the camera, unlike the checkerboard where there will always be at least two solutions. This is not important for camera calibration, as only the relative position between each pair of camera-patterns are of interest. But it is critical if the board is used as a landmark during a free run, as in chapter 9, because then one is interested in the camera pose relative to the pattern over time.

During camera calibration the cameras were kept steady to reduce motion blur, while the calibration board was moved around in front of the camera. Camera calibration is model fitting, where the model is a lower dimension estimate of the reality. To ensure the a good model fitting it is important to move the calibration board in the entire field of view (FoV) of the camera in the selected distance range. As there are limited sight underwater, and the test pool is quite shallow (1 m), a distance range of 0.2 to 3 meter was chosen. Images of the calibration board where collected in the entire FoV over the distance range and with angles. section 4.4.3 shows a illustration of the calibrated distortion model, and it is worth noting just how little distortion there is. Considering the system is under water with a 125°

[9]https://github.com/ethz-asl/kalibr/blob/master/aslam_offline_calibration/kalibr/python/kalibr_create_target_pdf

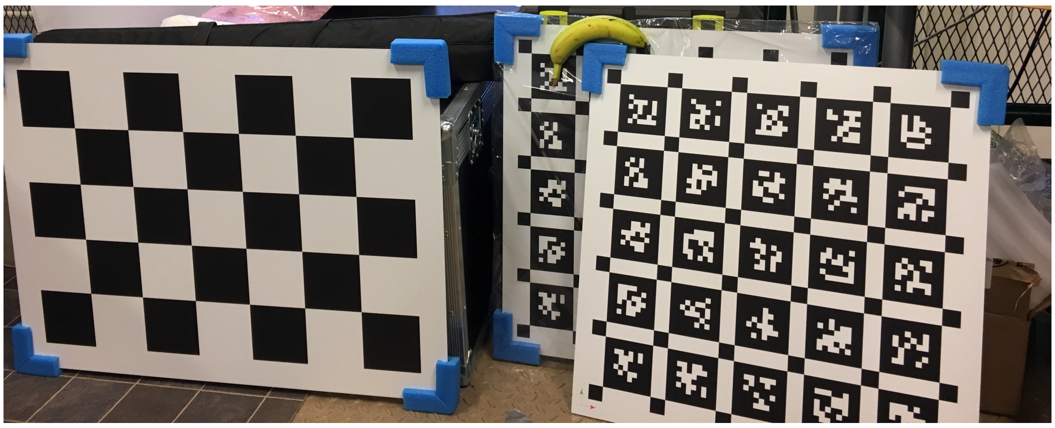


Figure 4.9: New calibration boards printed directly on aluminum frames. Left 60x80 cm standard checker board, right 60×60 cm april-grid. Banana for scale.

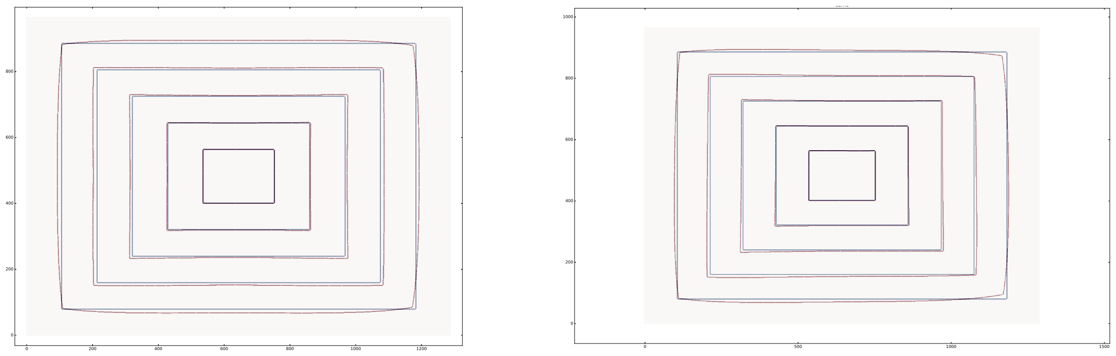


Figure 4.10: Radial and tangential distortion for the left and right camera. The red squares displays how much the blue squares would be distorted if they where captured by the camera.

Table 4.5: Calibration result from representative environments.

	Mean	std
Left camera	0.90 pix	0.44 pix
Right camera	0.91 pix	0.37 pix

lens. It appears that the domed shape of the protection glass is effectively counteracting the water-glass-air distortion, and that the wide angle low distortion truly has low distortion.

Resulting values after camera calibration using Kalibr for pinhole model with radial and tangential distortion: reprojection error left camera: 0.22 pixels, right camera: 0.24 pixels. However these values are not representative for how the system performs because the images where taken with a stationary camera, the camera model where optimized for these exact images and outliers where removed. Thus a verification experiment where created where the camera was moving fast, the camera model was fixed and no outliers where removed. The results for the cameras can be seen in table 4.5.

These results includes partly bad lightening and a small degree of motion blur, and are representative for the accuracy of the camera system and the camera model during run-time.

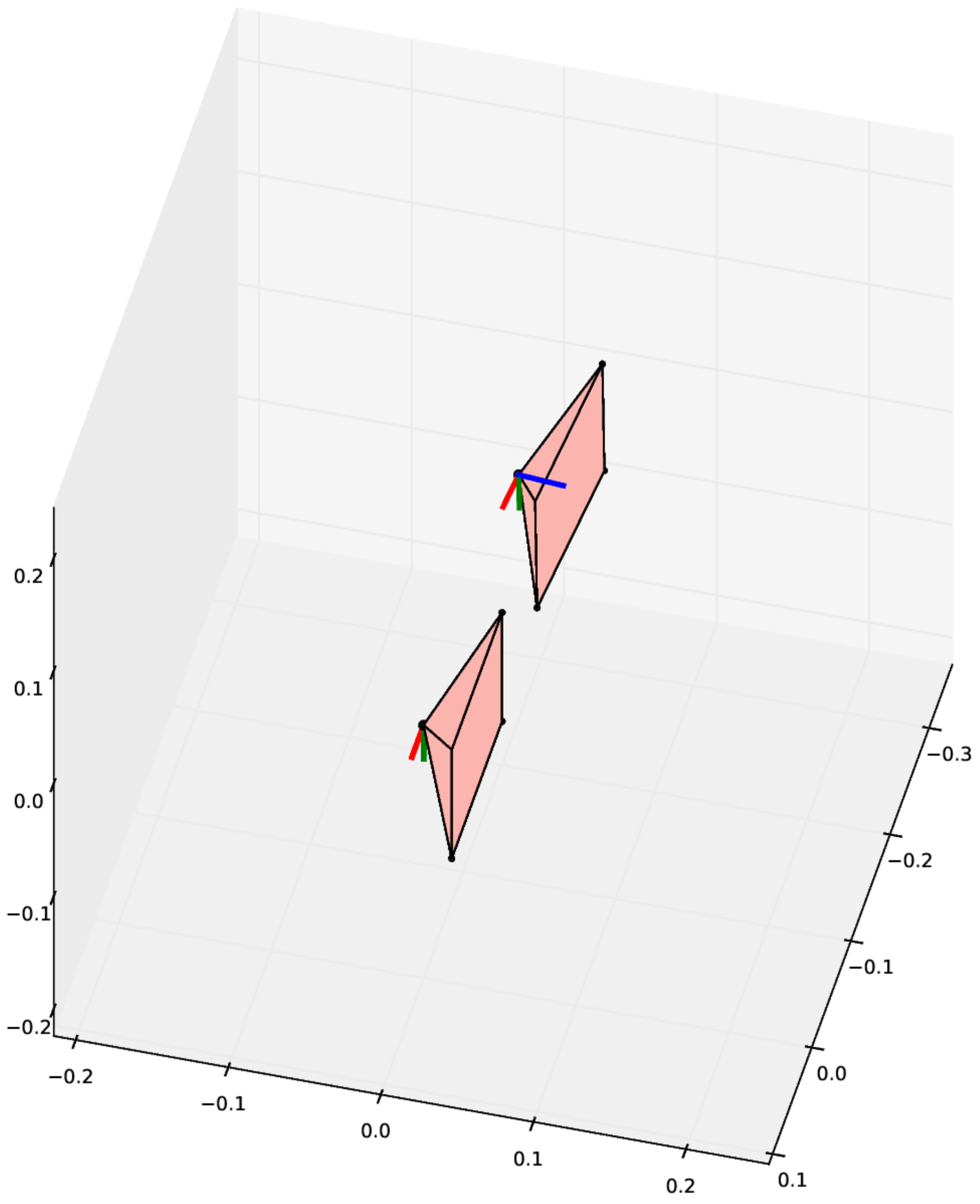


Figure 4.11: Stereo camera calibration result.

5 | Electronic Design

BEFORE implementing the hardware platform, certain aspects of the electronic design needs to be examined.

5.1 Synchronization

In order to achieve accurate and robust sensor fusion, it is important that the sensors are synchronized [35], [36]. There are two kinds of synchronization when talking about sensor fusion.

- Time synchronization: Each measurement have a time-stamp to represent when the measurement was taken. This time-stamp are given by a synchronized clock.
- Measurement synchronization: The measurements themselves are synchronized such that the sensors performs measurements at the same time.

Time synchronization is most important, because every sensor fusion algorithm needs to be able to compare the time-stamps from different sensors. Further the accuracy of the time-stamps directly effect the accuracy of the fusion algorithm similarly to how the accuracy of the measurements effect it. The difference is only that the effect of inaccurate time-stamps are dependent on the rate of change in the measurements. The biggest source of inaccuracy of a time-stamp is generally a unknown and varying time between the acquisition of the measurement and when the time-stamp is set.

Each measurement goes trough several steps before it arrives to the sensor fusion algorithm, see fig. 5.1. The time for each step varies slightly for each measurement, particularly the transfer of the measurement data from the sensor to the sensor fusion algorithm can vary a lot. For example there can be a buffering effect, where in order to save processing time the system stack several small measurements together before they are transferred or after they are transferred, but before they are read by the operating system. In such cases it is important that the time-stamp is set before the measurement is transferred because the transfer time can vary by several sampling periods and is generally unknown. To make the time-stamps accurate they should be set as close to the acquisition as possible. For sensors such as cameras where the acquisition time, called shutter time, can be quite long and often vary based on the amount of light in the scene. In high speed systems such as UAVs, a common approach to increase accuracy of the time-stamp is to set it to be in the middle of the acquisition [36].

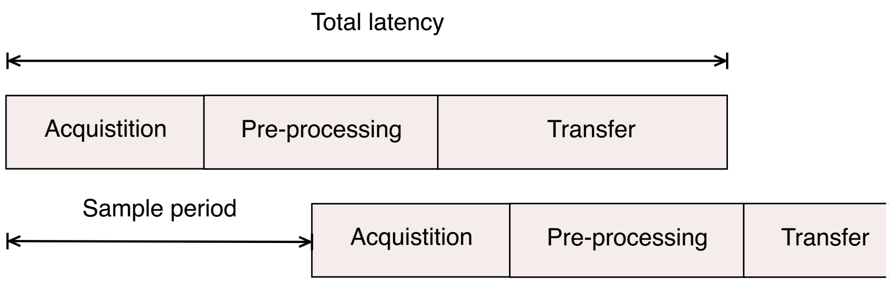
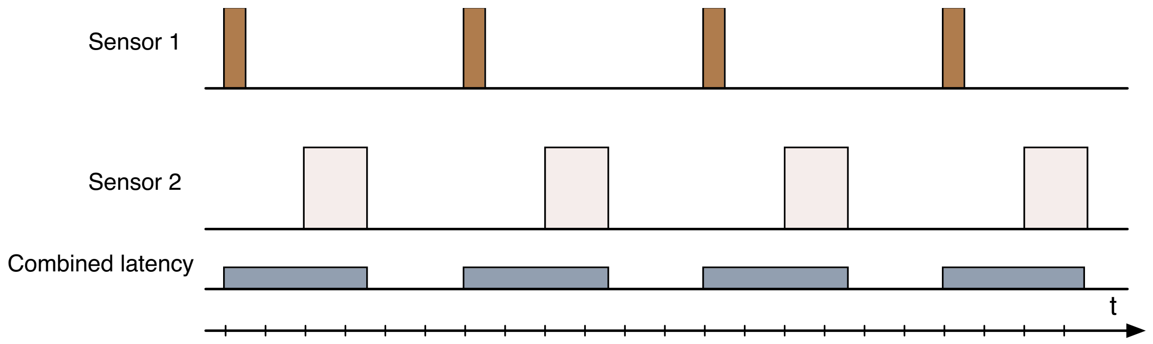
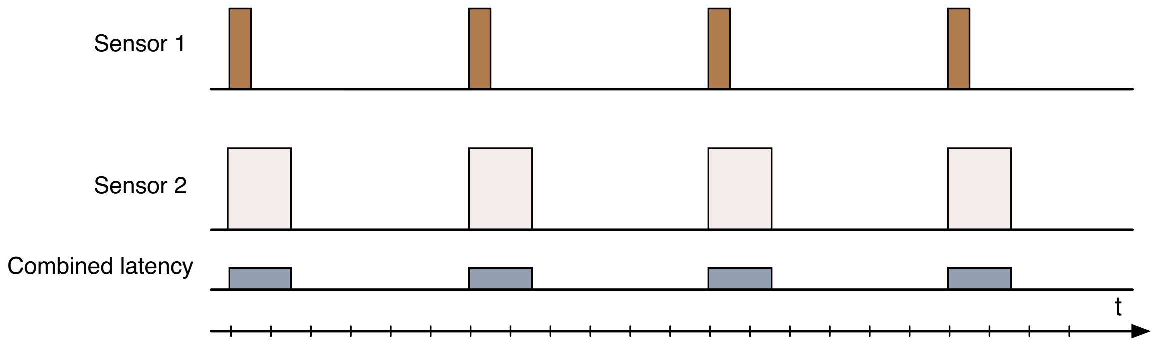


Figure 5.1: Measurement latency is composed of the time it takes to acquire the measurement, pre-processing and the time it takes to transfer it to the sensor fusion algorithm.



(a) Combined sensor latency for asynchronous measurements.



(b) Combined sensor latency for synchronous measurements.

Figure 5.2: Combined sensor latency for fusion algorithms who require both measurements before they can give an output. The width of each sensor measurement represent its total latency.

Measurement synchronization is not equally important, and there are many approaches to deal with asynchronous measurements. For example, a basic Kalman filter handles asynchronous measurements by default, as long as they arrive in the correct order. Often this is not the case. One can easily imagine a simple inertial measurement being acquired after a GPS measurement, but due to different latencies, it arrives first to the Kalman filter. To overcome this, the measurements need to be sorted based on sampling time. The easiest way to do this, is to store the last few inertial measurements in a buffer while waiting for the GPS measurement to ensure the measurements are transferred to the Kalman filter in the correct order. For asynchronous measurements, determining the required length of this buffer can be challenging, especially if you do not know when the next GPS measurement will arrive. Generally, the main advantages of operating with synchronous measurements is simplicity and minimized delay [35], see section 5.1. There are exceptions where measurement synchronization is crucial, however, for example when dealing with stereo camera images. In this case, images from two cameras are directly compared to estimate the depth of the scene.

5.1.1 Implementation

The pressure sensor is connected as an auxiliary sensor to the IMU, therefore, in the context of synchronization, IMU or inertial measurements also includes the pressure sensor or pressure measurements. Both the IMU and the cameras are equipped with digital pins for sending and receiving trigger signals in the form of 5 or 3.3 V level shifting signals. As the IMU has the highest measurement frequency, it runs in free running mode. In this mode the internal controller of the STIM300 IMU outputs measurements at a chosen frequency. The IMU runs in free running mode instead of being triggered externally, because in free running mode, the STIM300 IMU use the trigger signal to inform when all data from a measurement is sent, allowing for more robust data transfer.

The initial plan was to simply connect the output sync signal of the IMU directly to the camera and let the camera take a picture at every new IMU measurement. Unfortunately, the cameras were too old to have this ability, thus, a micro-controller was installed to register the trigger signal from the IMU and trigger the camera every *n*th time. As shown in fig. 5.3. By triggering the camera every *n*th inertial measurement, measurement-synchronization is achieved. This system does, however, still have very poor time-synchronization. There is no way of accurately synchronizing the internal clocks of the camera and IMU with the computer clock. Thus, the only option is to set the time-stamps of the measurements when they arrive at the computer. Unfortunately this is after being pre-processed and transferred, which, for both the images and inertial measurement, introduces significant latency variation. Another problem was that there was no way of communicating with the

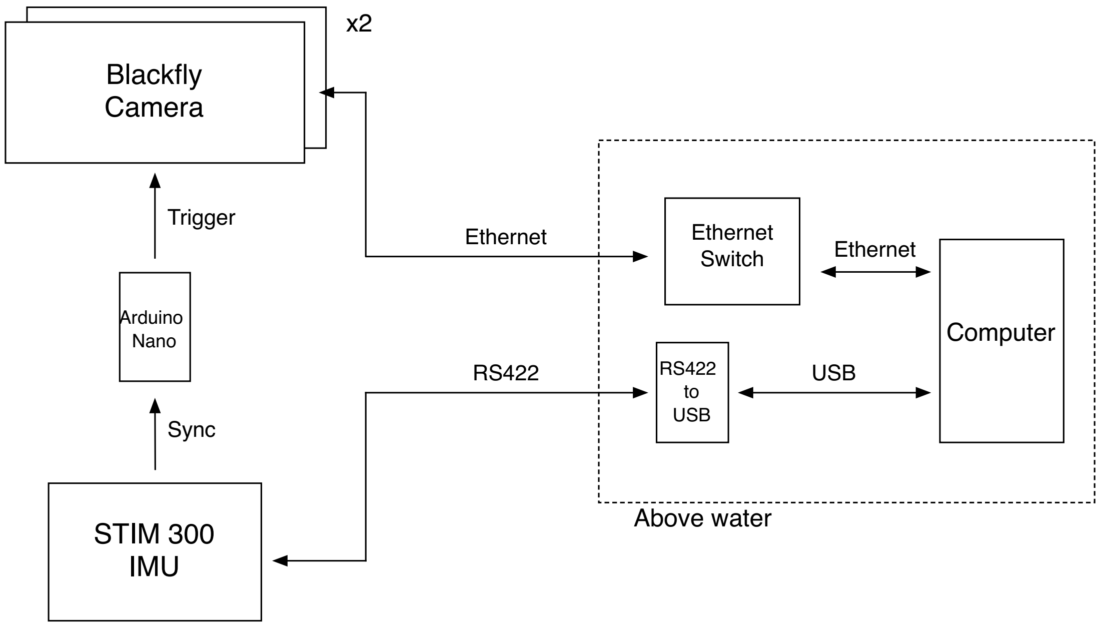


Figure 5.3: Data flow for the Visual-Inertial sensor.

micro-controller, thus the entire enclosure had to be opened if the triggering ratio between the camera and the IMU needed to be adjusted.

To achieve more accurate time-synchronization, and allow for communication between the micro-controller and the computer, the IMU data was redirected through the micro-controller. At this time, partly due to hardware faults, Ethernet was decided used as communication protocol between the IMU and the computern instead of the intended RS422. This altered design is shown in fig. 5.4. The micro-controller clock is then synchronized with the computer clock over the Ethernet connection. Thus the IMU time-stamp are set directly by the micro-controller when it registers the sync signal. Resulting in a near perfect time-synchronization for the inertial measurements.

The cameras can provide a time-stamp as meta-data for each image, however the internal clock of the camera used to set these time-stamps needs to be synchronized with the computer clock. To achieve this, a simple start-up routine was created. During start up, after the micro-controller clock is synchronized with the computer clock, the micro-controller triggers the camera with a 2 second period, and sends the trigger-time to the computer. Assuming an image uses less than 1 second to transfer to the computer. If an image and a trigger-time message from the micro-controller arrives within 1 second apart, then they must be matching. After finding 10 such matches, the median time difference is calculated and added to every future time-stamp. Thus the camera is synchronized with the micro-controller clock, which again is synchronized with the main computer clock.

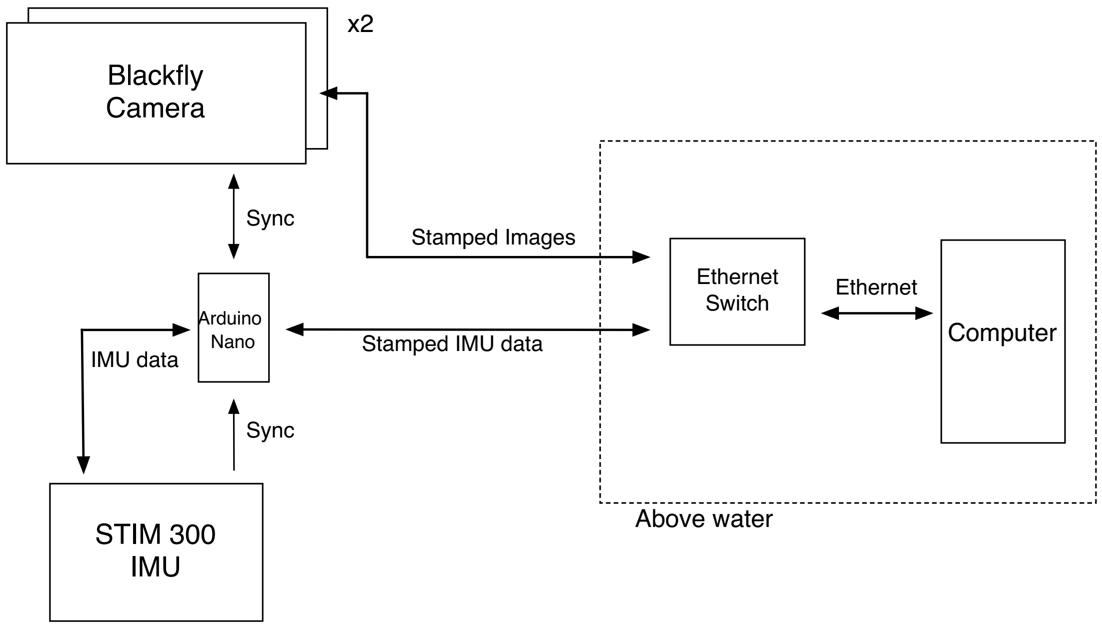


Figure 5.4: Data flow for the Visual-Inertial sensor whit more accurate time-synchronization

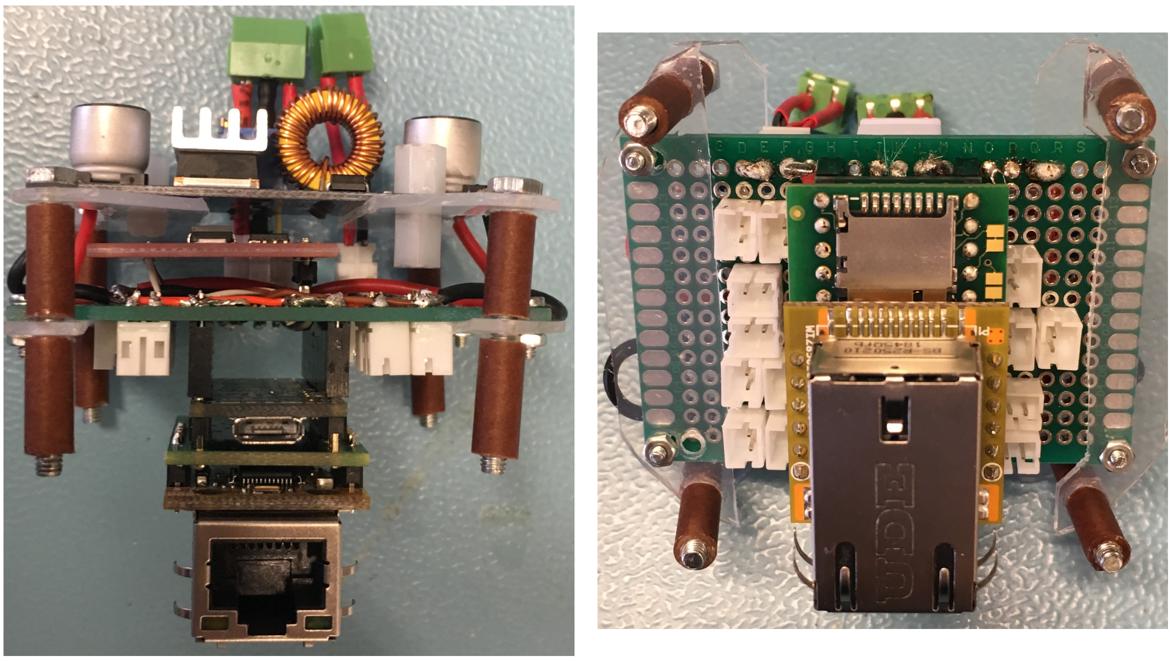


Figure 5.5: Control module. Synchronizes sensors, adds timestamps to inertial and pressure measurements, regulate voltage levels, control lights and watch over the leak sensors.

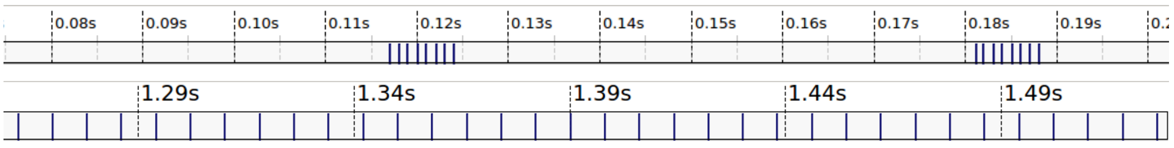


Figure 5.6: Buffering effect on inertial measurements. The figure displays two time lines, where the blue lines represents time-stamps of the inertial measurements. Top shows the time-stamps when set based on arrival time of the inertial data to the computer. Bottom shows the time-stamps when set based on the triggering signal directly from the IMU.

5.1.2 Experiments

First a result of the buffering effect is presented, then time-synchronization experiments are discussed, before lastly measurement synchronization is discussed.

In fig. 5.6, there is a clear display of the buffering effect. In both the timelines, the IMU is sampling at 125 Hz. However, for the upper timeline, the time-stamp is set after the measurement arrives to the sensor fusion algorithm according to fig. 5.3. While in the lower timeline the time-stamp is set directly by the microcontroller once it receives a triggering signal from the IMU as shown in fig. 5.4. One can clearly see that the time-stamps have been clumped up into batches, due to the fact that the measurements themselves have been stored in buffers during the transfer.

In order to estimate the time synchronization, a continues, spatial and temporal visual inertial calibration method [37], [38] was used, it is implemented in the ROS Kalibr toolbox. It uses the sensor model described in eq. (4.1) for the IMU, and supports the same sensor models as discussed in chapter 4 for the camera models. The pinhole model with radial and tangential distortion models is used. The sensor model parameters used are the ones found during previous calibration of the cameras and the IMU.

The setup is quite similar to camera calibration, except the intrinsic and extrinsic camera models, as well as the noise parameters for the IMU model, are known. The goal is to estimate the temporal and spatial offset between the cameras and the IMU. Instead of keeping the cameras still and moving a calibration board in front of the cameras, the calibration board is kept still and the VI-sensor is moved in front of the calibration board. This is so the IMU can measure the movements, and it is important to excite all the axes of the IMU in order for the calibration algorithm to estimate the IMU biases correctly. Since the cameras needs to be moving and rotating relatively fast during the calibration it is crucially for the camera to be operating with a low shutter time and good lightening to avoid motion blur. The calibration method uses the known camera model as well as the known calibration pattern to estimate the relative position between the camera and the calibration pattern. Splines are used to interpolate the pose between image measurements. Thus it is important that the frame rate of the cameras are as high as possible to reduce the

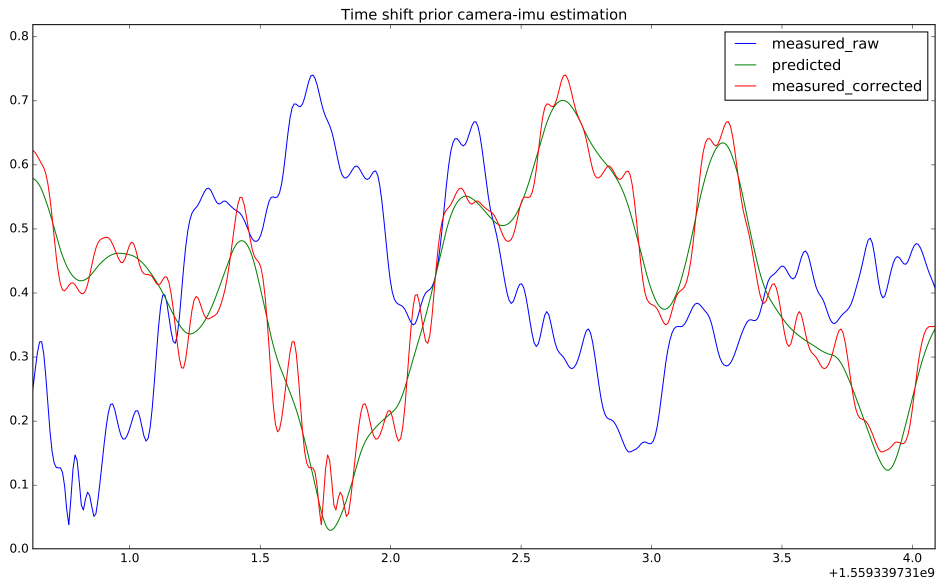


Figure 5.7: Time synchronization between camera measurements and IMU measurements during development and not working properly. The blue line represents the absolute rotation rate in rad/s from the IMU measurements as a function of time. The green line represents the predicted rotation rate estimated from the camera models and the calibration boards. The red line represent the time-corrected IMU measurement.

distance between two images which again make it possible to fit splines more accurately.

The relative spatial placement can be seen in fig. 6.7. During the first implementation, see fig. 5.3, the ROS node, running on the main computer, received the inertial measurements directly from the IMU. The node pulled the USB buffer with a given frequency, when new measurements arrived it read the datagram and parsed the raw values into structures and converted them into SI units according to the data sheet, before it was published as a ROS IMU topic. For some time it ran with the same frequency as the sampling frequency of the IMU, but it turned out to be a slightly lower frequency, because the buffer increased over time as new measurements arrived from the IMU faster than the node read them. Thus the time offset between the camera measurements and the IMU measurements became very large as seen in fig. 5.7. The temporal offset became close to 1 sec before the buffer was full.

After implementing the second synchronization design as seen in fig. 5.4, there was first a constant temporal offset of 5.0 ms. After subtracting this offset the temporal offset was measured to be ±0.1 ms over three calibrations. See fig. 5.8.

After verifying that the timestamps are accurate it is easy to verify the measurement synchronization. Both the cameras are triggered simultaneously by the micro-controller every 6th inertial measurement as seen in fig. 5.9.

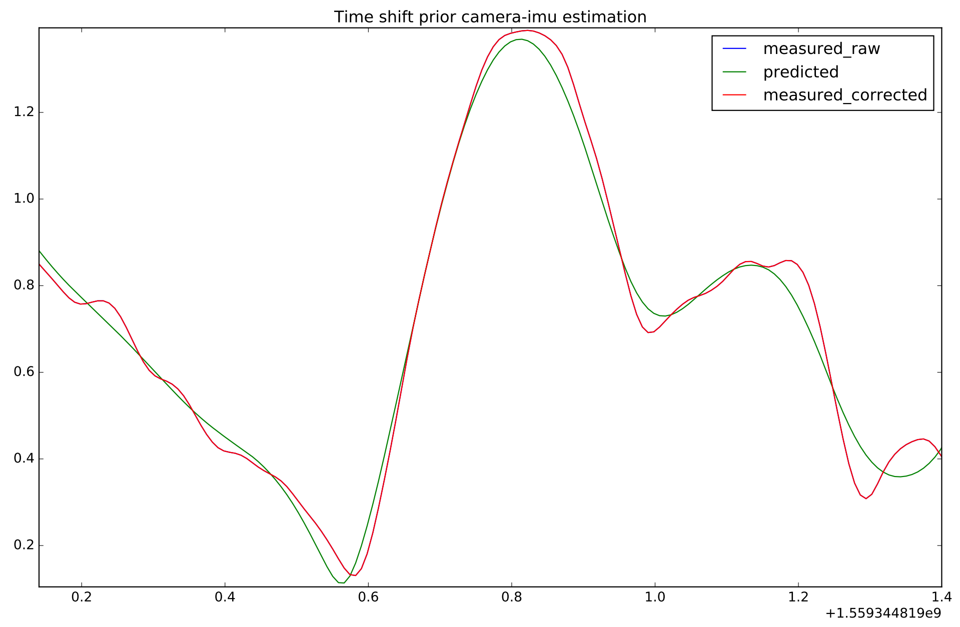


Figure 5.8: Resulting time synchronization between camera and IMU after accurate synchronization.

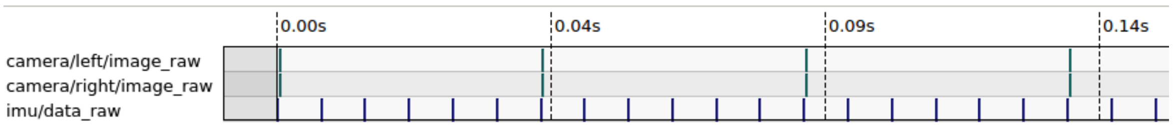


Figure 5.9: Measurement synchronization between camera measurements and IMU measurements.

6 | Mechanical Design

The hardware setup consists of multiple sensors, as described in chapter 4. However, without a well planned assembly strategy, the sensor will not be able to provide a good data foundation. In order for the sensor model to work well, it is important to know where all the sensors are located in relation to each other. These relative positions will need to be consistent so that the entire sensor package avoids the need for frequent re-calibration. This will be achieved by having the modules rigidly attached. The sensors will also need to be enclosed in water tight compartments. Where possible, it will be advantageous to find commercial products, as this will free time for other work. Where there are no suitable commercial option easily available, the components are custom produced at the institute of Cybernetics mechanical department or 3D printed.

6.1 Water Tight Housing

For water tight housings, it was looked towards Blue Robotics as they have multitudes of underwater housing products. After evaluating their various solutions, it was decided to use their 2" watertight enclosures with dome caps, with custom length for the camera housing. This way the cameras can be oriented freely regardless of what they are mounted to. Additionally, there are no out of the box product to mount the cameras within the cylinders. It was therefore necessary to design a 3D printed mounting structure to fit within this housing, capable of holding the camera in a manner which ensured that the camera is rigidly attached, as can be seen from figure 6.2a. This mounting structure will enable the camera lens to be place in the center position with regards to the dome cap, as seen in figure 6.2b. It was also decided on using one of Blue Robotics 3" enclosures to house the synchronization circuit. This extra size is meant to account for the size of the micro controller and cable

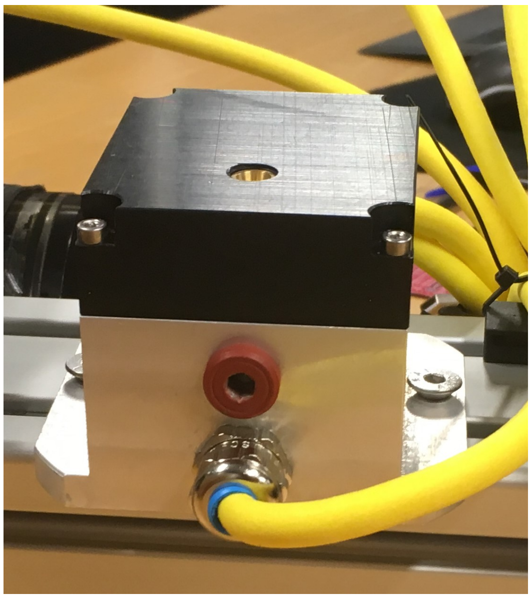
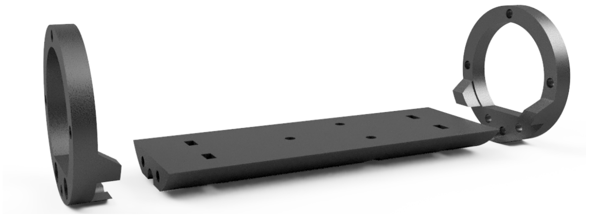


Figure 6.1: Water housing IMU and preassure sensor

management. The last components that needed to be protected from the water is the IMU and the pressure sensor. There were no easily available commercial option for this, and after some dialogue with the mechanical department of institute of Cybernetics at NTNU, they manufactured the part shown in figure 6.1.


(a) Camera mount 3D design

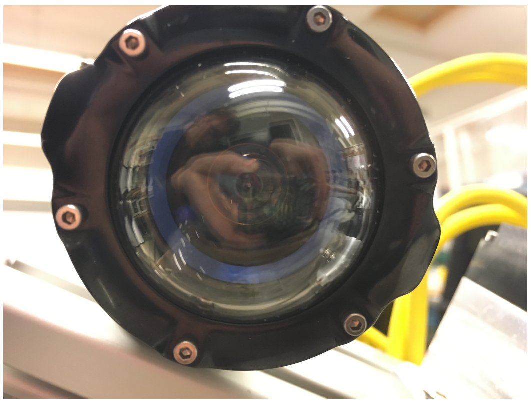

(b) Lens centered

Figure 6.2: Mounting the cameras inside the water tight cylinders

6.2 Mounting the Components

For the bearing construction, AluFlex was contacted due to their products being found to be flexible for a sensor rig that may be expanded in the future, or slightly adjusted as it is created. After close dialogue with the manufacturers, the design settled on an aluminium rail of dimensions 80 × 80 mm cross section and 320 mm length. The design can be found in figure 6.3.

The last piece of the puzzle was then how to rigidly attached the camera cylinders to the bearing construction. This was done by designing a two component mounting which can be clamped together around the camera cylinder, while the bottom mounting part can be screwed to the bearing structure interface. The top part can be seen in fig. 6.4a and the bottom part in fig. 6.4b. Four sets of these constructions were then 3D printed, two for each of the camera cylinder, as seen in figure 6.4c. This enabled two mounting points of the cylinder that can be placed arbitrarily along the bearing constructions rails.

With all the components mounted to bearing constructions, the final hardware platform provides two rigidly attached camera cylinder, a pressure sensor, an inertial measurement unit, and a synchronization circuit. Additionally, there are attached some cable management components and a connection point between the hardware platform and the USM, see fig. 6.5. The connection point enable the hardware platform to be mounted at a small angle. The complete system can be seen in figure 6.6.

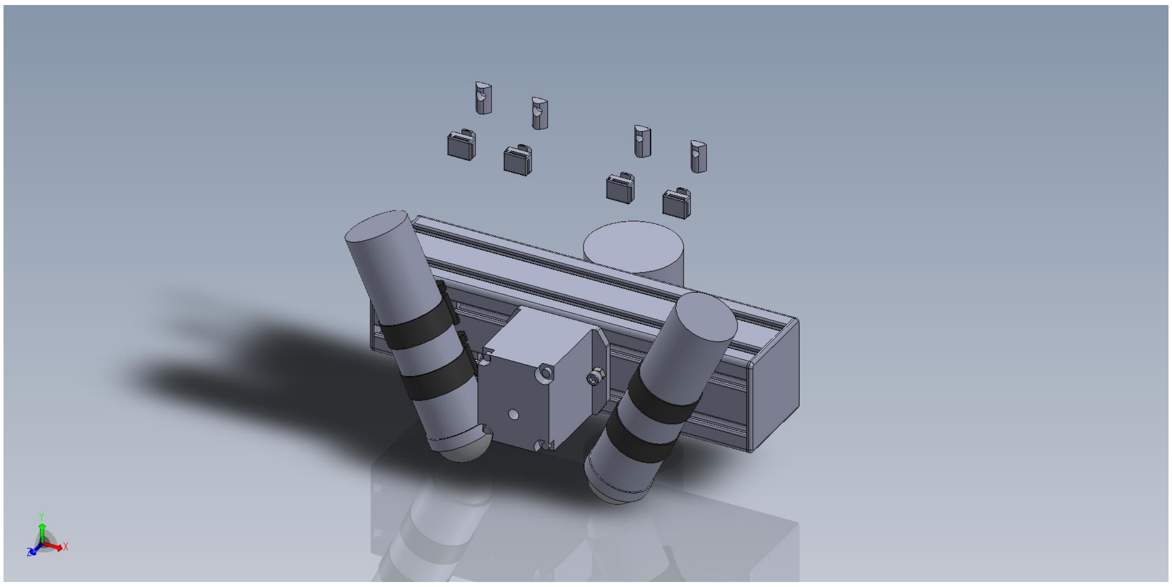
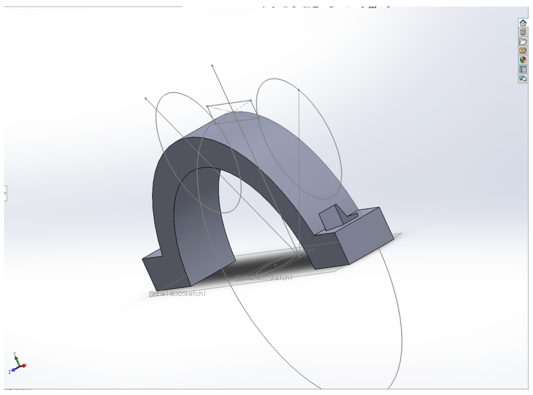
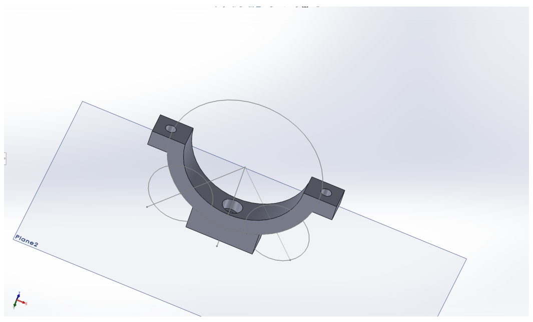
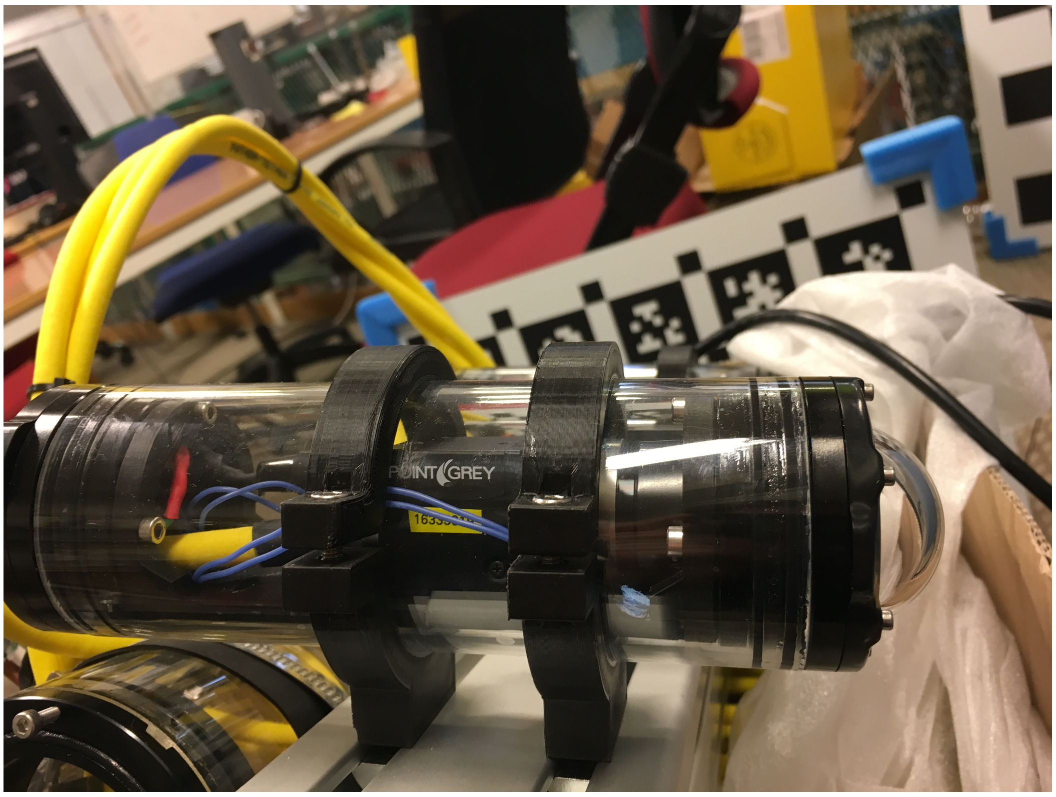


Figure 6.3: Aluminium bearing structure


(a) Top mounting part.


(b) Bottom mounting part.



(c) Mounted camera

Figure 6.4: Connecting the camera cylinders to the bearing structure.

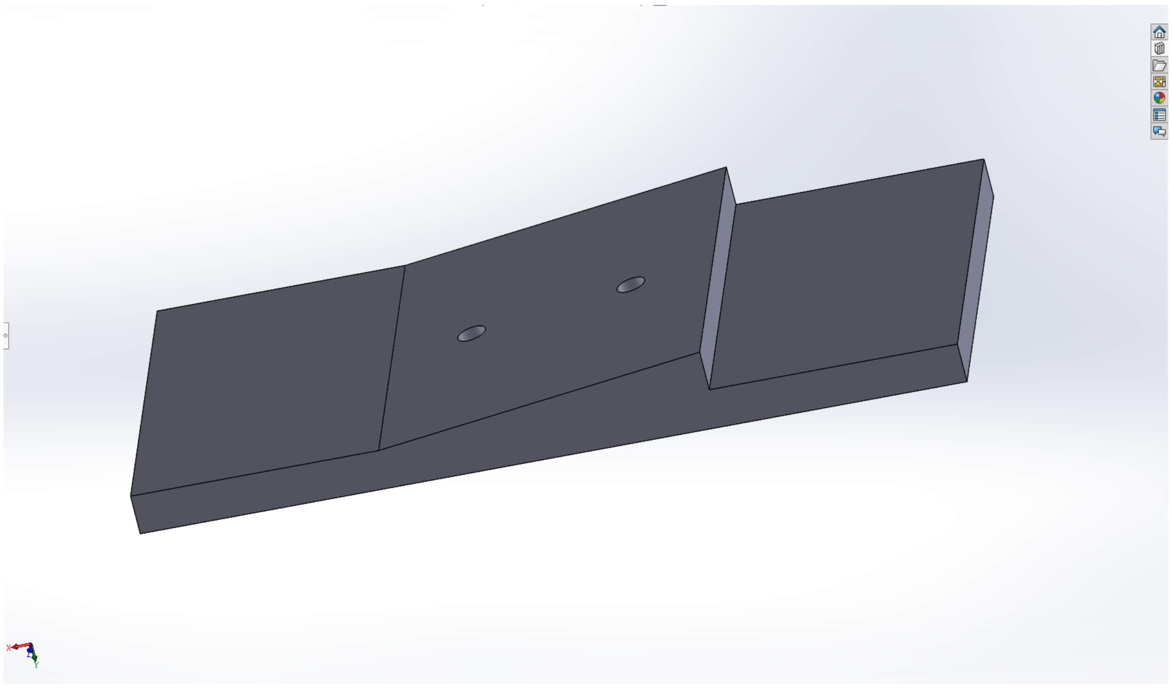


Figure 6.5: Connection point between the hardware platform and other objects.

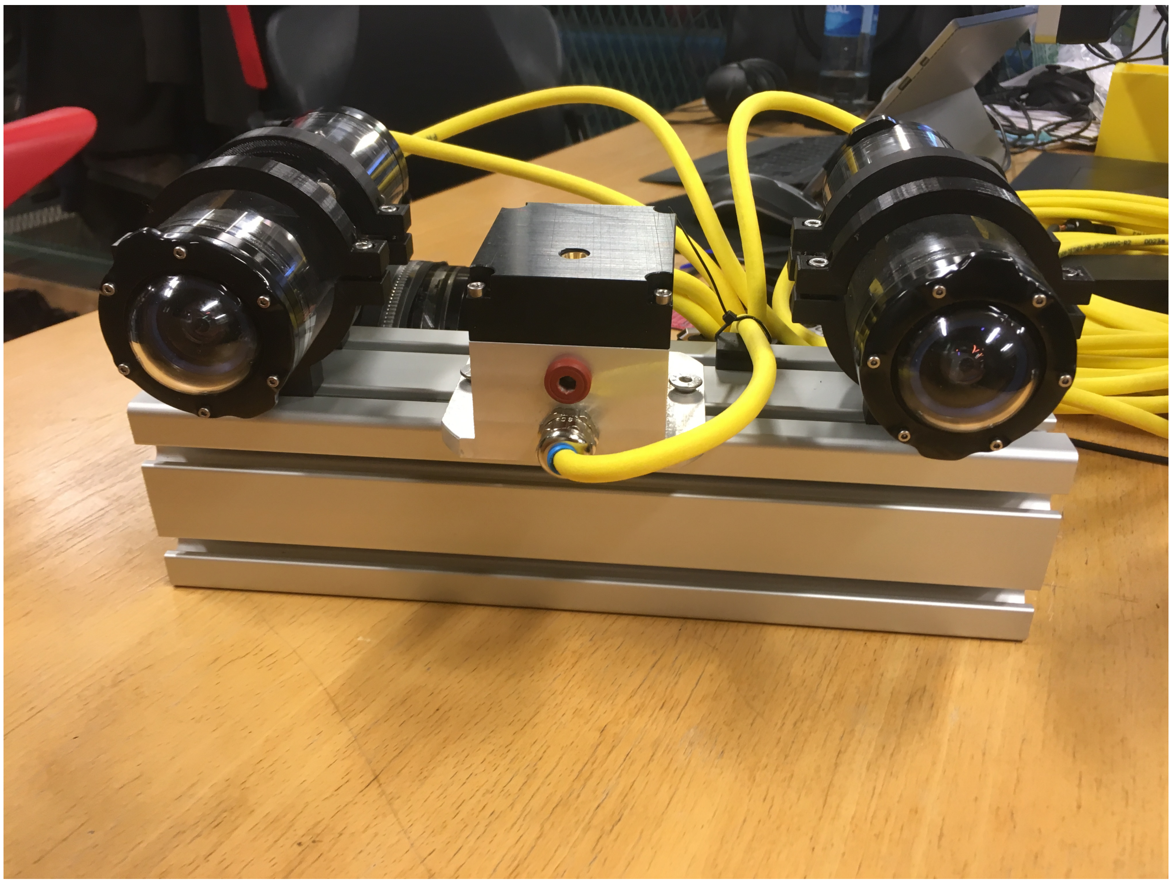


Figure 6.6: Complete hardware setup.

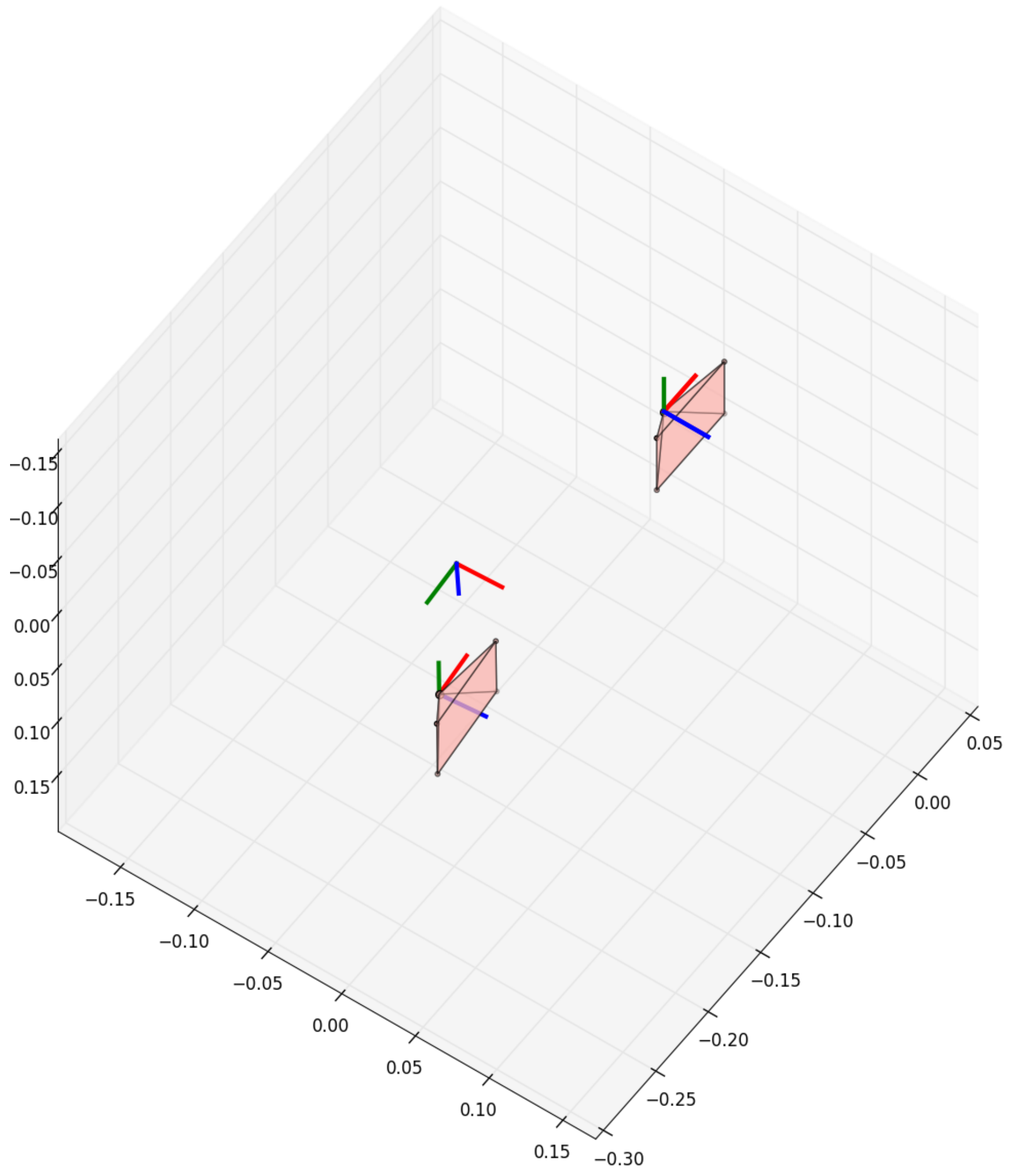
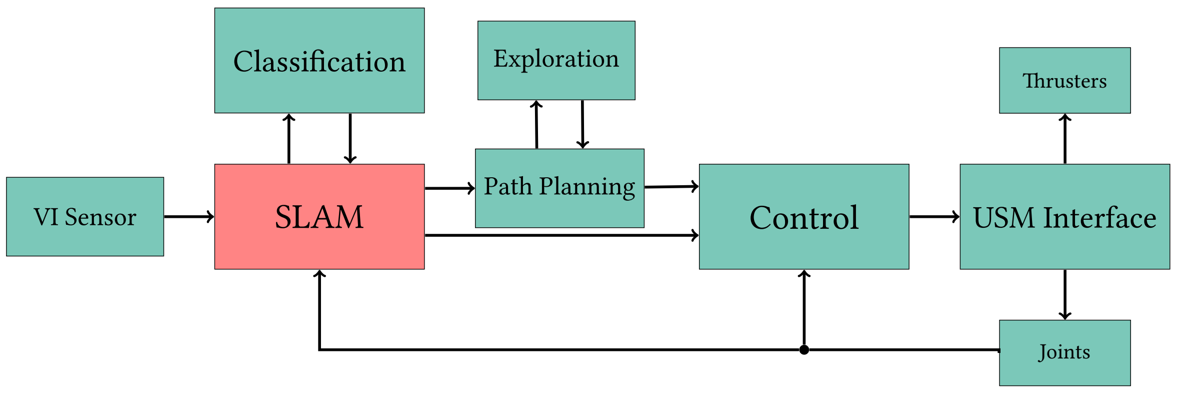


Figure 6.7: Relative sensor placement found by calibration. Pyramids represents left and right camera, with their respective coordinate frame, the middle coordinate frame belongs to the IMU.

Part III

SLAM



7 | Background

Simultaneous localization and mapping is a very hot research topic, it is how we humans navigate and it is now changing how robots are navigating. The underwater environment is a difficult, but interesting area for SLAM. Visual SLAM is a complicated, but useful because of localization:

- Can work in gps denied environments.
- Without drift when not exploring.
- Cm precision.

And mapping:

- Localize relative to surroundings, not some distant coordinate system.
- Allows interaction with surroundings at good accuracy.

There have been a lot of land based SLAM methods, but few are tested underwater. In this thesis we will take a popular open source land based SLAM method, integrate a open source version with IMU measurements integrated and test it underwater as a center peace in a larger autonomous pipeline. Then in section section 16.2 it is fused together with a classification module to generate a semantic map.

This part will first introduce some background information regarding visual SLAM, then quickly go through how the chosen visual method, ORB-SLAM operates, then introduce modifications to incorporate IMU measurements, before lastly present and discuss experiments.

7.1 SLAM Definition

From a very high level point of view there are three required components for SLAM:

- Exteroceptive sensor, that is a sensor which measure external state.
- Link between measurement and internal state.
- Link between individual measurements, or data association. Multiple measurement observing the same landmarks. In other words there need to be a link between internal state and external landmarks through the measurement.

Given enough such links and measurements one can simultaneously solve for both localization and map representation. This can be done with different sensors. One of the most used is LiDAR, another popular sensor for SLAM is camera which will be the focus of this thesis. Then it is often referred to as visual SLAM. Cameras have very fast and accurate place recognition capabilities. Both for finding data association between successive images used for tracking, but also for long distance place recognition used for loop closures. Vision based SLAM methods are generally divided by two dimensions.

- Sparse vs dense. Is simply the discussion on how dense the map should be. Dense methods requires more computational power and needs to use assumptions such as smoothness and colour conservation to simplify the problem. Sparse methods don't need to make assumptions and can run in real time on a relative small CPU, but the resulting map is not directly usable for object avoidance or interactions.

- Direct vs indirect is the discussion on how to perform data association. The question should one compare the image intensities directly or indirectly by first extracting image features.

Definition 7.1. Given a stream $\mathcal{I}_j = \{I_0, \dots, I_j\}$ of images up to time j, a set $\mathcal{L}_j = \{l_0, \dots, l_i\}$ of landmarks and a set $\mathcal{Z}_j = \{z_{i,j}\}$ of observations of landmark l_i in image I_j. SLAM has the goal of finding the optimal 3D positions $\mathcal{P} = \{p_i\}$ of the landmarks and the 6D poses $\eta_j = \{\eta_0, \dots, \eta_j\}$ of the images in real time.

The total state $\mathcal{X}_j = \{\eta_j, \mathcal{P}\}$ is the union of the camera poses for all the images and the positions for all the detected landmarks. Throughout the SLAM part we notate the pose as η_j to simplify notations, the full notation is $\eta^j_{s/m}$, that is the pose of frame s given relative to frame m, for each image I_j. Where s is the sensor frame. For stereo or mono SLAM, s is equal to the rectified, undistorted, left camera frame c_0. While for visual inertial SLAM, s is equal to the IMU frame.

The map coordinate frame m is, unless specified otherwise initialized to s at $j = 0$, that is the time step of the first image I_0 used to initialize the map.

7.2 Front-end

The front end of a SLAM system is mainly responsible for data association. For a complete SLAM system there are two forms of data association, close and far. The close kind is used during tracking, when the images are partly overlapping and the goal is to align them, or extract and match features which can be used to align them. In order to estimate how the camera is moving relatively to the environment, hence tracking the environment. The other kind is used for loop closing, and is intuitively called place recognition.

7.2.1 Place Recognition

Place recognition typically consist of a smart way to compactly summarize a image and store it in a structure which makes it very fast to compare a new image to the existing ones, already in the structure. The most used place recognition methods for visual SLAM are DBoW[39] and FAB-MAP[40]. FAB-MAP was long considered the golden standard of place recognition, it represented images with a bag of words and uses a Chow-Lui tree to learn the covisibility probability between words. DBoW was the first to introduce create bag of words out of discrete descriptors (BRIEF). Thus reducing the feature extraction with an entire order of magnitude.

Bag of words is a technique to discretize images into a set of visual words, thus the entire image can be represented as a bag of such words. The words are created offline from huge data sets, during run time each image is classified based on which words it contains. By storing the combination of words in smart structures such as threes, a new image can very quickly find other images which share many of the same visual words.

7.3 Back-end

The back-end handles all the optimization, it used to be filter based, but recently there was a shift towards nonlinear optimization based methods[41]. This has to do with the success of structure from motion.

7.3.1 Bundle Adjustment

Structure from motion (SfM) essentially solves the same problem as monocular visual SLAM. Whereas SLAM comes from the robotic community, with focus on real time and localization, SfM is a problem from the computer vision field with more focus on the mapping accuracy. Given only a single monocular camera, the goal is to create a 3D representation of the environment. If only one picture is available, this is mathematically impossible. However, if the camera moves and takes pictures from several angles, the problem become solvable. This movement of the camera is what gives the name structure from motion. Bundle adjustment (BA) is the optimization problem at the core of most SfM and SLAM algorithms.

Definition 7.1 (Bundle adjustment). Given a set $\{I_j\}$ of images, a set $\{l_i\}$ of landmarks and a set $\{z_{i,j}\}$ of observations of landmark l_i in image I_j. BA has the goal of finding the optimal 3D positions $\mathcal{P} = \{p_i\}$ of the landmarks and the 6D poses $\eta = \{\eta_j\}$ of the images. It is formulated as:

$$\min_{\mathcal{P},\eta} \sum_{i,j} \left\| z_{i,j} - \Pi\left(p_i | \eta_j\right) \right\|_2^2, \tag{7.1}$$

where Π, defined in eq. (4.14) from chapter 4, projects the 3D point p_i into the 2D image coordinates of image I_j, given the pose η_j, assuming a calibrated camera model.

7.3.2 Factor graphs

Definition 7.2 (Factor graph, [42]). Suppose $g(x_1, \ldots, x_n)$ factors into a product of several local functions, ϕ_j each having some subset X_j of $X = \{x_1, ..., x_n\}$ as arguments; i.e., suppose that

$$g(x_1, \ldots, x_n) = \prod_{j \in J} \phi_j(X_j) \tag{7.2}$$

A factor graph is a bipartite graph that expresses the structure of the factorization (7.2). A factor graph has a variable node for each variable, a factor node for each local function, and an edge-connecting variable node, x_i to factor node ϕ_j if and only if x_i is an argument of ϕ_j.

Factor graphs are a commonly used representation within the SLAM community. Their ability to abstract a complex problem into visible nodes and edges while maintaining all the structural information has proven very useful. There are several solvers which uses factor graphs as a way of representing the optimization problem, effectively as a interface to the solver, the open source g2o [43] and Google's Ceres [44] being the most popular.

As an example we can show how a factor graph optimization problem can represent the BA problem eq. (7.1). From eq. (7.2) we set up the factor graph optimization problem:

$$\begin{aligned}
\min_X g(x_1, \ldots, x_n) &= \min_X \prod_k \phi_k(X_k) \\
&= \min_X \log\left(\prod_k \phi_k(X_k)\right) \\
&= \min_X \sum_k \log\left(\phi_k(X_k)\right),
\end{aligned}$$

by choosing $X = \{\eta, \mathcal{P}\}$, $X_k = \{p_i, \eta_j\}$, $\phi_k = \exp\left(\left\|z_{i,j} - \Pi\left(p_i|\eta_j\right)\right\|_2^2\right)$, we end up with the BA problem in eq. (7.1). The resulting factor graph can be seen in fig. 7.1 for an example with 3 camera poses and 9 landmarks.

7.4 Batch BA

The high complexity of BA makes it difficult to run in real time. The two main difficulties are:

- BA is non-linear and requires accurate initialization.
- BA complexity increases with the cube of the sum of images and landmarks [45].

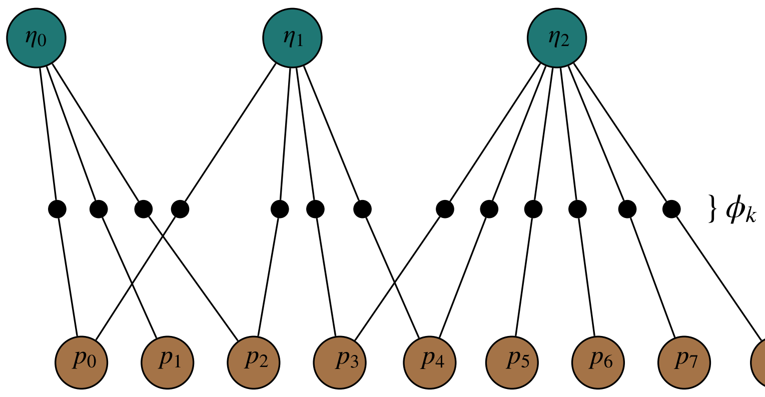


Figure 7.1: Bundle adjustment represented as a factor graph. Green nodes represents the pose η_j of the camera when a image I_J was taken. Brown nodes represents the 3D position p_i of a landmark seen across multiple images. The small black circles represents factors ϕ_k and are equal the exponential of the reprojection error. The result of BA is found by minimizing the product of all the factors.

Thus current embedded hardware is not capable of solving the full BA in real-time. Instead of developing fundamentally new methods for solving the full SLAM problem incrementally, clever engineers and scientist have found ways to use the effective, accurate and established BA solvers for SLAM by dividing the problem into smaller batches, and solving them one by one.

7.4.1 Key-Frames

Key-frames are a subset of images $\mathcal{I}_k \subset \mathcal{I}_j$ with their poses, $\{\eta_k\} \subset \{\eta_j\}$ chosen to summarize the full SLAM problem by not including redundant information, see fig. 7.2. Choosing a

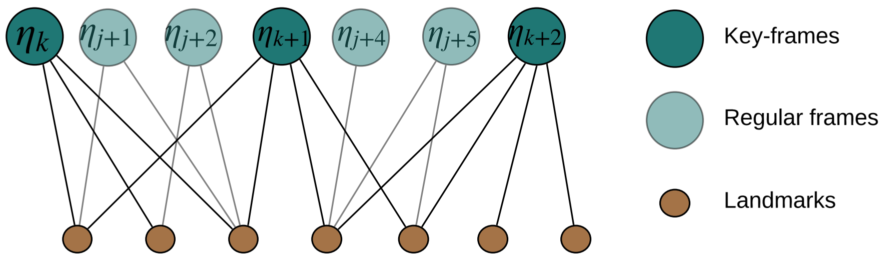


Figure 7.2: Key-frames is a subset of the regular frames chosen to summarize the SLAM problem.

good key-frame selection policy is an interesting problem as one has to balance between a sparse selection and dense selection. Too sparse selection, and there wont be enough common landmarks and the problem will become ill-defined. Too dense selection and the problem is too computationally expensive. Additionally, individually some images contains more landmarks than others and are thus better suited to be chosen as a key-frame.

7.4.2 Windowed Optimization

Instead of performing BA over the entire state X or X_{key}, only a small part of the problem is solved at a time. This is achieved by splitting the states into three subsets, the states X_{excluded} which are excluded from the current optimization, the states X_{included} which are included in the optimization, and the states X_{fixed} which connects the two subsets together. X_{fixed} is included in the optimization, but is kept fixed to ensure a smooth connection between the included and excluded states. See fig. 7.3.

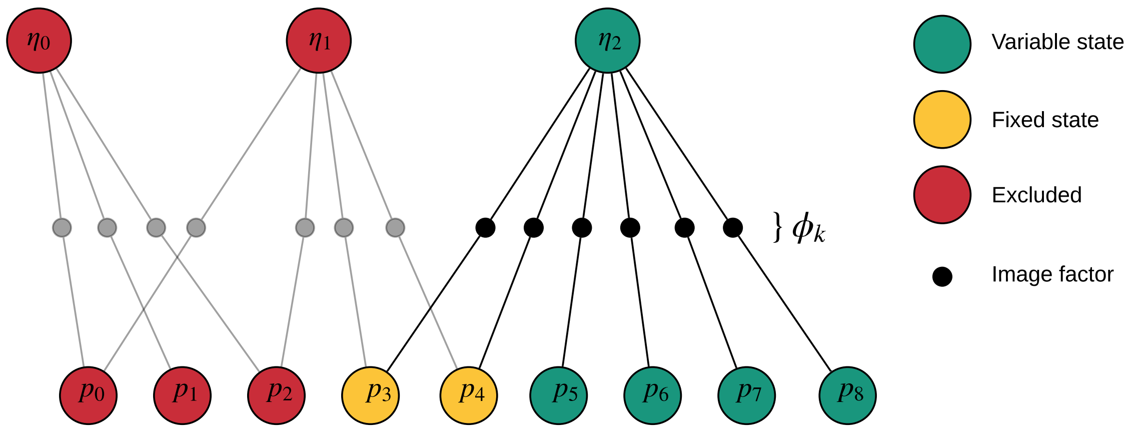


Figure 7.3: Windowed optimization. The yellow states are fixed during the optimization and are only included to maintain a smooth connection between the included green states and the excluded red states.

7.4.3 Fixed-lag-Smoothing

Fixed-lag-smoothing, also referred to as sliding window optimization, solves the full SLAM problem, but only for the last τ time-steps. That is $\eta_j = \{\eta_{j-\tau-1}, \dots, \eta_j\}$. Old states are marginalized out. Filtering within SLAM refers to the problem, where at every time-step j, the goal is to estimate only the current pose η_j and corresponding map $\mathcal{P}_j$. This is the same as fixed-lag-smoothing with $\tau = 1$.

7.4.4 Parallel Tracking and Mapping

First successfully applied in PTAM [45], they created a visual SLAM system which could work in unknown environments. While competing well, in terms of robustness and accuracy, against existing solutions which required a prior model of the environments. The trick was splitting the SLAM problem into two parts, tracking and mapping, which could run in parallel in two separate threads, utilizing the multi-threading capabilities of modern CPUs.

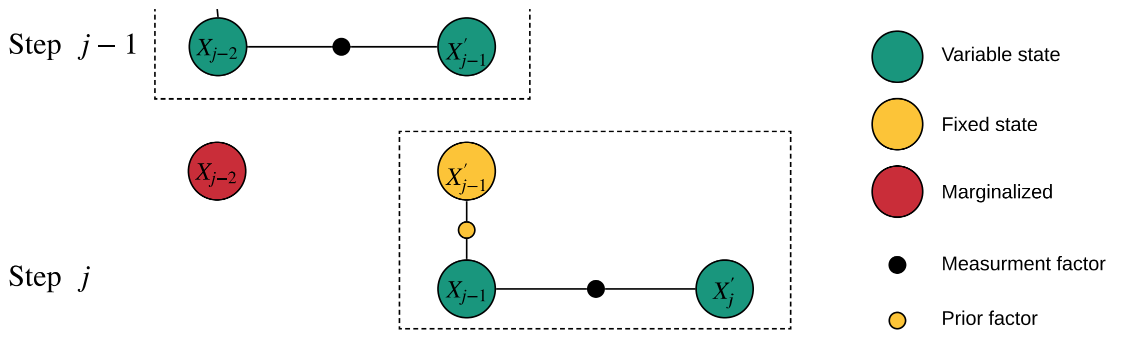


Figure 7.4: Sliding windowed optimization with prior factors. For each optimization step, priors are added as factors between last result and matching, current optimization variables.

Tracking run a fixed-lag-smoothing at frame rate, keeping the map fixed and only including the current state as a variable ($\tau = 1$), marginalizing old states except for a chosen few, named key-frames. The mapping runs one of two windowed optimisations, first local BA, where the included variable states are the most recent key-frame, its 4 closest neighbors and all their map points. The fixed states are all the other key-frames which observes the included map points. Ones the local BA has converged, the mapping thread runs global BA, including all the map points and key-frames until a new key-frame arrives. This model of splitting the problem has proven successful and are used in many state-of-the art SLAM and VO methods [30], [46]–[48].

7.5 Incremental BA

The recent advancements in visual SLAM results from a leap from SfM to SLAM [41], where excellent engineered implementations of BA run in real-time [49], as described above. These batch BA methods represents most of today's state-of-the-art visual SLAM methods [30], [45], [46], [48], [50], [51]. However there are a lot of work on developing specialized SLAM solvers. Current research has three main focuses:

- Incremental smoothing.
- Global optimal solutions.
- Estimated co-variance.

Within the robotics and state-estimation community BA is referred to as smoothing. Unlike filtering smoothing referres to when one includes all the previous states in the optimization problem. Incremental smoothing and mapping (iSAM) [52], and iSAM2 [53], uses a special case of factor graphs named Bayes trees and QR-factorization of the measurement matrix to reorder the variables so that only the states and their uncertainties that are effected by new measurements are included in the optimization problem. SLAM++ [54] is a general framework for incremental maximum likelihood estimation based on a sparse block data-structure with focus on estimating the uncertainty in real-time by updating them incrementally. Which uses information theory measures to integrate only informative and non-redundant contributions to the state representation. ICE-BA [49] improves upon the aforementioned incremental smoothing and mapping methods by, in their own words, "better leveraging the specific block matrix structure in SLAM". Furthermore, they guarantee the minimization of the re-projection function and inertial constraint function during loop closure. With the controversial question "why bundle adjust?", it was proposed to replace BA with a two step procedure[55]. First, the camera orientations are estimated, then, using a quasiconvex formulation, the remaining problem can be solved efficiently and globally optimally. Furthermore, the work by Mangelson [56] illustrates that the Pose-Graph SLAM and Landmark SLAM can be formulated as polynomial optimization programs that are sum-of-squares (SOS) convex. Using this representation, they demonstrate a method for solving planar Pose-Graph SLAM that are globally optimal, thus not requiring any initialization.

However, these newer methods are still in a early stage, thus in this thesis the well tested and popular ORB-SLAM will be used.

8 | SLAM Method

As argued in [3], for underwater visual slam, a indirect method is advantages, because they are not equally dependent on accurate sensor models as direct methods. Sparse methods are considered more accurate, and robust for the same computational power compared to dense methods [29], [51]. Lastly ORB-SLAM is open source with many forks and improvements. For further argumentation, the reader is refered to [3].

8.1 ORB-SLAM

ORB-SLAM is a open source, popular, indirect sparse SLAM method. It is based on PTAM [45], where in addition to the tracking and mapping thread a loop closing thread is added.

8.1.1 Connectivity Graph

In order to keep track of the covisibility and connections, ORB-SLAM operates with three graphs.

- Covisibility graph, each key-frame is a node, and they are connected by a undirected edge if they share common map points. The weight of the graph is equal to the number of common observations.

- Spanning three, this is a subset of the connectivity graph, and is a three with the root at the first key-frame, connecting all the nodes with a minimal set of edges.

- Essential Graph, this is the union of the Spanning three, Covisibilty graph edges with more than 100 observations and loop closure edges.

8.1.2 Place Recognition

ORB-SLAM uses DBoW2[39] for place recognition, it is a custom modification from DBoW, and the words are built from ORB descriptors. In addition to utilizing the DBoW three to quickly look up similar images, ORB-SLAM also uses the three as a speedway for searching for ORB matches between current frame and last key-frame.

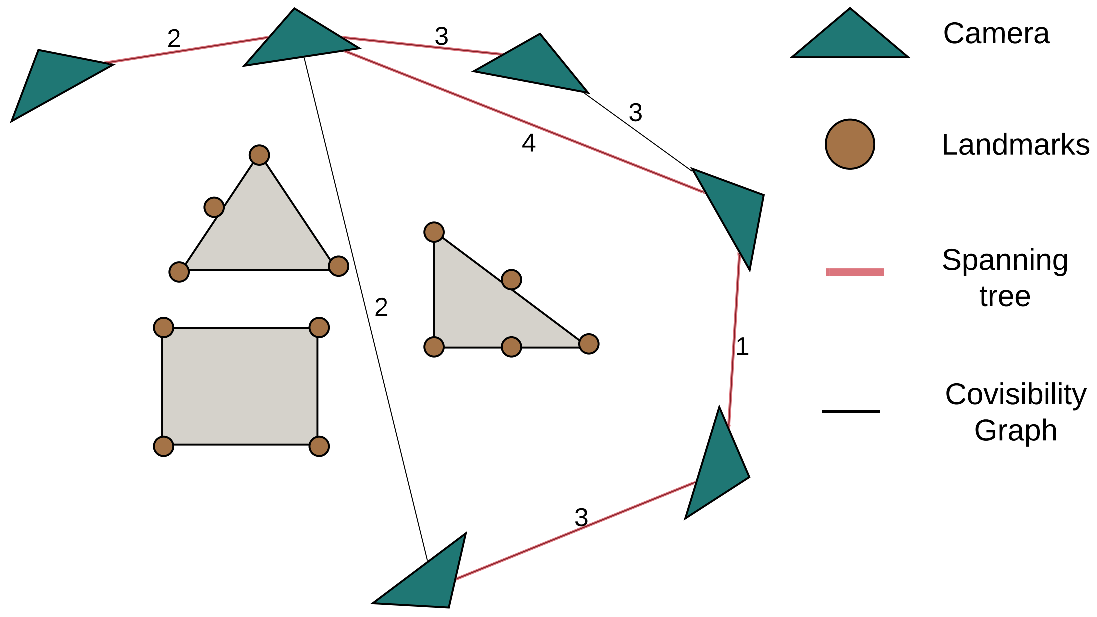


Figure 8.1: Covisibility graph. The number represent how many common observations the two key-frames share.

8.1.3 Initialization

For monocular SLAM, the map initialization is a difficult task. The way it is done is to wait for the camera to move in a direction which is parallel to the image plan, so that two images taken a little apart from each other can be used as one stereo pair. Then the same geometry as explained in section 4.4.1 applies. However the transformation between the cameras are not known in advanced. To get around this ORB-SLAM uses one of two models to represent the problem. Either the Homogeneous matrix H, or the fundamental matrix F. in order to use the Homogeneous matrix, the scene has to be a planar surface, while no such requirements are needed for using the Fundamental matrix. After estimating both matrices using RANSAC, one of them are chosen based on the reprojection error. If the parallax is large enough (at least 1 degree) the chosen model is used to generate the initial map. Lastly a global BA is ran over the newly created map.

8.1.4 Tracking

The tracking thread runs in real time at frame rate. During normal operation the map is initialized and tracking was successful for last frame, then the tracking performs the following operation on every frame: Detect FAST corners, trying to distribute the corners evenly over the picture, by dividing the picture into grids and using the best corners from each grid, instead of the best overall corners. Then ORB descriptors are computed on the

FAST corners.

Estimate the pose η_j of the current frame in three steps:

1. First get a rough initial estimate η'_j assuming a constant velocity motion model from the last two poses η_{j-1} and η_{j-2}.

2. Using η'_j as initialization find a improved estimate η''_j by by performing windowed motion only BA with the map points which where found in the last frame and the current pose, keeping the map points fixed.

3. Finally, using η''_j as initialization, optimize the pose again using all the map points.

If the tracking is lost the frame is instead converted to a bag of words, then used to search for matches among all the key-frames, using the DBoW2 library. If a match is found the relative position between the matched key-frame and the current frame is calculated and tracking continues.

The last step of the tracking thread is to decide if the current frame meets the necessary criterias for becoming a key-frame:

1. More than 20 frames must have passed from the last global relocalization.

2. Local mapping is idle or more than 20 frames have passed from last key-frame insertion.

3. Current frame tracks at least 50 points.

4. Current frame tracks less than 90% points than the Key-frame with the most shared map points.

Many key-frames are initially inserted to handle fast turns and quick movements and then later removed from the map.

8.1.5 Mapping

The mapping thread runs in close to real time. It runs in parallel to the tracking thread, so that while the tracking thread is tracking the current map, the mapping thread is updating and maintaining the map. While the tracking thread runs its routine on every frame, the mapping thread only runs the routine on every key frame. The mapping thread is interrupted by the tracking thread if there has been 20 or more new frames since last key frame. Thus the mapping thread should not use more than 20*FpS s per key-frame.

For every new key frame K_j the mapping thread begins by inserting K_j into the covisibility graph. Then potential outlier map points are filtered out by two criteria, where the first only are checked during the first three key frames after the map point was created:

- The tracking must find the point in more than 25 % of the frames in which it is predicted to be visible.
- If more than one key-frame has passed from map point creation, it must be observed from at least three key-frames.

Next new map points are created by finding and matching new key-points between the current key-frame and the connected key-frames. Then the map is updated by running widowed BA over all the connected key-frames $\mathcal{K}_c$, and their map points. All neighboring key-frames are included in the optimization, but remains fixed. The Levenberg-Marquadt method implemented in g2o [43] and the Huber robust cost function are used. Lastly redundant key frames are removed according to one criteria:

- 90% of the map points have been seen in at least three other key-frames in the same or finer scale.

8.1.6 Loop closure

After the mapping thread is done with integrating the latest key-frame I_k to the map, the loop closure thread start searching for loop closures. To ensure there are no false positive a strict set of rules are followed. First the visual similarity between I_k and its neighbors in the covisibilty graph are calculated and the lowest one is used as a minimal similarity criteria. Thus any key-frames with a lower similarity than the least similar of the closest neighbors are discarded. Further any candidates must be connected to at least two other loop candidates in the covisibilty graph. For each candidate a similarity transformation, that is a 7 DoF (including scale) transformation, is computed between the current key-frame I_k and the loop candidate I_l. If there are enough map point matches at the new location, the loop closure is accepted. The error is distributed over the Essential Graph, by performing a similarity transformation optimization over graph distributing the error and correcting scale drift. Finally the map points are corrected by transforming them using the exact same transformation that where used on their key-frames.

8.2 Visual Inertial ORB-SLAM

The creators of ORB-SLAM extended it to include inertial measurements, and thus published the first paper describing a complete Visual Inertial SLAM system with loop closures, localization, reusable maps and pure localization mode [28]. To integrate IMU measurements, there were most changes in the tracking and mapping threads because two additional states;

- Linear velocity $\vec{v}_{s/m} \in \mathbb{R}^3$

- IMU biases $\vec{b} = \{\vec{b}_{acc}, \vec{b}_{gyro}\}$ from the sensor model in eq. (4.1).

The state associated with each image I_j is expanded from only including the pose η_j to also include the new states. This updated state is named the odometry state $\Omega_{s/m}$ where s is the sensor frame, which is fixed relative to the body frame b, and m is the map frame which is fixed relative to the NED frame, which is simplified to assumed inertial. It is defined as $\Omega^j_{s/m} = \{\eta^j_{s/m}, \vec{v}^j_{s/m}, b^j_{acc}, b^j_{gyro}\}$ and throughout the SLAM part the notation is simplified to: $\Omega_j = \{\eta_j, \vec{v}_j, b_j\}$.

8.2.1 Tracking

The constant velocity model is replaced by integrated inertial measurements, giving a much more accurate initial pose estimate η'_j. IMU factors are included in the local motion only BA, which runs in two different modes dependent on if the map has been updated by the mapping thread (Local BA) or the Loop closing thread (Loop closure). If the map was recently updated, the local motion only BA includes the following states:

- variable states $X_{included} = \{\Omega_j\}$,
- fixed states $X_{fixed} = \{\mathcal{P}_{local}, \Omega_k\}$

where Ω_k is the odometry state for the last key-frame.

If the map have not been updated since the last time the tracking ran, the local motion only BA is implemented as a sliding window optimization with

- variable states $X_{\text{included}} = \{\Omega_{j-1}, \Omega_j\}$,
- fixed states $X_{\text{fixed}} = \{\mathcal{P}_{local}, \Omega'_{j-1}\}$

where Ω'_{j-1} is the solution from the prior optimization and is connected to Ω'_{j-1} via a prior factor, as shown in fig. 7.3. The reader is referred to [28] for a detailed info in how the error terms are defined.

8.2.2 Mapping and Loop Closing

The mapping also includes the extra IMU states in the local BA, where the last n key-frames are included as variable states, to keep them IMU states connected to the excluded states, a extra state Ω_{n+1} is included as a fixed state. Otherwise the mapping and Loop Closing remains the same. During initialization, the pure visual SLAM part initializes first, then the estimated trajectories are used to estimate the IMU states, one at a time, before the Loop Closing thread runs a full global BA, with all states, to properly initialize the IMU states.

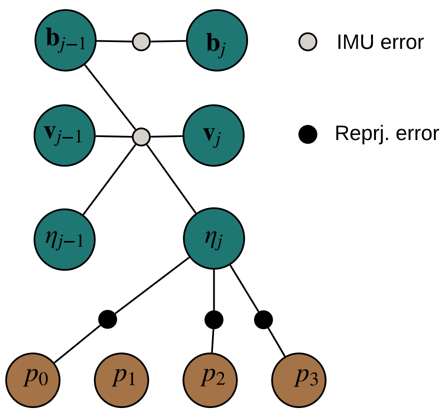


Figure 8.2: Widowed optimization variables and error terms.

Algorithm 1 VI-Tracking (I_j, $\{\vec{a}_{IMU}\}_{j-1,j}$)

```
    Input: New Image $I_j$, IMU measurements since last image $\{\vec{a}_{IMU}\}_{j-1,j}$
    Output: Current estimated odometry state $\Omega_j$
1:  ORB = ExtractORB($I_j$)
2:  $\Omega'_j$ =PreIntegrate($\{\vec{a}_{IMU}\}_{j-1,j}$)
3:  $X_{init}$ = GuidedSearch($\Omega'_j$,ORB)
4:  if Map has changed then
5:      $X_{included} = \{\Omega_j\}$
6:      $X_{fixed} = \{\mathcal{P}_{local}, \Omega_k\}$
7:  else
8:      $X_{included} = \{\Omega_{j-1}, \Omega_j\}$
9:      $X_{fixed} = \{\mathcal{P}_{local}, \Omega'_{j-1}\}$
    $\Omega_j$ = WindowedBA($X_{included}$, $X_{fixed}$, $X_{init}$)
        return $\Omega_j$
```

8.3 Implementation

ORB-SLAM and ORB-SLAM2 are both released as open source code, which is a huge factor of why it have become so popular. There are many forks and modifications to ORB-SLAM, and choosing which one to build upon is a critical step. One key factor to keep in mind is to try to build upon and improve the work of other people in the open source community. In this thesis we uses the only one with a ROS wiki page[1], as it is recently created, are being maintained by AppliedAI[2] and have had several updates during the last months. It is already loosely integrated with ROS.

The ROS node starts the tracking, thus the ROS node and the tracking module run in the same thread, while the mapping and loop closure threads are spawned as individual threads using c++11 thread functionality, see fig. 8.3. The ROS node subscribes to image

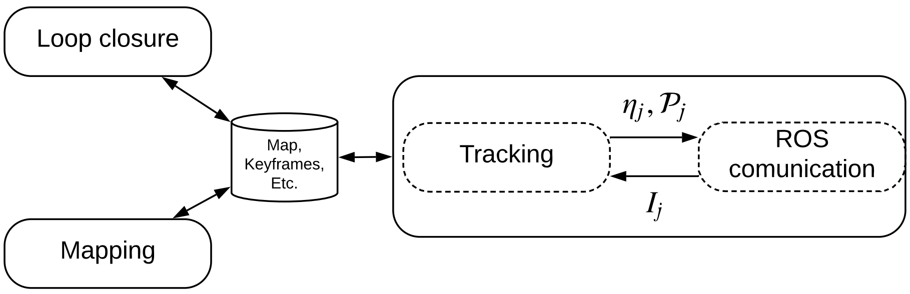


Figure 8.3: ORB-SLAM implementation. ROS communication and tracking runs in the same thread.

topics and when a new image I_j arrives it runs the tracking function. ORB-SLAM performs tracking and returns the estimated pose η_j, and the current map point cloud $\mathcal{P}_j$.

8.3.1 IMU Integration

The creators of ORB-SLAM published a paper [28] of how to integrate IMU measurements into their visual SLAM system as described in section 8.2, but did not release the source code. However there have been a couple attempts of implementing the method described in the paper as open source code, with the most successful being the github repository LearnVIORB[3]. The Visual Inertial SLAM implementation in this thesis is forked from LearnVIORB with no new features and only three performance improvements:

1. Upgrade the included, outdated g^{2}o version to the newest official version.

[1]`http://wiki.ros.org/orb_slam2_ros`
[2]`https://appliedai.de/`
[3]`https://github.com/jingpang/LearnVIORB`

2. Replace several raw pointers with C++11 smart pointers and added null-pointer checks.

3. Change from polling to callback based sensor data collection.

The first improvement being the most important as the included g^{2}o version, when compiled with newer C++ compilers, resulted to numerical errors and segfaults every time a new map was initialized. But also the most time consuming as most of the optimization class had to be updated to work with the new g^{2}o API. Similarly the second improvement was an attempt to reduce the number of segfaults occurring due to raw pointers pointing to empty memory. The last improvement was more in style with the ROS guideline of how to handle messaging between nodes.

Finally the visual inertial implementation where merged into the ROS integrated version described above. Where the estimated velocity and pose where combined and published as a odometry message.

Additionally a modular integration with the object classification module where implemented as described in section 16.2. The map points where assigned a class and published with a class and class score as a PointCloud2 ROS message.

9 | Experiments

9.1 Setup

9.1.1 Datasets

Several sequences have been collected as ROS bags with the visual sensor described in part II. Each sequence contains the following synchronized measurements:

- Stereo camera 644x482 pix images at 20.83 Hz.
- 3 axes accelerometer measurements at 125 Hz.
- 3 axes angular rate measurments at 125 Hz.
- Pressure measurements mapped to depth [m] at 125 Hz.

Most sequences are from the pool at the Eelume lab, but there are also some from the Dora dock, an old submarine dock. The Eelume pool is approximately 6 m by 3.5 m, and 1 m deep. The datasets were gathered while walking in the pool holding the VI-sensor with a stick. As the pool normally has very little contrast, several small, black metal objects where distributed across the pool. In addition, an Aprilgrid was placed at the bottom in one end of the pool. This was primarily to be used to calculate a ground truth, but also served as a easy starting point for the SLAM algorithm to generate a good initialization. The chosen sequences from the Eelume pool are listed in table 9.1. The first three (lin x) where recorded

Table 9.1: Selected Dataset from the Eelume pool.

	Length [m]	Time [s]
lin 1	19.2	58
lin 2	14.1	31
lin 3	13.9	33
rot 1	7.4	30
rot 2	8.3	27
rot 3	7.9	24
rot 4	8.1	23
map 1	53.1	271
map 2	41.3	235

while trying to mainly move in the XY plane, with as little rotations as possible, and with increasing speed. The following four (rot x) where recorded in a similar manner, but with one specific rotation where the rotation rate increased for each run. The last two where long recordings where the VI-sensor was moved across the entire pool, in order to build a complete map of the pool bottom.

9.1.2 Computer

All the results in this chapter where generated by running the ROS bags in real time speed on a stationary computer with a Intel core i7 CPU.

9.1.3 Tracking Parameters

The following tracking parameters were used during the runs:

- Number of ORB features per image: 1000.
- Minimum FAST corner response: 15.
- Maximum FAST corner response: 25.

The biggest change being the increased requirements for the FAST corner response. This means that the required contrast required for a corner to be used as a feature is increased. The reason was to filter out corners detected at shadows caused by either the person walking in the pool holding the camera, or shadows caused by waves in the surface distorting light, which shows as moving shadows at the bottom.

9.2 Visual Odometry Accuracy

9.2.1 Score System

In order to numerically evaluate the SLAM system, the chosen sequences where started and stopped above the April grid, which was laying at the bottom of the pool. For the first and the last image of the sequence the poses, named η_o, η_{gt} respectively, of the camera relative to the April grid where calculated based on the camera model, and the known layout of the April grid using Kalibr. Then these two poses where rotated so the first pose η_o became equal to the map coordinate frame m from the SLAM results, as m is equal to the pose of the first frame η_0. After transforming η_o and η_{gt}, so that $\eta_o = \eta_0$, assuming the SLAM algorithm estimates the trajectory perfectly, the last estimated pose, η_{slam} from the SLAM algorithm should be equal to the last pose η_{gt} estimated from the Aprilgrid. Then the overall translation error t_{abs} is calculated as the RMSE between translational parts of eta_{slam} and η_{gt}. As monocular SLAM do not estimate scale, the scale was estimated by first extracting the ground truth pose from all the frames which observed the calibration board

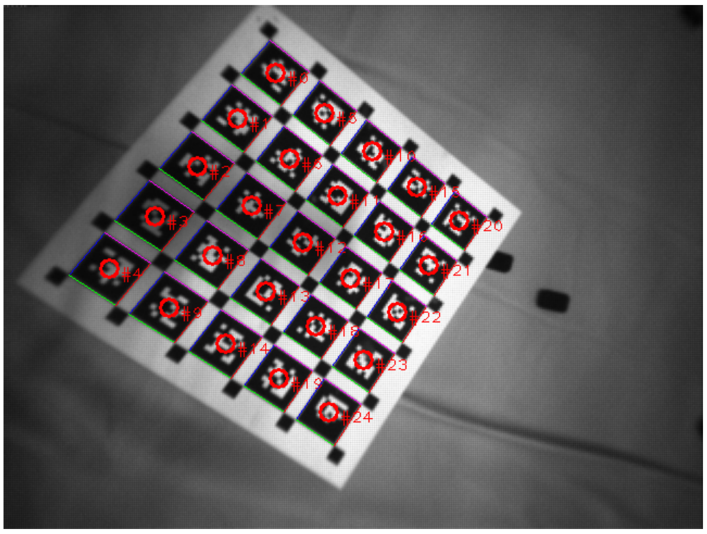


Figure 9.1: Aprilgrid detected and used to generate ground truth.

in the beginning of the sequence then performing a similarity transform optimization to minimize the RMSE between the ground truth poses and the matching estimated poses. For this test to be any interesting loop closure was turned off. With loop closure turned on, the SLAM system recognized the April grid and performed a loop closure, distributing the error across the entire trajectory, resulting in a near perfect score every time. Thus this test essentially estimates the accuracy of the visual odometry part of the SLAM system; how much error is accumulated over time.

9.2.2 Results

The calculated results using the scoring system described in section 9.2.1 from some selected sequences are displayed in table 9.2.

Table 9.2: Visual odometry results from the Eelume pool.

	Visual GT scaled	VI	VI GT scaled
lin 1	0.72	0.93	0.87
lin 2	0.63	0.69	0.65
lin 3	X	X	X
rot 1	0.33	0.30	0.27
rot 2	0.47	0.40	0.38
rot 3	X	0.58	0.54
rot 4	X	X	X

As the sequences are rather short, the IMU initialization was kick-started by supplying the estimated IMU biases and gravity size from previous runs, thus the IMU initialization completed only 5 seconds after the map were initialized.

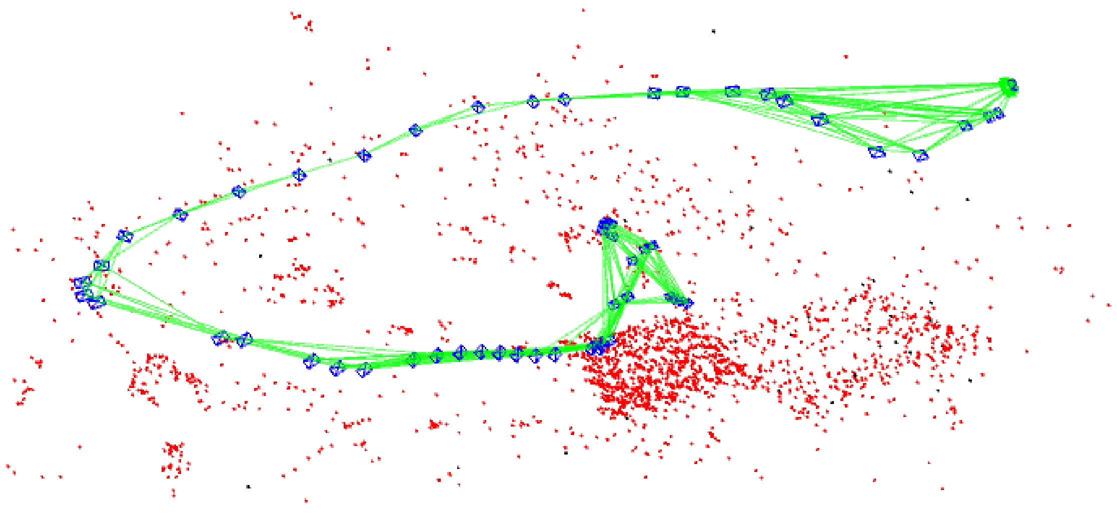


Figure 9.2: Monocular SLAM. The blue frames represent key-frames, the green edges represents the connectivity graph and the red points represents the map points.

fig. 9.2 shows one of the monocular runs from rot 3 sequence. If one looks closely to the map points one can see the calibration board beneath the start and end poses. There has been a slight scale drift during the run and as a result the calibration board near the end has become larger than in the start.

9.2.3 Discussion

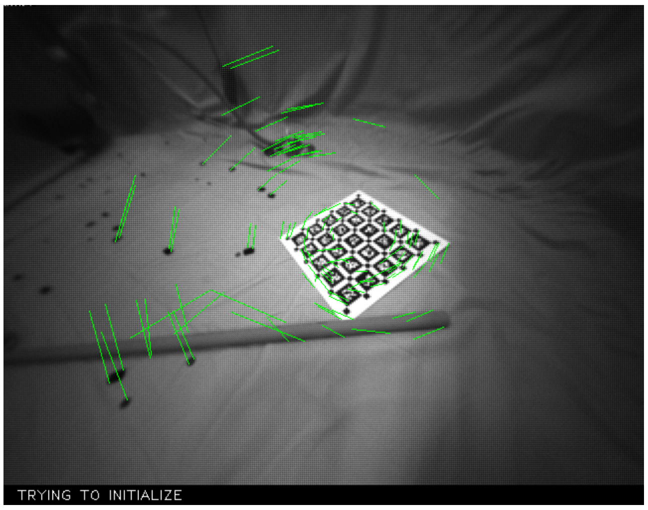
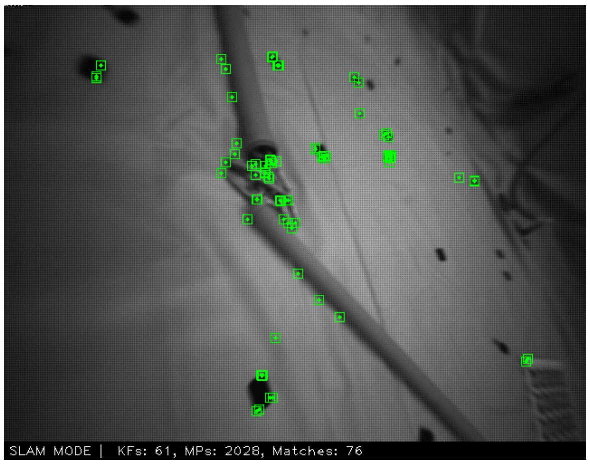


Figure 9.3: ORB-SLAM monocular initialization. The green line represents the tracked features used to calculate the movement of the camera.

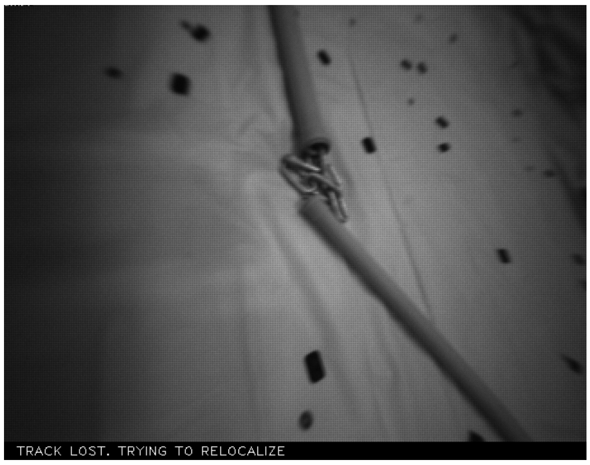
The first thing to note is that the map initialization requires some translational movement in parallel to the image plane, so called parallax, and rotational movement is normally not enough. fig. 9.3 shows a image where ORB-SLAM is unsuccessfully trying to initialize with a rotational movement. This makes sense as the system requires parallax in order to get a large enough baseline to initialize the map.

In [29] the authors of SVO2 compare their result to ORB-SLAM2 mono, running without loop closure in the ICL-NUIM dataset. There ORB-SLAM2 achieves a RMSE between 0.02 and 0.37 m, unfortunately they do not inform about the length of the run, but the room used is a little smaller than the Eelume pool and with plenty of features. Further they run in air, thus having very accurate camera models. Considering these facts, the errors shown in table 9.2 are below what is achieved in-air, but not by very much, particularly considering the difficulty of the sequences. Further it is worth noting how the pure visual system performs better than the visual inertial system on the lin 1 and 2 sequences. Including Inertial measurements seems to give worse results than without. However in seq. rot 1-3 the VI-sensor outperforms the purely visual system. This is expected, because the monocular camera system are not able to estimate its pose accurately during pure rotations [55]. However the IMU is very good at estimating rotations, thus this is a excellent result demonstrating the good fit between the two sensors. The reason the pure visual SLAM is outperforming VI-SLAM in the linear sequences might be due to the fact that the VI-SLAM system introduces extra states to estimate, thus more room for error. Assuming good parallax, (linear movement parallel to image plane), and plenty of good features, the visual system might have everything it needs, and thus the BA will produce very accurate results. fig. 9.4 displays a little experiment to get a estimate of how fast rotations the system can handle in ideal environments. The camera is moving far from the bottom, thus keeping many high contrast corners in view, giving plenty of features to track. The camera was moved into position looking another way, see fig. 9.6b, then suddenly turned to look into the new scene. If it was turning to a already mapped scene, then even if lost tracking it would immediately relocalise in the existing map. section 9.2.3 displays the absolute angular rate from run rot 3 and 4, and the peaks represents the point when the turn happened. Thus, in this particular setting the VI-sensor can handle up to around 2.3 rad/s. This is obviously highly dependent on the frame rate, scenery, shutter speed etc. But it gives a rough estimate of how well the VI-sensor can perform in the Eelume pool.

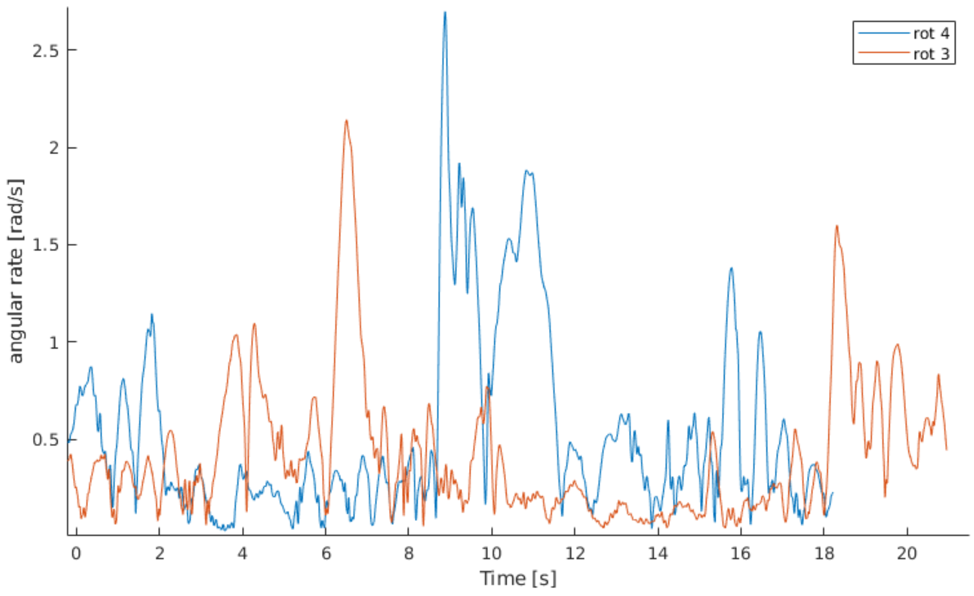
The scale drift due to lack of parallax become very clear in fig. 9.5a, where after initializing a little later than the VI-SLAM system, the scale drift enormously during the turn in the upper left corner. After that it keeps on going, but the scale is now much lower, resulting in what appears to be almost no movements. On the other hand the VI-system handles the turn very well.



(a) Sharp turn in seq. rot 3.

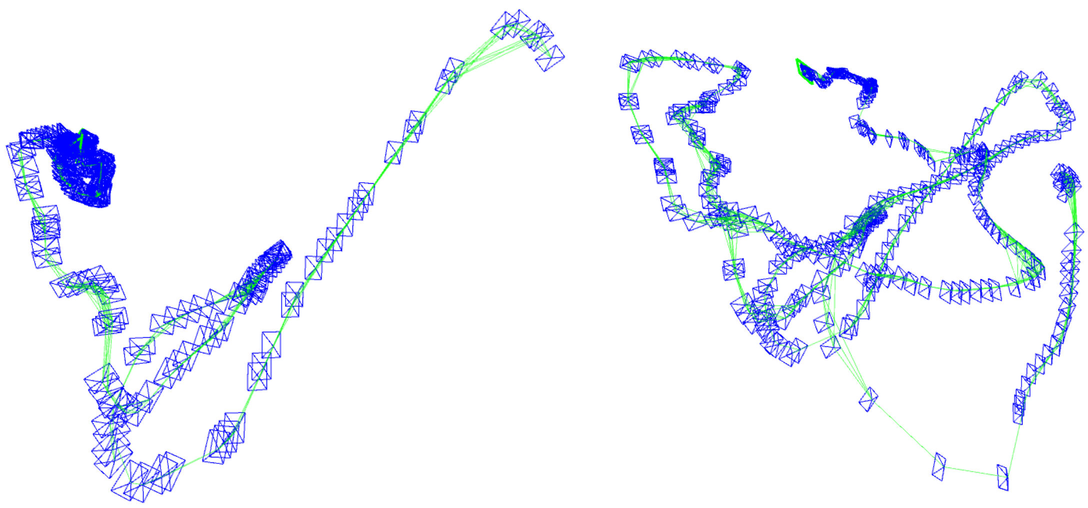


(b) Sharp turn in seq. rot 4, track lost.



(c) Angular rate during seq. rot 3 and 4.

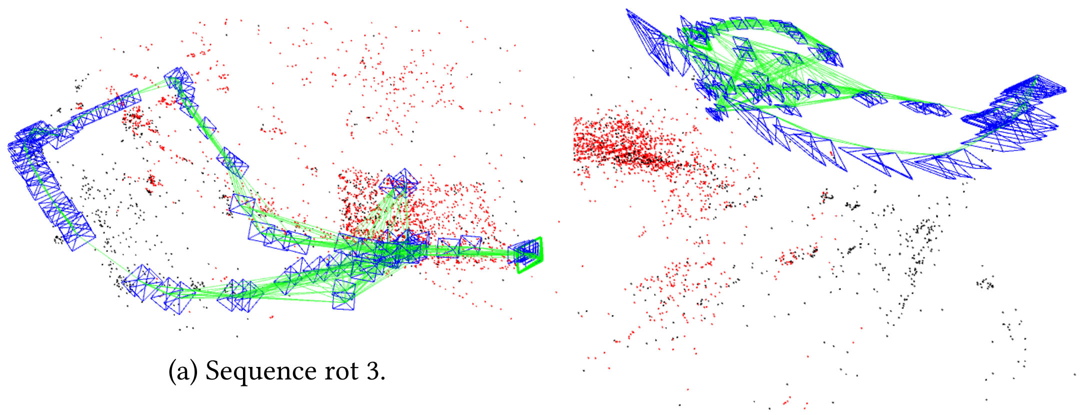
Figure 9.4: Rotation experiment to get a estimate for maximum rotations before lost tracking.



(a) Monocular SLAM.

(b) Monocular Inertial SLAM.

Figure 9.5: Running the map 2 sequence. The Monocular SLAM run initializes a little later than the VI-SLAM and suffer from severe scale drift after a pure rotation in the top left corner.



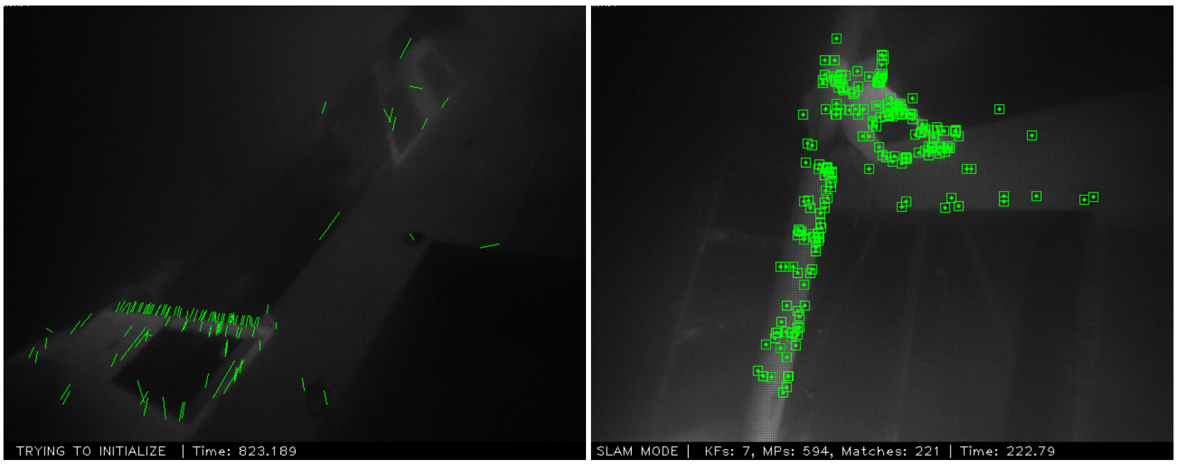
(a) Sequence rot 3.

(b) Sequence rot 2.

Figure 9.6: Dense key-frames selection during fast turns.

9.2.4 Dora Dock

To get test the SLAM system in more realistic environments, several data sequences where collected from Dora submarine dock. The environment is very large and under a half roof, thus quite dark. It is a structured, man made, underwater environment, with pipes and metal structures. From fig. 9.7a it can be seen an example of where the SLAM system struggle to initialize in these difficult conditions. Even though the hardware platform have the capabilities to mount a light source, due to time constraints there where not time to incorporate them for this project. This resulted in the hardware platform being highly dependant upon external light sources. The poor lighting conditions seem to be a significant contribution to why the system struggled. After some tuning fig. 9.7b show an example of where the SLAM module managed to initialize and maintain tracking while looking at the same area.


TRYING TO INITIALIZE | Time: 823.189
SLAM MODE | KFs: 7, MPs: 594, Matches: 221 | Time: 222.79

(a) ORB-SLAM2 trying to initialize in very difficult environments.

(b) Tracking in difficult environment. Green squares represent tracked map points.

Figure 9.7: Examples from the testing done at the Dora Dock facilities.

9.2.5 Tracking Time

The tracking time determines the maximum frame rate of the SLAM system as well as in combination with the image latency, determines the overall pose estimate latency. In table table 9.3 the tracking times for different scenarios are displayed. During runs where there are very many key-frames looking at the same area, the mono tracking could reach as high as 41.4 ms over some time periods.

Table 9.3: Tracking time during different modes.

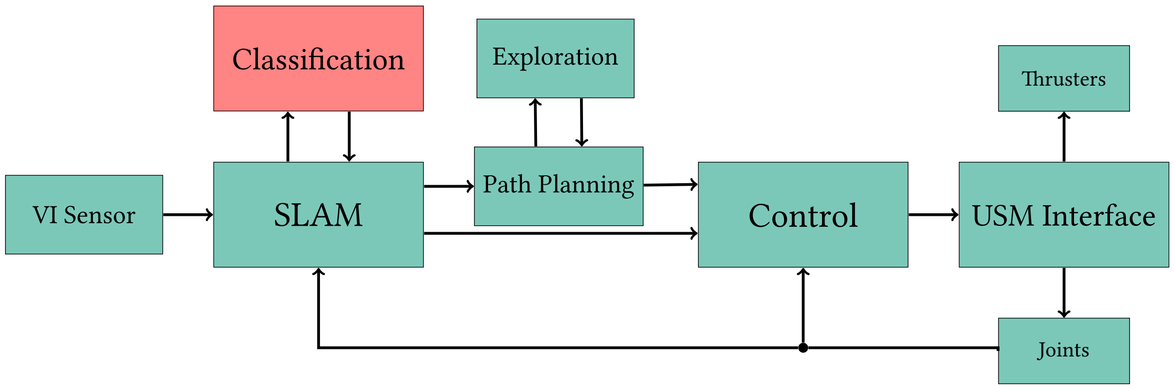
	Mean [ms]	Std [ms]
Init mono	61.3	5.1
Mono	26.4	4.2
Init VI	35.0	7.9
VI	30.0	7.2
Localisation	38.2	4.0

9.3 Summary

This part presented a monocular inertial SLAM system implemented to work in an underwater environment. The system was tested in the controlled environment of the test basin and for a more complex case, the Dora Dock. The system performed well in test basin. However, the system struggled in the more realistic environment and did not manage to complete any of the datasets from the Dora Dock. From the test basin datasets it was determined that the robustness of the monocular SLAM system is increased when augmenting it with an Inertial sensor. This increase in robustness was especially noticable with respect to rotations and scale drift.

Part IV

Classification



10 | Computer Vision

In order to provide contextual awareness about what is in the environment of the USM, it is natural to look to computer vision. Computer vision explores how machines can understand what is present in the environment in a similar manner to how humans do. When a person sees an object, such as a house, this person would instantly recognise the object as a house. This process is very complex, with many different proposed solutions, which results in a large field of research. It is, therefore, necessary to reduce the scope somewhat. Consequently, this thesis will mainly focus on the problem of finding an underwater pipe, though it should ideally be able to provide a strategy to find other objects of interest as well. Objects of interest will be referred to as separate classes.

This thesis will look at how several subsystems can work together to achieve an overarching goal. With this in mind, the goal of the classification section is to find a solution that can be deemed sufficient to solve the problem of finding pipes, rather than the best performing solution. Once a sufficient solution has been found, the work will be concentrated on integrating this solution with the other system components.

As described in section 2.2, there are three different scenarios that are relevant for this system. These conditions are quite different and have their own purposes. The first scenario was the simulator. This was used to test the system and see how it operated in known, highly controlled environments. When testing in the simulated environment, a simple and robust computer vision algorithm can be used to mimic a fully operational classification system. For this reason, a computer vision algorithm that finds any instances of a specified color range was developed (see algorithm 2). This algorithm is implemented in Python and utilizes the open source library OpenCV, which is placed in a ROS wrapper. This algorithm receives image messages defined in a ROS format, and transforms them into the required OpenCV format using cv_bridge [57]. The image is then converted to the *hue saturation value* (HSV) representation using the OpenCV function cvtColor [58]. The HSV representation can be viewed as a cylinder, as seen in fig. 10.1, where the *hue* determine which color the pixel has and is defined between 0 and 180 in OpenCV [58]. *Saturation* determine how where the color is on a scale from gray — defined from 0 to 255 [58] — to purely — for lack of a better word — the color specified by the hue. The value determines how the color is on a scale from dark to light — defined from 0 to 255 [58]. By specifying upper ($HSV_{\text{upper}} = [22, 255, 255]$) and lower ($HSV_{\text{lower}} = [18, 200, 30]$) bounds for the accepted color values, the algorithm accepts a section from the HSV color cylinder, and uses them to create a binary mask image using the OpenCV function inRange [58]. The binary image is then subjected to the morphological transformation dilation and erosion [58], in that order, to remove any holes smaller than the specified kernel — a matrix of specified size. After removing small holes, or specs, any contours was detected using the OpenCV function FINDCONTOURS [58]. Any contours of a smaller pixel area than a specified minimum of 100, or larger than a maximum of 100 000, are filtered out. The remaining contours are then enclosed with bounding boxes using the OpenCV function BOUNDINGRECT. These bounding boxes are then added to a custom message, darknet_ros::BoundingBoxes, defined by the darknet_ros repository [59]. The bounding box message is then published to the ROS network.

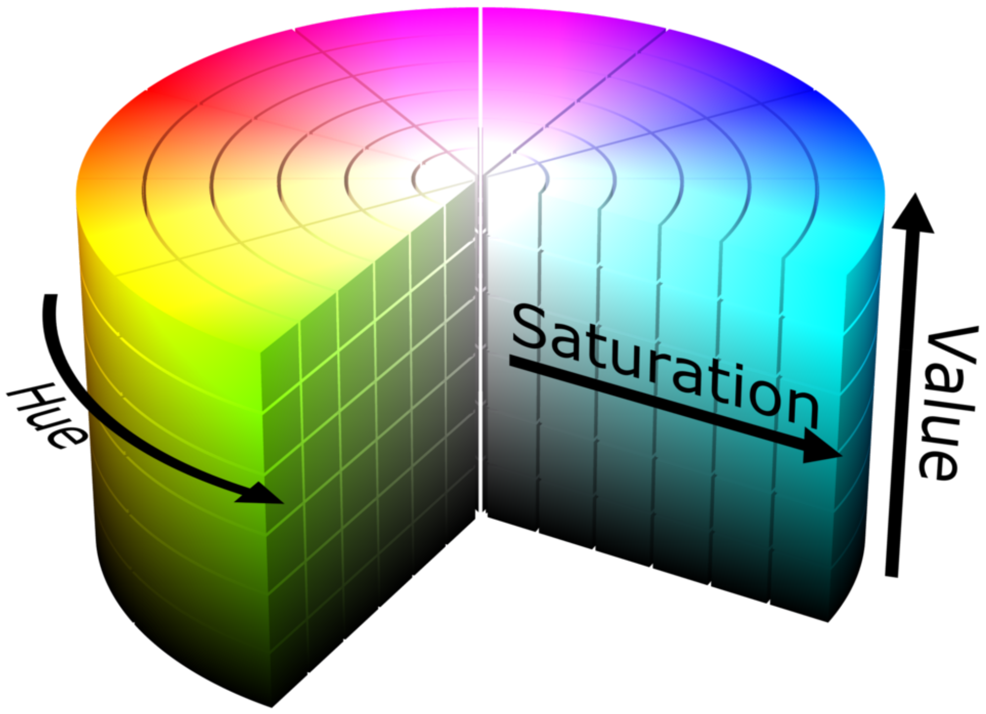


Figure 10.1: The HSV color model mapped to a cylinder, from [60]

When working in a scenario more in line with the operational conditions, such as the testing and operational conditions from section 2.2, it is not reasonable to assume that the pipe can be represented by unique colors. Although the testing scenario may utilize the bright blue bottom to simplify the problem, utilizing this known environment feature will render any algorithm based on it useless for other scenarios. For these reasons, the color segmentation based algorithm from algorithm 2 was not adjusted to solve the problem for the testing conditions. In Utbjoes project assignment [2], an algorithm to detect pipes under water was developed (see fig. 10.2). This algorithm only provided the orientation of the pipe and not localization of the pipe in the image. A weakness of this scheme was its sensitivity to parameter tuning, which proved only to be suitable for similar ranges close to the range the algorithm was tuned for [2].

Algorithm 2 Color based pipe detection

```
1:  Initialize class with subscribers publishers and color parameters
2:  hsv_lower ← lower bound of accepted colors
3:  hsv_upper ← upper bound of accepted colors
4:  callback:
5:  cv_image ← image from camerea node
6:  hsv_image ← cv_image converted to HSV color representation
7:  mask ← hsv_image converted to binary based on hsv_lower and hsv_upper
8:  mask ← mask dilated
9:  mask ← mask eroded
10: contours ← contours detected from mask
11: for contour in contours do
12:     Filter out if area is too small
13:     Define bounding box
14: Publish bounding boxes
```

Before starting any further development, it is important to be aware of the fact that this system will need to interact with other components in the ROS environment. The first step will, therefore, be to prime the existing code to be compatible with the framework and structure outlined in section 3.3. As both the classification and SLAM components need the camera frames to function, it is natural to have a separate node that feeds the frames both to the classification and the SLAM module. To this end, a node was implemented that received the camera stream before publishing the frame to the ROS network. This frame is published on the topic that is read from the classification node. For development purposes, the possibility of reading the image frames from a video file, instead of a camera, was also implemented.

The improvement on the image classification was to spatially locate — this spatial location refers to the pixel coordinates — the pipe in the image. This was done by creating a similar histogram of lines based on their angles, as was done for the sorting of the angles. Three parameters were stored for each location, the u and v pixel value, as well as the cumulative line length. Each time a new line segment was located, it was added to the average u and v pixel value at the correct angle in a manner that blended the average by weighting the effect of this line based on the line length and the cumulative line length. It was quickly identified that, for the case where the two sides constituting a pipe section lined up so that one of them fell into one angle slot and the other in another angle slot, the spatial location would be chosen only based on one of the sides. In order to get around this issue, the estimation of where the pipe segment is located based its evaluation of both of the neighboring angle slots to the angle slot chosen as the pipe section. This fixed the issue, though the algorithm saw rapid changes in where it thought the spatial location was. This was a similar issue to what was identified in Utbjoes project assignment [2], which was

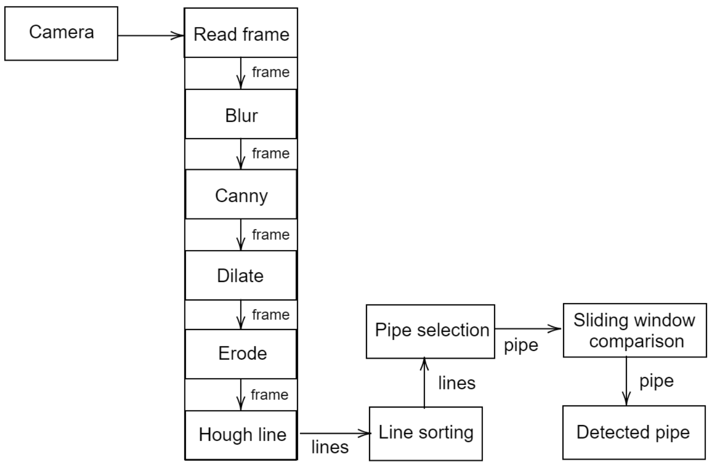


Figure 10.2: Classification system scheme presented in [2]

solved by a sliding window memory. A similar solution was decided to be implemented, however, instead of having multiple functions with a similar purpose, the sliding memory was separated into its own class. This was to ensure that the vision of creating a reusable codebase is upheld for potential future works. With the sliding window class implemented

Algorithm 3 Edge detection based algorithm

```
 1: Initialize class with subscribers publishers and edge detection parameters
 2: *callback*:
 3: $frame \leftarrow$ image from camera node
 4: $frame \leftarrow frame$ subjected to a gaussian blur
 5: $frame \leftarrow frame$ converted to binary image through canny edge detection
 6: $frame \leftarrow frame$ dilated
 7: $frame \leftarrow frame$ eroded
 8: $lines \leftarrow$ lines detected from $frame$
 9: for line in lines do
10:     Create and fill line length and line location (pixel coordinates) histograms
11:     Update $u\_avg$ and $v\_avg$ based on a weighted sum operation
12: Calculate u and v for the primary and secondary pipe segment
13: Calculate average angle
14: Pass through sliding window function
```

it was used with the primary and secondary pipe section. The algorithm worked well in lab conditions, but struggled when it came to the added complexities of real world conditions. The main impact seems to be the varying distance of the camera to the pipe, as the tunable parameters is highly dependant upon this distance. A solution may be variable tuning based on the distance to the camera view. However, at this point the development of this approach was abandoned and the focus shifted towards machine learning approaches to see if they can prove to be a suitable solutions for the operational conditions.

10.1 Deep Learning

Artificial neural networks are a part of the artificial intelligence field inspired by biological neural networks [62]. A variant of such a network, is the multilayered perceptron network [63]. Figure 10.3 shows how a one-layer network structure may look. These networks accepts an input which are then subjected to a hidden layer of perceptron nodes that provides outputs used as input by — for the example in fig. 10.3 — the output layer of two perceptrons. The arrows in fig. 10.3 represent outputs from the the previous layer that are used by the next layer. These arrows also illustrate how the information passes between the various nodes. Each node will sum up its inputs after multiplying them with a weight value defined for each input, in the sum there is also a bias term. These weight biases are updated in the training process. The sum of the weighted inputs and biases are then subjected to an activation function that determines the output of the node. In a multilayered perceptron network there would be additional hidden layers that accepts the output of the previous hidden layer as its input.

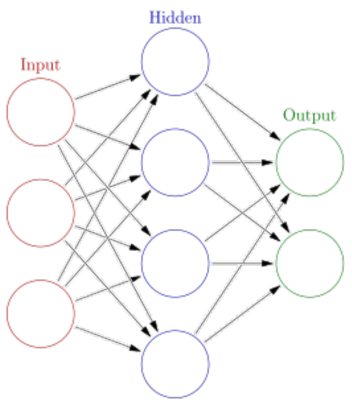


Figure 10.3: Neural network layer illustration, from [61].

Deep neural networks refers to having a large number of hidden layers, in contrast to their counterparts shallow neural networks. Figure 10.4 shows an illustration of how a fully connected neural network is structured. Fully connected networks are categorised by having their nodes connected to every node in the previous and next layer [63]. When considering deep learning, there are a large number of hidden layers which each builds on the previous. This means that the first layers may find simple features, whereas the subsequent layers can be utilized to build more complex features, that can then be used to extract meaningful information from, for example, an image. Figure 10.5 illustrates an example of how this may look for a neural network trying to detect faces. These features will be learned through the training process, in contrast to traditional computer vision approaches where these features would need to be defined through feature engineering. When considering lower level features, such as various lines, it stands to reason that these can be found without much consideration of where in the image they are located. This thesis will look at utilizing supervised learning. Supervised learning uses labeled data enable using a learning algorithm [65] that after comparing the output of the network with the labeled ground truth, the network is adjusted in accord to this learning algorithm.

Convolutional neural networks (CNNs), illustrated in fig. 10.6, are built up of several

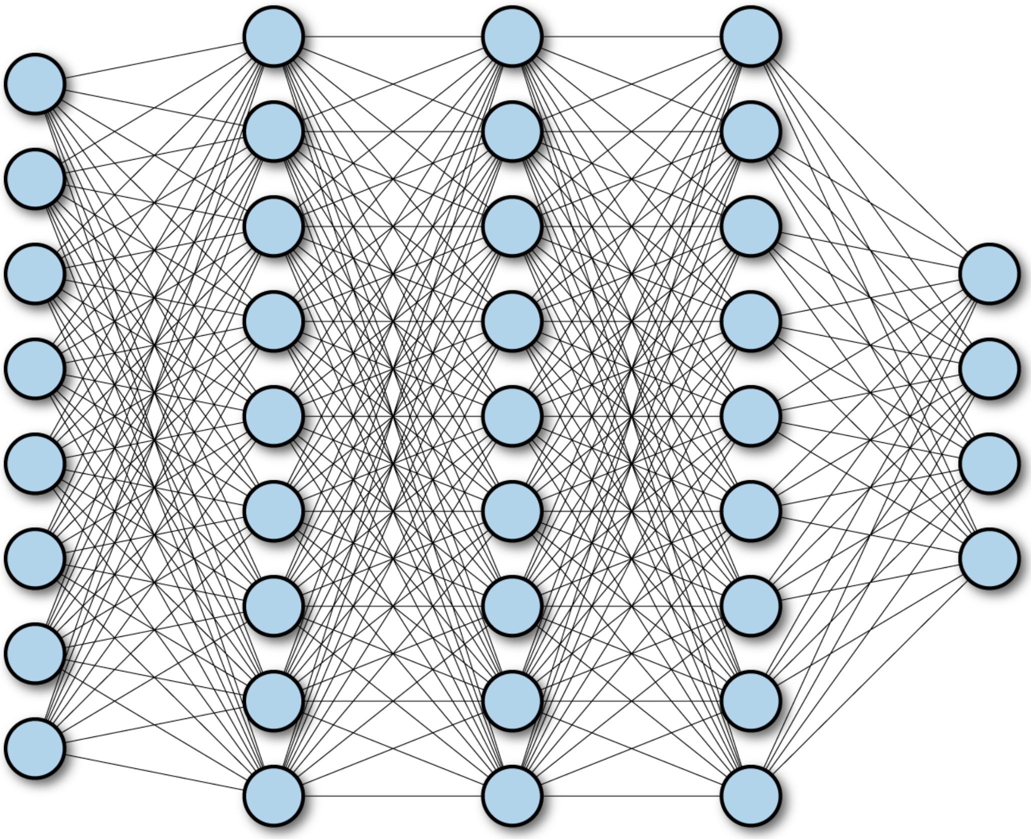


Figure 10.4: Illustration of fully connected neural network, from [64].

convolutional layers and *pooling* layers. A convolutional layer is, essentially, a set of matrices that traverses an image, multiplying the pixels they encompass at any given location to compute the pixel value of the output. A pooling layer, on the other hand, is a matrix that reduces the output dimension while preserving the information present in the input.

The convolutional layers subject the input to several filters — each filter is defined by one of the matrices constituting the convolutional layer. Each of these filters are traversed over the input and creates a layer in the output. This means that the output will have the same depth as the number of filters. The values in these filters are the weights to be learned in the convolutional layer. The same weights are used for the filter regardless of where on the image the filter is moved. This setup makes CNNs easier to train without much loss of performance as they have much fewer connections than a fully connected network [66]. With a large amount of filters, the output will often increase in dimensions when compared to the input. To keep the information transported in the network to be unnecessarily large, each of the convolutional layers are followed by a pooling layer. Pooling layers consolidates the information from the convolutional layer into a representation of smaller dimension. One type of pooling is max pooling, which looks at an area and only keeps the maximum value. By doing this, the strongest outputs in from the filters in the convolutional layer are preserved. However, when it comes to classify the high level features provided by the last pooling layer, a fully connected network is often used as this network can draw upon each of the high level feature when considering what the output should be.

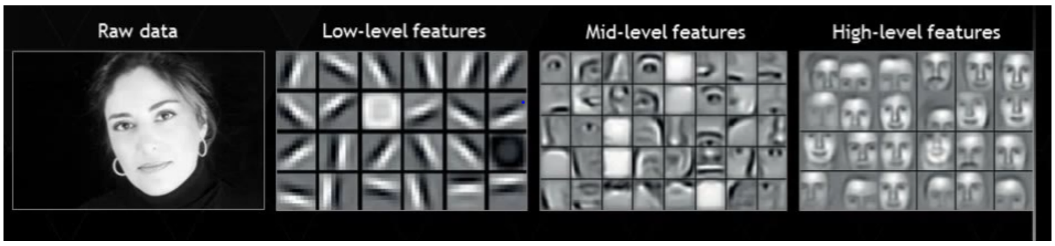


Figure 10.5: Illustration of the feature extraction process, from [67].

There are several ways of finding classes in the image using deep learning network and this thesis will consider classification, detection, localization, segmentation. Classification refers to finding whether or not a class is present in the image. However, classification does not provide any information about where in the image the class is. Figure 10.7a show how the output of a classification network may look, it would say that there is a pipe in the image, but it would not know where in the image the pipe is located. Additionally classification is limited to classify the image as a single class.

Detection refers to finding which class is in the image as well as where in the image the class is in the image, specified by a bounding box — four parameters that constitute

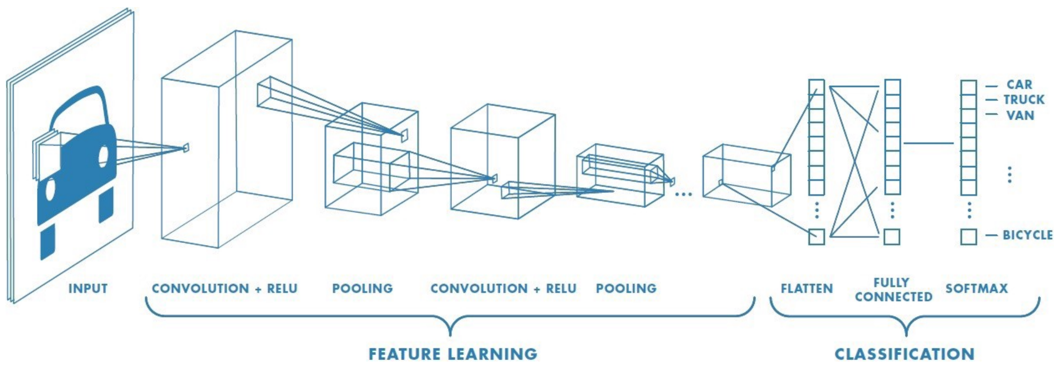


Figure 10.6: Illustration of a convolutional neural network, from [64].

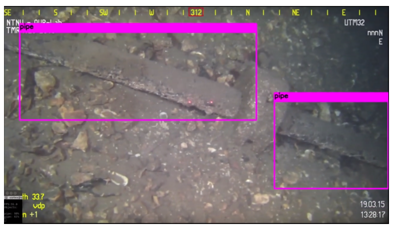
a rectangle that enclose the class in the image, see fig. 10.7b. A object detection network can handle multiple instances of various classes. The information provided by detection is provided in pixel coordinates and not euclidean coordinates.

Localization is often used to refer to augmenting a classification network to be able to locate where in the image the class is. For this thesis localization will refer to providing the information about the classes in euclidean coordinates rather than pixel coordinates. The localization of the objects will not be considered in the classification module since the SLAM map is generated from a setup that can provide euclidean information with relation to the key frames. It is therefore assumed that if the classification module can provide the pixel location, the objects can be localized in euclidean coordinates when the system is combined with the SLAM module.

Segmentation refers to labeling the image on a pixel-by-pixel basis, as can be seen in fig. 10.7c. Rather than presenting the location of a class in the image as a bounding box — as would be the case for detection — segmentation will produce an image that have each its pixel denoted as a specific class. There are two main ways of segmentation: instance segmentation and semantic segmentation. A network performing semantic segmentation will label all the pixels based on which class it is a part of. The difference between instance and semantic segmentation is that instance segmentation will also separate between different instances of a class. For instance if one had an image with multiple persons, the instance segmentation network will label them person 1, person 2, etc, while the semantic segmentation network will label them as just person.

When looking at the problem of providing context to a point cloud representation of the world, such as a SLAM map, the best suited solutions seem to be either segmentation or detection. For pipe identification, it could be valuable to represent bends and intersections as multiple pipes, and as such either detection or instance segmentation will be the best choice. Rather than trying to develop the network from scratch, this report will look at retraining an existing network to the new problem of underwater pipe detection. Both detection and instance segmentation networks are able to handle several classes and thus

(a) Classification.


(b) Detection.

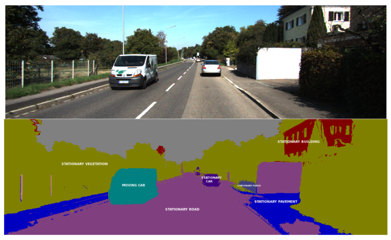

(c) Segmentation, from [68].

Figure 10.7: Illustration of the difference between classification, detection, and segmentation.

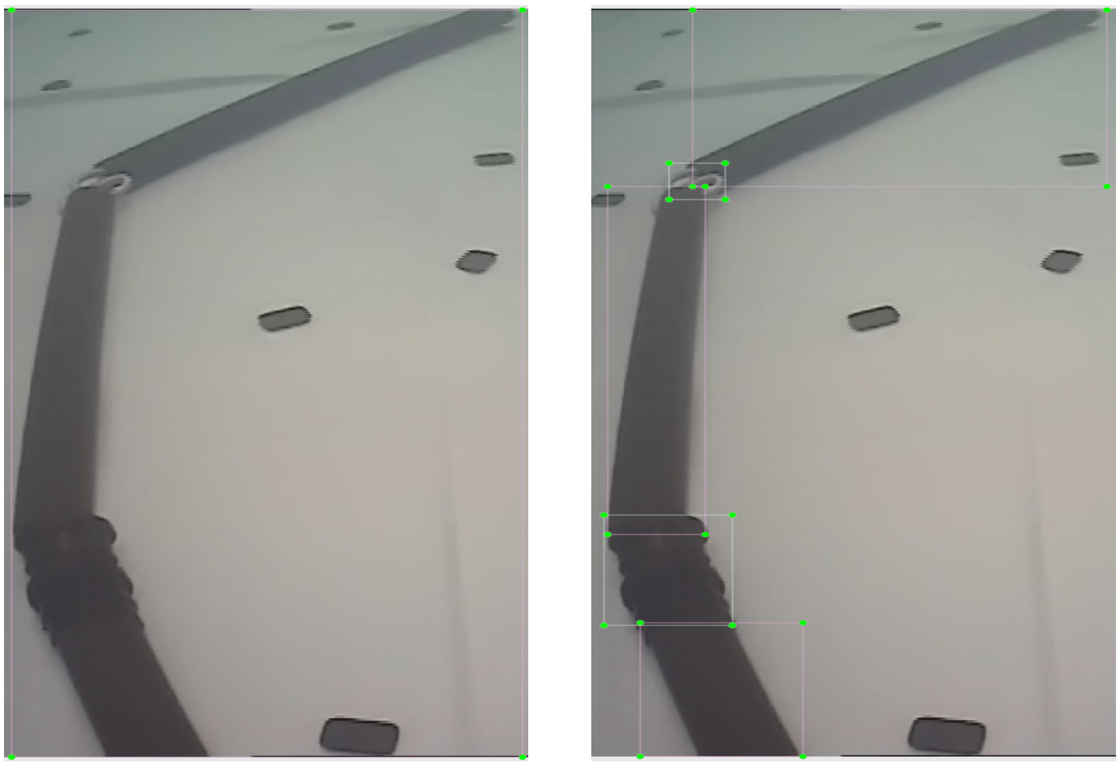
the solution should be able to handle additional classes as well.

10.1.1 Creating the Dataset

Prior to any dataset creation there need to be established a labeling scheme. Both instance segmentation and object detection networks would need their training data labeled with bounding boxes to represent the various instances of an object. Instance segmentation networks would additionally need the images labeled on a pixel by pixel basis and thus represent a significantly higher required amount of manual labor. Since both the networks need the dataset labeled with bounding boxes, the object detection networks will be considered first. This is because any effort done here may be utilized later if the detection strategy should be deemed insufficient and an instance segmentation network necessary. Another reason to start with the object detection network is that it may be possible, by making assumptions regarding the shape of the object of interest, to further narrow in where the object is within the bounding box. The primary object that will be classified is underwater pipes. A general feature of these pipes are that they the to be rather straight, when looked at from close proximity. When using the classification system to label the map points of the SLAM system, an approach to narrow in on which map points may fit along a straight line could be a *Random Sample Consensus* (RANSAC) algorithm.

The exception to the heuristic of straight line pipes over short distances is naturally where a pipe junction creates a sharp bend in the pipe, as shown in fig. 10.8a. In order to preserve the heuristic of straight pipes, the pipe is represented as several instances of straight pipe segments bound together by pipe junctions, as shown in fig. 10.8b. When comparing the bounding box in fig. 10.8a to the boxes in fig. 10.8b, the latter would result in a significantly reduced amount of falsely categorized map points as a larger percentage of the various bounding boxes are filled with their respective class. Additionally, a class that represents the docking station entrance is added, as this is present in the test basin.

With the labelling scheme established, the next objective is to find a way to label the data in a accurate and efficient manner. The choice fell on the open source library LabelImg by Tzutalin [69]. LabelImg creates xml-files that contain the information of what classes



(a) Labeling entire pipe as one pipe. (b) Labeling pipe as pipe segments and junctions

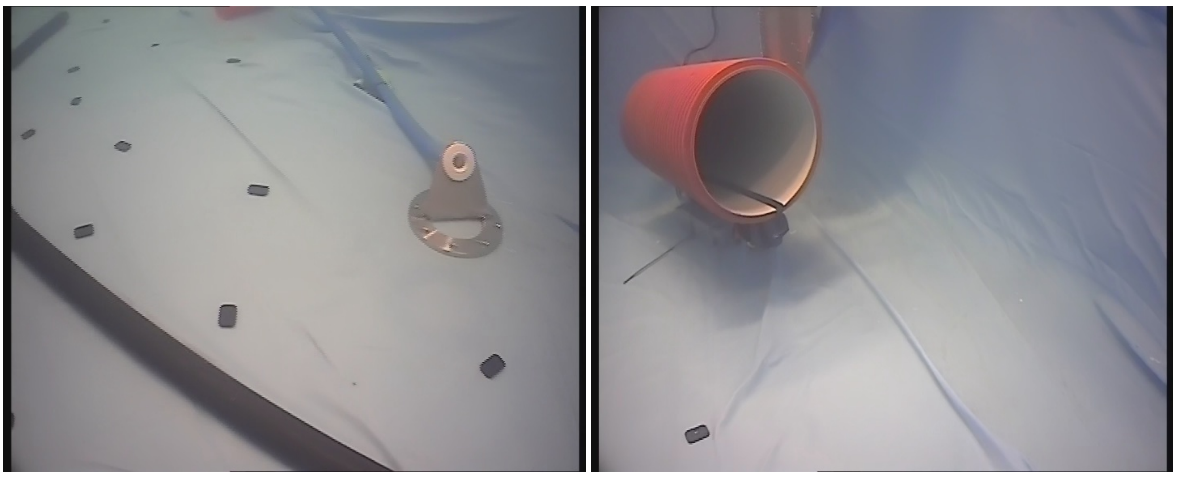
Figure 10.8: Two alternative strategies for labeling the pipe.

are present in the image, and where they are located in the image. The information in these xml-files can then be adapted to suit the format required by the object detection network in question.

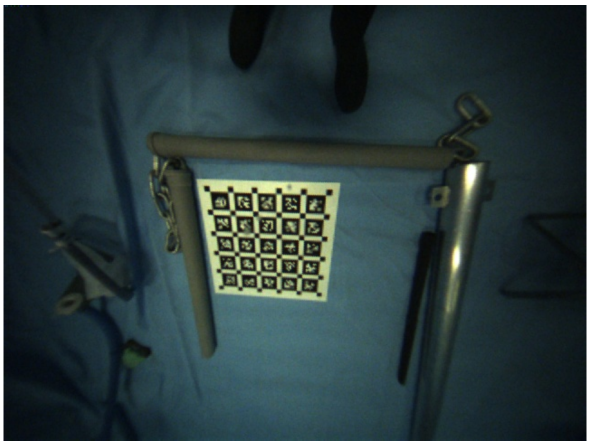
Both the testing and operating setting are relevant platforms for the system to operate using the hardware platform. These two cases are considered as the conditions that the classification module needs to be able to solve. There are considerable differences between the two cases, most notably that the ground is blue in the test basin, while it is textured by rock, sand, etc. in the operational case. Since the cases deviate significantly and one would not expect to encounter conditions close to the test basin in an actual operation and vice versa, these cases will be handled by creating a separate dataset for each of the settings. When creating a dataset it is important that the network is subjected to as many of the conditions it will meet as possible in order for it to handle these cases later on. For instance, if the network has only been subjected to vertical pipes one could not fault it for being unable to recognize horizontal pipes later on. The dataset should be balanced and contain pipes of different orientations and sizes. A way to improve the dataset is to use image augmentations. Image augmentation could be used to get varying orientations from the same image [70]. It will be important to preserve the labels in these image augmentations. However, image augmentation will not be explored further in this thesis. This thesis would like to see if a strategy for gradually improving the dataset can be established. In order to isolate the impact of the gradual improvements to the dataset, it will not be tried in parallel with techniques such as image augmentation. To achieve this gradual improvement, the performance of the network will be observed and problematic conditions identified through qualitative inspection. The dataset will the be augmented with images similar to the problematic ones and the network retrained to observe the effect. This is not an argument against image augmentation as this should be explored as well, but this thesis will focus on how added images can impact the performance and propose a strategy to make the network increasingly robust.

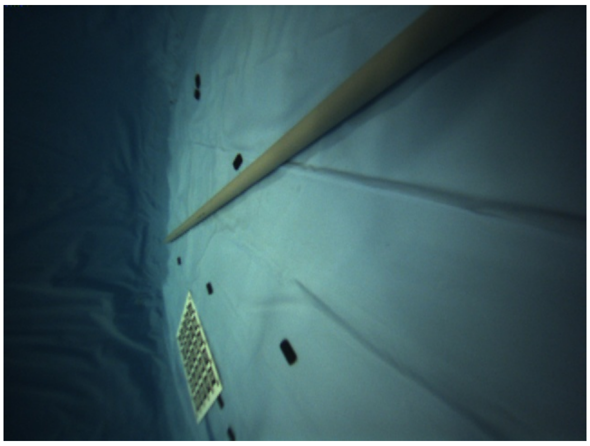
Test conditions

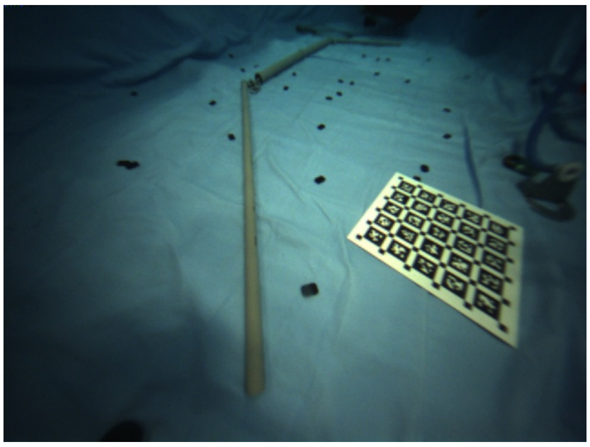
The hardware platform was constructed as a part of this project and thus not available at the start. For this reason the first data gathered from the test basing was recorded using a GoPro camera — the same recording that was used as ground truth in [2]. This video was split into separate images which was then labeled and used for training the system. Once the hardware platform was ready it presented a significant change in the conditions of the data, see fig. 10.9. To have the network more attuned to the conditions from the hardware platform there where gathered more data using the hardware platform. In order to get a representation of different views of the pipe, the hardware platform was rotated various ways while moving the camera around the test basin. When it comes to the scale of the pipes there is not a great potential for varying depths in the test basin and the dataset will therefore focus mainly on varying orientations of the pipe. Figure 10.10 shows an example of neighboring images where the hardware platform being rotated to have various angles of the pipe in the image.



(a) Recorded with the GoPro camera. (b) Recorded with the GoPro camera.


(c) Recorded with the hardware platform.


(d) Recorded with the hardware platform.


(e) Recorded with the hardware platform.

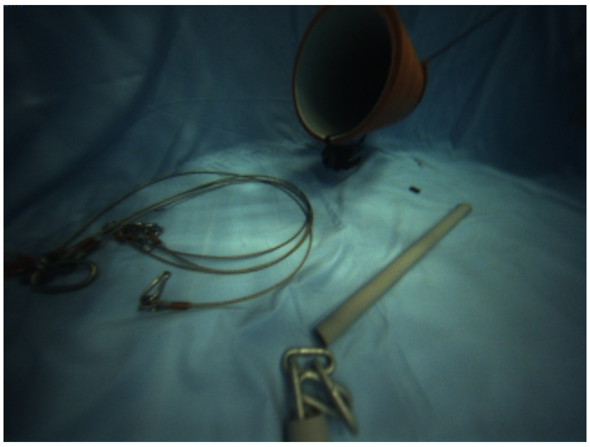

(f) Recorded with the hardware platform.

Figure 10.9: Images from the three recordings used for training data.

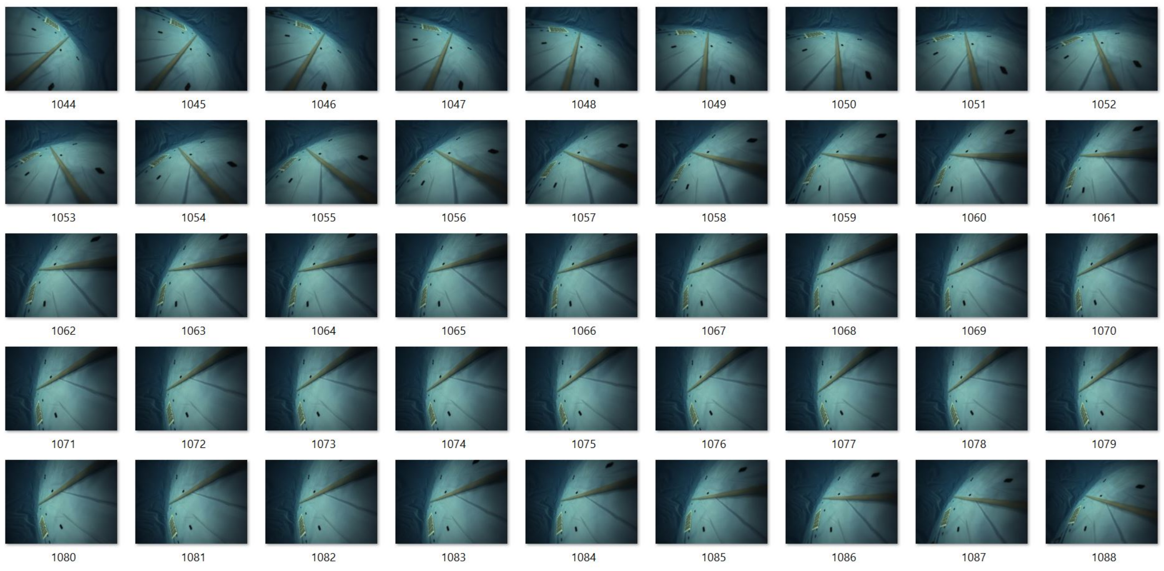


Figure 10.10: Rotating hardware platform to get a variety of pipe angles in the training data.

Operational conditions

The basis for the operational conditions are — as stated in section 2.2 — a video from an actual underwater pipe. This video will act as both the basis for the training set as well as gathering information about the performance of the network. In [2] the training data was gathered from the entirety of the video by saving images at a set interval from the entire video. For this thesis another approach is used. A section of the larger video where the pipe is present, represented at various of orientations and distances is selected and split into individual images. These images are then labeled manually. This will preserve parts of the video with vastly different conditions from the selected video to see how the network performs on previously unseen conditions. The dataset will then be expanded to see if the performance may be improved significantly by adding a relatively small number of images from these conditions. Figure 10.11 show some examples of the various pipes present in the selected video at different orientations and distances.

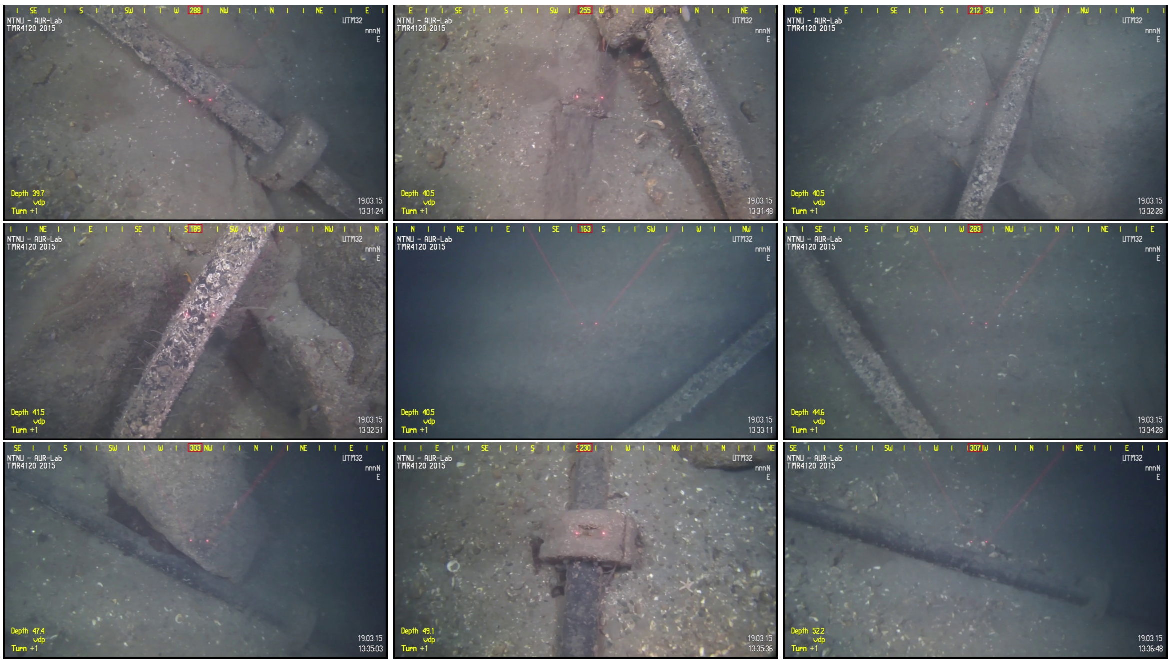


Figure 10.11: Images from the selected part of the video.

10.1.2 Evaluation of Results

When comparing the lab condition to the operational conditions it is significantly easier to imagine a network working in the lab conditions. Laboratory conditions are much easier to test in and draw qualitative and quantitative results from. However, the operational condition is arguably the most interesting case, as this will be closest to the conditions the system will face in the ocean and hardest case. To gauge the performance of the network on underwater conditions, it will be evaluated based on the operational conditions.

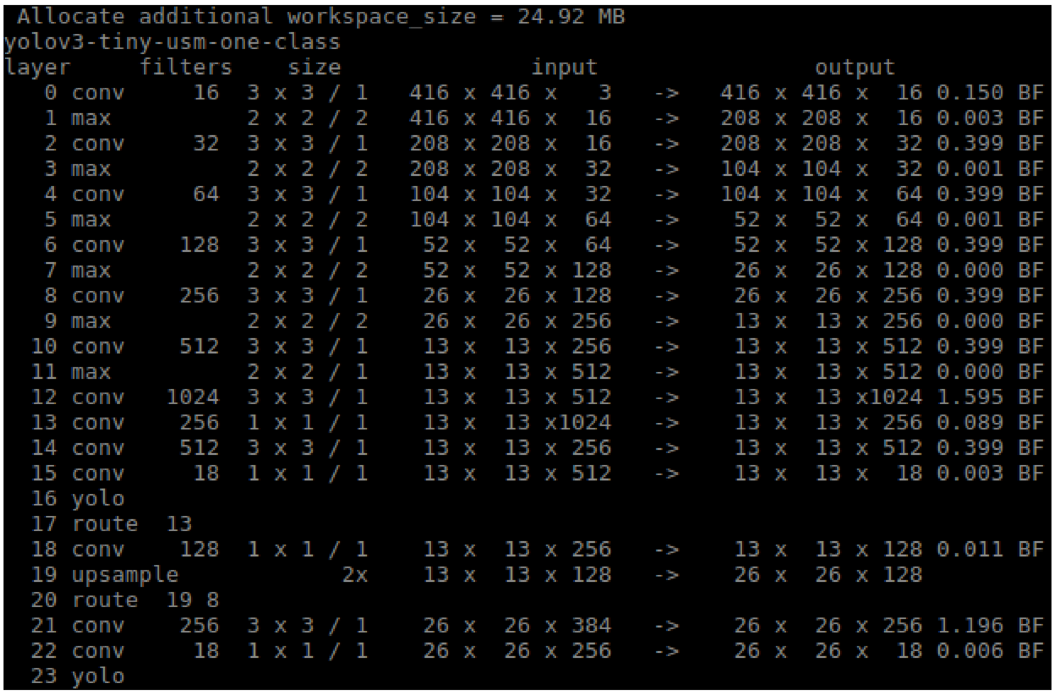
In order to get a reliable metric for the network performance, the operational case dataset will be consisting of more than 10 000 images, all labeled manually. These images will then be sectioned in to 10 subsets of around 1000 images each. These subsets will then act together to cross validate the performance and reducing the impact from outlier cases. The process for this will be to train the network using nine of the subsets and using remaining one for validation. This will be repeated so that the network has 10 different training foundations that are different from the remaining validation images. This cross validation will provide a basis for evaluating the performance. The result from the training session will be a graph illustrating the *mean Average Precision* (mAP) — mAP is the mean value of the average precision (the precision being the overlapping area of the predicted bounding box and the ground truth bounding box divided by the ground truth bounding box area) for each of the classes — and the loss. The mAP value indicate how well the network perform on the validation set, while the loss is calculated from from the training set. In short, the loss should be as low as possible, while the mAP should be as high as possible.

10.1.3 Network Selection

There are many existing network that are available to be evaluated. However, rather than comparing a wide variety of networks to find which is the most suited for the task, this thesis will focus on finding a network that perform sufficiently. Once a network has been deemed sufficient the work of interfacing the modules will be prioritized over comparing different networks.

One example of a object detection network, is the region-based CNN (R-CNN) which divides the detection process into three main stages: region proposal, feature extraction, and classification [71]. In the original R-CNN network the feature extraction where done using a CNN network. The next improvement done upon R-CNN came with Fast R-CNN [72], where the classification is achieved using a CNN. Another improvement was then added with Faster R-CNN by adding a CNN to do the region proposal [73]. An extension of this network is Mask R-CNN which combines a mask network on top of the bounding boxes provided by faster R-CNN to produce the instance segmentation.

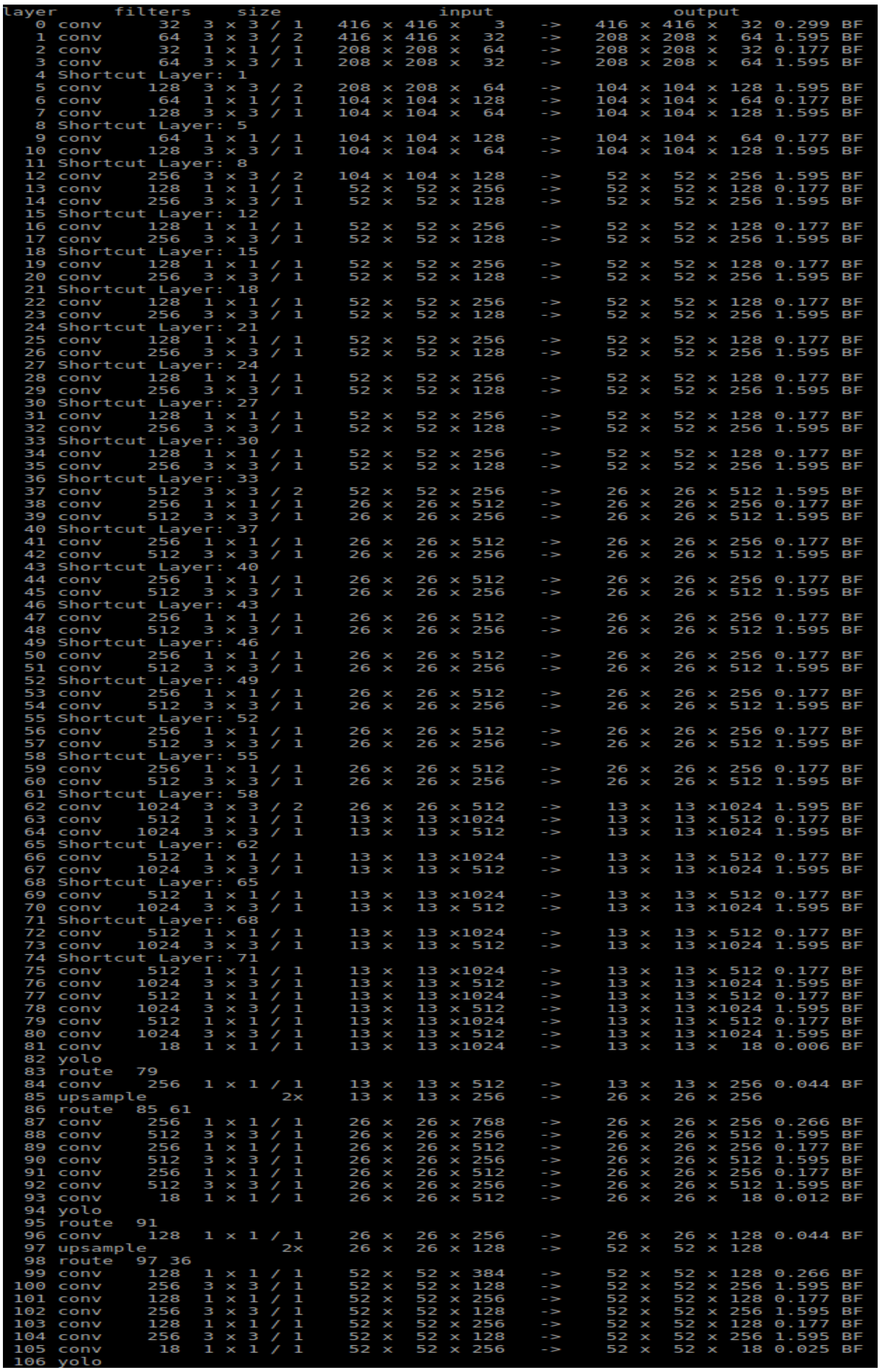
Another type of object detection network is the YOLO (You Only Look Once) network [74] which is very fast by designing its structure for fast performance. It achieves this speed by doing all the steps in one CNN. It was later released an improved version YOLOv2 [75], which improves the accuracy while preserving real time performance. Recently it was further improved with YOLOv3 [76].



```
 Allocate additional workspace_size = 24.92 MB
yolov3-tiny-usm-one-class
layer     filters    size              input                output
   0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16 0.150 BF
   1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16 0.003 BF
   2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32 0.399 BF
   3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32 0.001 BF
   4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64 0.399 BF
   5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64 0.001 BF
   6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128 0.399 BF
   7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128 0.000 BF
   8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256 0.399 BF
   9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256 0.000 BF
  10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512 0.399 BF
  11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512 0.000 BF
  12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
  13 conv    256  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 256 0.089 BF
  14 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512 0.399 BF
  15 conv     18  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x  18 0.003 BF
  16 yolo
  17 route  13
  18 conv    128  1 x 1 / 1    13 x  13 x 256   ->    13 x  13 x 128 0.011 BF
  19 upsample            2x    13 x  13 x 128   ->    26 x  26 x 128
  20 route  19 8
  21 conv    256  3 x 3 / 1    26 x  26 x 384   ->    26 x  26 x 256 1.196 BF
  22 conv     18  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x  18 0.006 BF
  23 yolo
```

Figure 10.12: Network structure of the YOLOv3-tiny network.

Both the R-CNN and YOLO methods seem to be strong candidates. The Mask R-CNN network also performs instance segmentation, which may be helpful when looking to provide the best labeling of the point cloud map. YOLO on the other hand has the advantage of being a fast network and could potentially operate in real time. YOLO has another advantage in being built on darknet [77], which can easily be integrated into ROS, where the rest of the system is implemented, by using the open source library darknet_ros [59]. Due to this ease of integration and since the dataset will first be labeled towards a object detection network, YOLOv3 and its smaller cousin YOLOv3-tiny is selected as the networks to try to apply to the problem of detecting underwater pipes. YOLOv3-tiny is a smaller version of YOLOv3 and is thus faster, but not as complex. The layers of YOLOv3-tiny can be seen in fig. 10.12, whilt the layers of YOLOv3 can be seen in fig. 10.13. If YOLOv3-tiny proves to be sufficiently accurate when facing the complexities of the operational case, it will be used as it will have the most rapid performance. Should YOLOv3-tiny be shown to lack the complexity to handle operational conditions, it will be necessary to use YOLOv3.



```
layer     filters    size              input                output
   0 conv     32  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  32 0.299 BF
   1 conv     64  3 x 3 / 2   416 x 416 x  32   ->   208 x 208 x  64 1.595 BF
   2 conv     32  1 x 1 / 1   208 x 208 x  64   ->   208 x 208 x  32 0.177 BF
   3 conv     64  3 x 3 / 1   208 x 208 x  32   ->   208 x 208 x  64 1.595 BF
   4 Shortcut Layer: 1
   5 conv    128  3 x 3 / 2   208 x 208 x  64   ->   104 x 104 x 128 1.595 BF
   6 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64 0.177 BF
   7 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128 1.595 BF
   8 Shortcut Layer: 5
   9 conv     64  1 x 1 / 1   104 x 104 x 128   ->   104 x 104 x  64 0.177 BF
  10 conv    128  3 x 3 / 1   104 x 104 x  64   ->   104 x 104 x 128 1.595 BF
  11 Shortcut Layer: 8
  12 conv    256  3 x 3 / 2   104 x 104 x 128   ->    52 x  52 x 256 1.595 BF
  13 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
  14 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
  15 Shortcut Layer: 12
  16 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
  17 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
  18 Shortcut Layer: 15
  19 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
  20 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
  21 Shortcut Layer: 18
  22 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
  23 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
  24 Shortcut Layer: 21
  25 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
  26 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
  27 Shortcut Layer: 24
  28 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
  29 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
  30 Shortcut Layer: 27
  31 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
  32 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
  33 Shortcut Layer: 30
  34 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
  35 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
  36 Shortcut Layer: 33
  37 conv    512  3 x 3 / 2    52 x  52 x 256   ->    26 x  26 x 512 1.595 BF
  38 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  39 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  40 Shortcut Layer: 37
  41 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  42 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  43 Shortcut Layer: 40
  44 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  45 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  46 Shortcut Layer: 43
  47 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  48 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  49 Shortcut Layer: 46
  50 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  51 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  52 Shortcut Layer: 49
  53 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  54 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  55 Shortcut Layer: 52
  56 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  57 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  58 Shortcut Layer: 55
  59 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  60 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  61 Shortcut Layer: 58
  62 conv   1024  3 x 3 / 2    26 x  26 x 512   ->    13 x  13 x1024 1.595 BF
  63 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
  64 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
  65 Shortcut Layer: 62
  66 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
  67 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
  68 Shortcut Layer: 65
  69 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
  70 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
  71 Shortcut Layer: 68
  72 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
  73 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
  74 Shortcut Layer: 71
  75 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
  76 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
  77 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
  78 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
  79 conv    512  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 512 0.177 BF
  80 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
  81 conv     18  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x  18 0.006 BF
  82 yolo
  83 route  79
  84 conv    256  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x 256 0.044 BF
  85 upsample            2x    13 x  13 x 256   ->    26 x  26 x 256
  86 route  85 61
  87 conv    256  1 x 1 / 1    26 x  26 x 768   ->    26 x  26 x 256 0.266 BF
  88 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  89 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  90 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  91 conv    256  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x 256 0.177 BF
  92 conv    512  3 x 3 / 1    26 x  26 x 256   ->    26 x  26 x 512 1.595 BF
  93 conv     18  1 x 1 / 1    26 x  26 x 512   ->    26 x  26 x  18 0.012 BF
  94 yolo
  95 route  91
  96 conv    128  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x 128 0.044 BF
  97 upsample            2x    26 x  26 x 128   ->    52 x  52 x 128
  98 route  97 36
  99 conv    128  1 x 1 / 1    52 x  52 x 384   ->    52 x  52 x 128 0.266 BF
 100 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
 101 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
 102 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
 103 conv    128  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x 128 0.177 BF
 104 conv    256  3 x 3 / 1    52 x  52 x 128   ->    52 x  52 x 256 1.595 BF
 105 conv     18  1 x 1 / 1    52 x  52 x 256   ->    52 x  52 x  18 0.025 BF
 106 yolo
```

Figure 10.13: Network structure of the YOLOv3 network.

10.1.4 Training the Network

In order to use the datasets created using LabelImg [69], they need to be adapted to match the form required by the YOLO network. This was done by adapting the labelling script from darknet [77]. With the annotations adapted to suit the YOLO format, it is necessary to split the dataset into validation and training sets. In order to bolster the conclusions that can be drawn from the training, the dataset have been split into subsets which will be used to cross validate the results. The dataset is shuffled randomly before being assigned evenly into ten subsets. In the training process, one of these subsets will be held back for validation, while the remaining subsets will be used for training. This will be the basis for the training and will be repeated for each of the subsets. The main indicator for how well the network solves the training will be mAP value.

When training a network, it is important to make sure that the network generalize the result and do not overfit — the process where the network starts to memorize the training data and becomes poorer at predicting for previously unseen data [78] — on the training data. More complex networks are more susceptible to overfitting when compared to simpler, less flexible networks [78]. In order to detect the point where the network starts to memorize the data the first training sequence will be run for 20 000 iterations for both YOLOv3-tiny and YOLOv3. The results from this training will be evaluated to see what will be a suitable number of iterations for the training process. Once this number has been detected the remaining sets will be trained with this amount of iterations. Figure 10.14 shows the results from training the YOLOv3-tiny network on the dataset representing the test basin. The network was able to operate well in the test basin. However, as the operational setting have the most interesting conditions to evaluate the network performance, the test basin set will not be discussed further. The reasoning for training this set was to have a network that can work in combination with the SLAM module. The result from this combination can be seen in section 16.2.

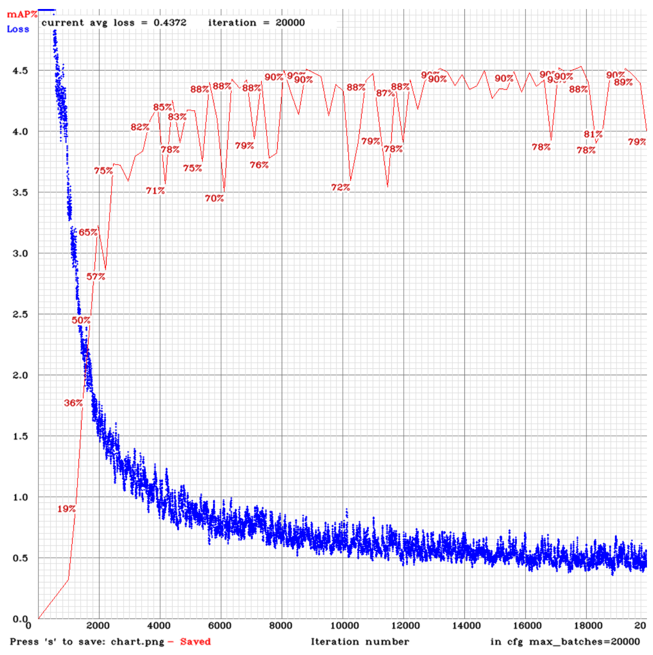


Figure 10.14: Training results YOLOv3-tiny on the test conditions.

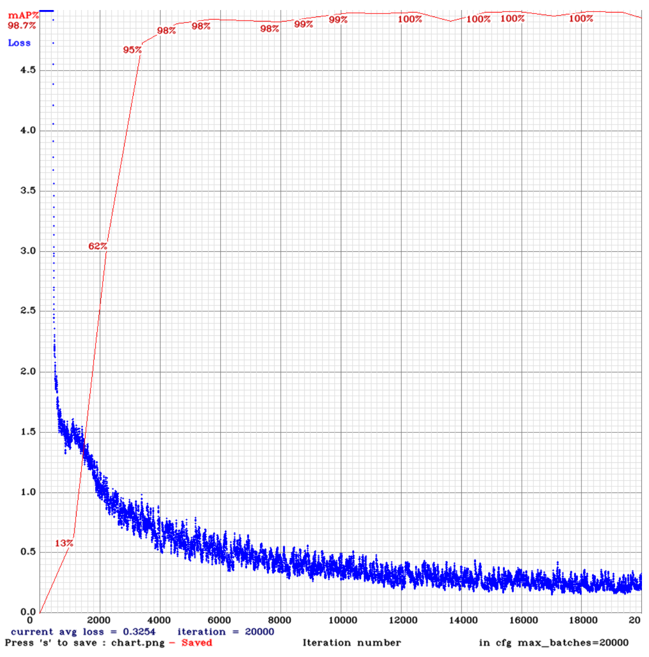


Figure 10.15: Training results YOLOv3-tiny using subset 0 as validation data for 20 000 iterations.

Figure 10.15 shows the results from the training sequence of YOLOv3-tiny when it has been trained for 20 000 iterations. Below 4000 iterations there is a significant increase in accuracy, which then starts to oscillate up to 100% mAP. When the network starts to improve on the edge cases and starts to slightly increase this indicates that it is overfitting. Indeed, when testing some examples on the weights at 4000 and 10 000, there was no large improvement between them. This indicates that the point where the training have been completed, and the network starts overfitting, lies around 4000 iterations. This is the basis for the subsequent training sequences to be run for 6000 iterations. It will then pass 2000 iterations more than what is indicated is necessary for it to start overfitting, which will make it so that it becomes apparent in the results that the optimal point is reached. At the same time this number of iterations will save a significant amount of time in the training process that would likely just have been occupied by the network overfitting.

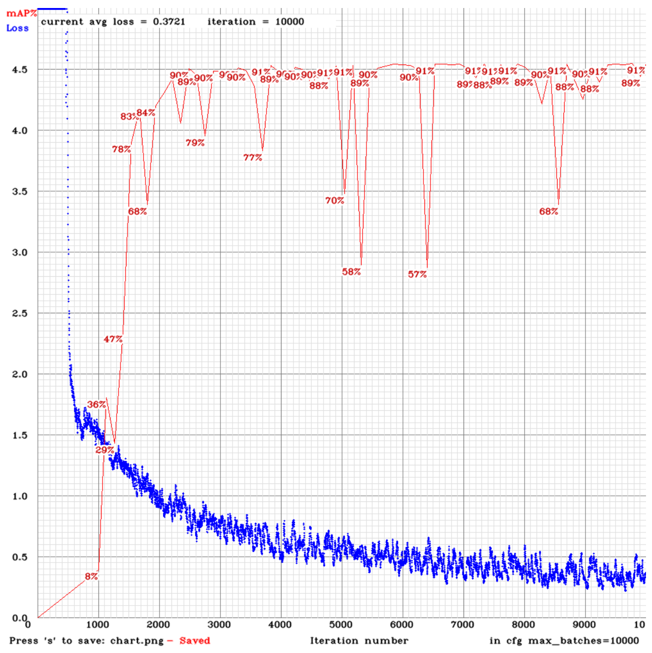
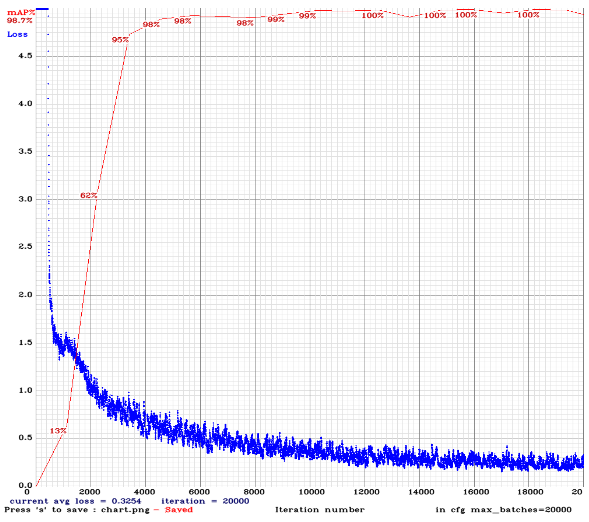
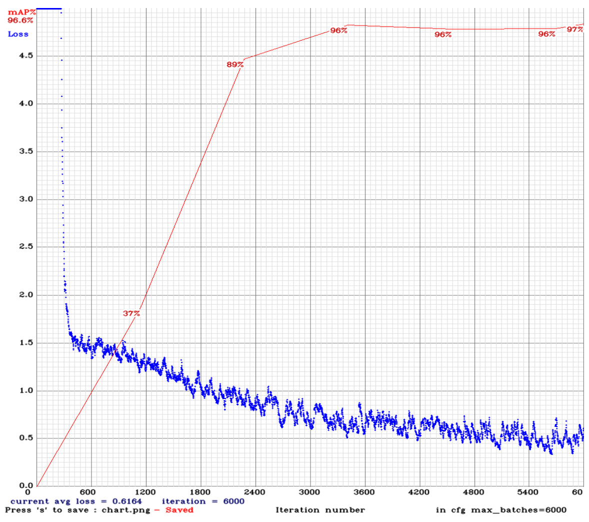


Figure 10.16: Training YOLOv3-tiny on set 0 as training data for 10 000 iterations.

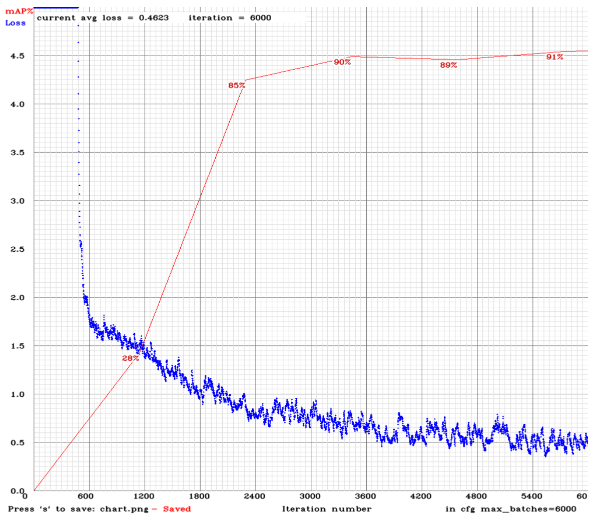
After completing the training for all the subsets as validation, it seems that the YOLOv3-tiny network are sufficiently complex to learn the complexities of the real condition, see fig. 10.17. From the results it can be seen that the mAP does not drop when the network is overfitting, as one would expect. This seems to be because the conditions in the validation data is very close to the images in the training data. In order to combat this issue it was tried to have the validation set is much larger than the training set by using one of the subsets as training data while the remaining subsets constitutes the validation data. This leaves around 1000 images for training and around 9000 images for validation. The training was then tried for 10 000 iterations to see how the performance develop. From fig. 10.16 it can be seen that the YOLOv3-tiny network is able to learn to represent the dataset on the limited training data of one subset. Figure 10.16 shows that the network is able to learn well from the training process given only one of the subsets as training data. Once the network reach around 5000 there are some sharp decreases in the mAP performance indicating that the network has had sufficient time to train within 6000 iterations for this case as well. Subsequent training sessions with similar amount of training data will therefore be conducted over 6000 iterations as previously.



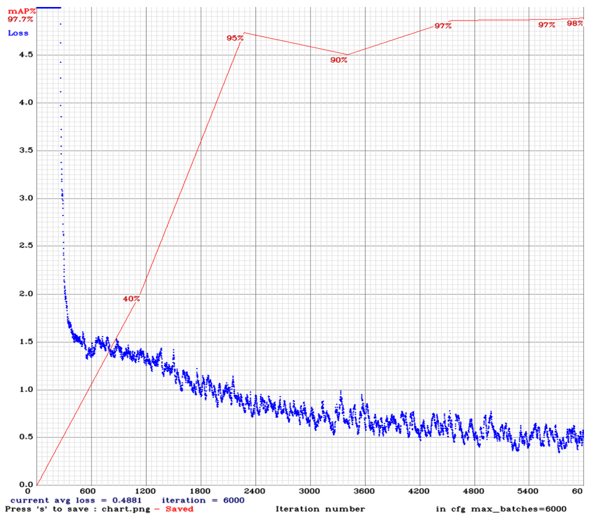
(a) Using subset 0 as validation data.



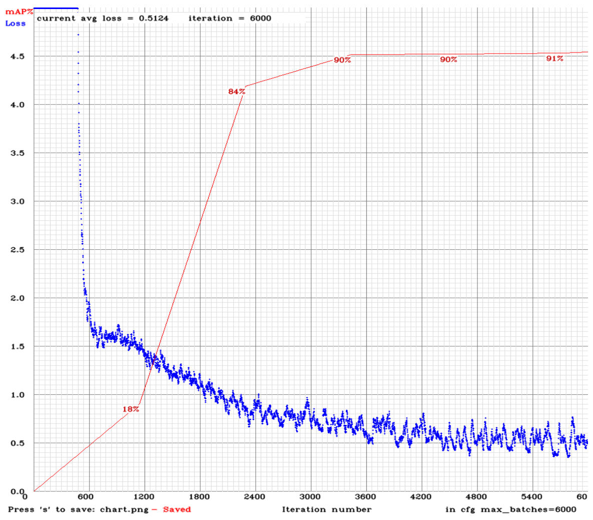
(b) Using subset 1 as validation data.



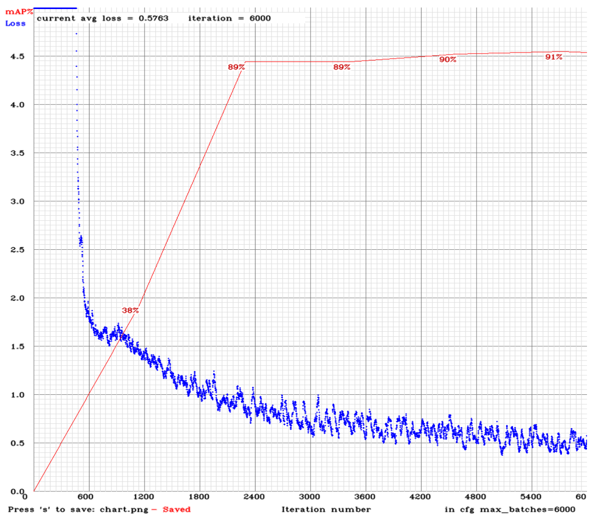
(c) Using subset 2 as validation data.



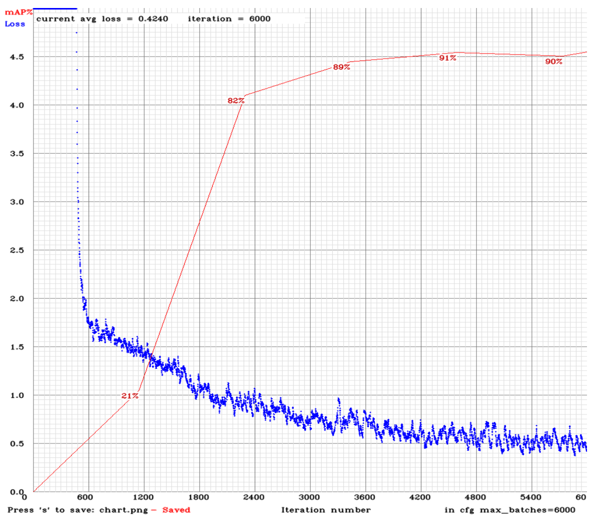
(d) Using subset 3 as validation data.



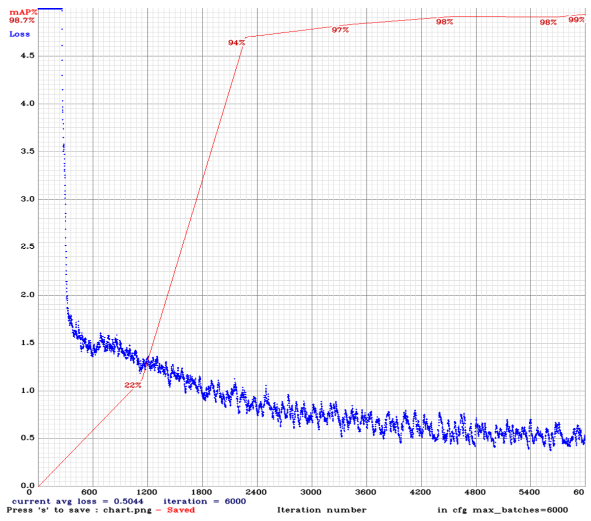
(e) Using subset 4 as validation data.



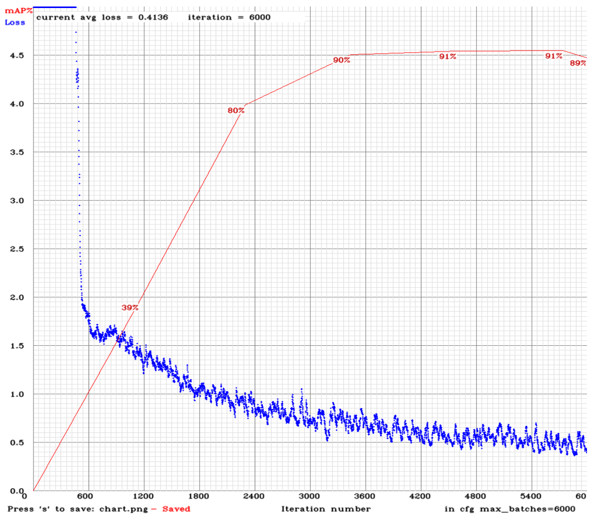
(f) Using subset 5 as validation data.



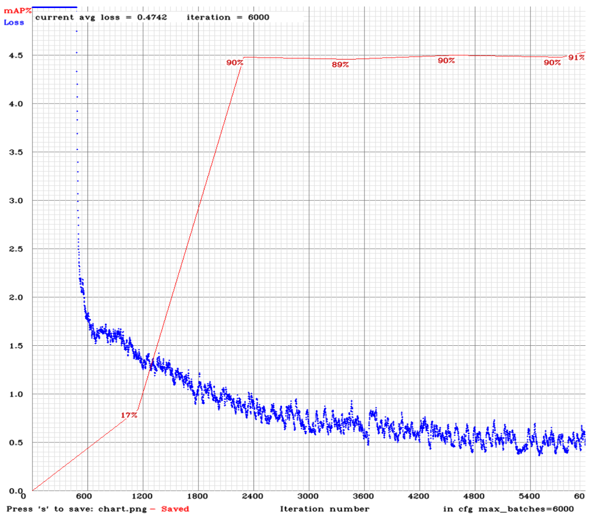
(g) Using subset 6 as validation data.



(h) Using subset 7 as validation data.



(i) Using subset 8 as validation data.

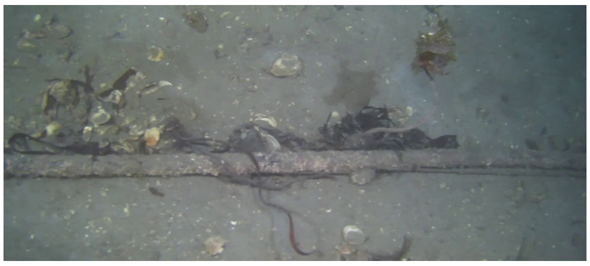


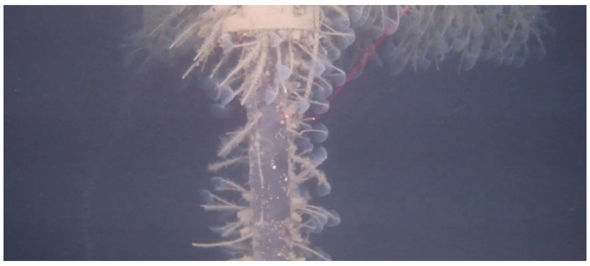
(j) Using subset 9 as validation data.

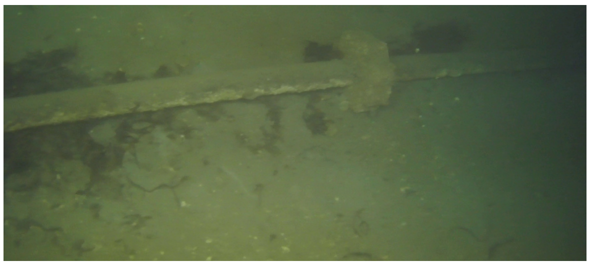
Figure 10.17: Cross validation training of YOLOv3-tiny.

From fig. 10.17 it can be seen that the network YOLOv3-tiny consistently reach a mAP% score of 90% or more. This result indicate that the network is sufficiently complex to handle the conditions present in the real conditions of the training set. Another insight gained from the training shown in fig. 10.17 came from subjecting the trained network to the entirety of the video where the clip constituting the dataset is collected from. Some of the challenges was when the lighting condition changed to include a green shear as seen in fig. 10.18c, when the pipe was much thinner as seen in fig. 10.18a, when the pipe was severely biofouled as seen in fig. 10.18b or when the pipe was predominantly covered by sand rather than biofouling, as seen in fig. 10.18d. This revealed that the dataset is not sufficiently representative of the conditions that are present in the video as the network struggled on all parts that strayed far from the point where the dataset was collected from.

In order to combat this the dataset was enhanced by adding images that represented the entirety of the video into the training data. These images was selected from the dataset created in [2] which contained data spanning the entirety of the conditions present in the video. From this dataset 668 images containing pipes was selected and labeled, additionally it was added 250 images not containing pipes. These images was then added to the training data. The goal behind this enhancement is to better represent the wide array of conditions met on an underwater pipe inspection. As labeling a dataset in the same scale as the already created dataset would be very labour intensive and time consuming, it was decided that the effect of this enhancement would be evaluated qualitatively by observing it in action on the varying conditions. This will of course not give the same certainty on the actual performance metric, but it will give an indication on whether or not it moved the needle in a positive manner.


(a) Thin pipe.


(b) Severe biofouling.


(c) Green shear in the light condition.

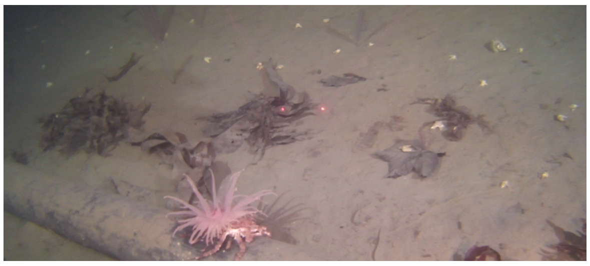

(d) Sand covered pipe.

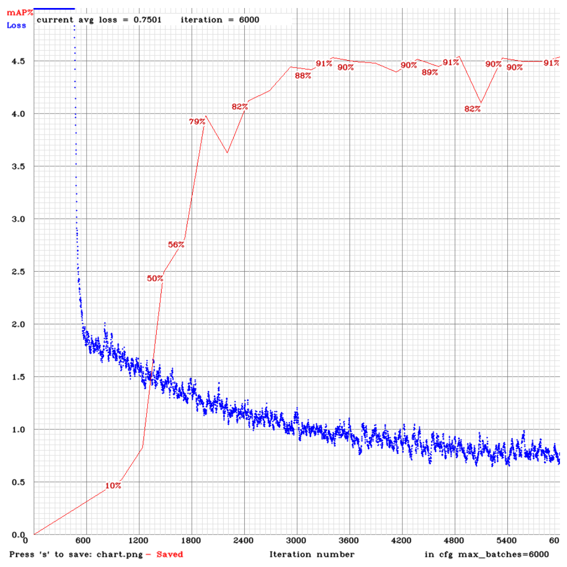
Figure 10.18: Examples of challenging pipe conditions.

Figure 10.19a show the result of the training process on the expanded dataset. These results indicate that the network are able to learn the features from these new added conditions. In order to get an impression on the potential improvement, the network was tried on the complete video. The impression from this test was a significant improvement over the more homogeneous dataset. The network still struggled more when the pipe is further away and where there was a green shear in the light conditions. Areas of the video that the pipe was not identified at all previously, this network was able to find the pipes at a much higher accuracy. The test indicate a need for more training data for some of the conditions. Though before expanding the dataset further it will be trained with YOLOv3 to see if the full version is able to generalize with this amount of data.

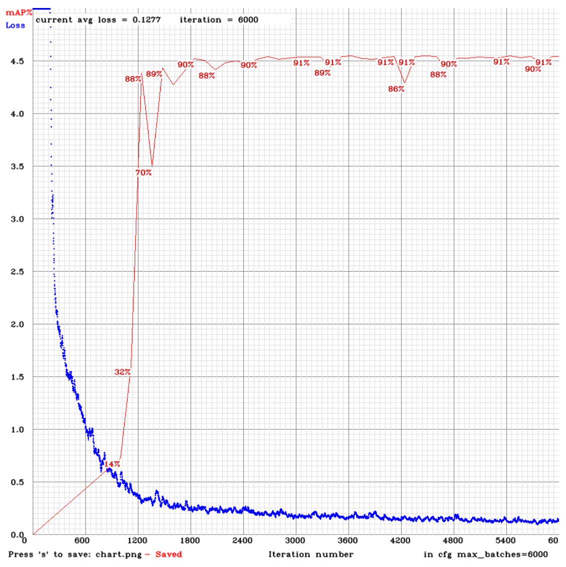
Figure 10.19b show the result when trying to train the full YOLOv3 network on the expanded dataset. This shows that the network is able to learn the data in the training set well. This is to be expected as YOLOv3-tiny was able to learn these conditions as well. The most interesting results came from observing these weights on the complete video. This test showed better performance than YOLOv3-tiny, though the network struggled with the same conditions, green shear and pipe far away. This indicate that it is worth to augment the dataset by adding more of these sort of images to see if this improves the performance notably. It was therefore added 63 more images containing pipes in the difficult conditions. After this second expansion the dataset was used to train the YOLOv3 network to see if the performance is increased.

Figure 10.19c show the result from training the YOLOv3 network on the dataset with two expansion. Due to an error the training was stopped short, but the 4000 iteration weights was used to try and gain insights that might improve the next training. These weights was then tried on the full video. In this test it was identified a significant improvement over the case shown in fig. 10.19b, especially on the problematic images identified from the weights of fig. 10.19b. These weights did struggle on some images still, mainly when the pipe was very far away. Though the most interesting result was probably that it experienced false positives when the camera was very close to the sea bed. In order to improve this it was added a significant number of images without pipe so that the network can get a better understanding of how the sea bed looks without pipes and thus reduce these false positives. Upon further inspection it was found that the images not containing pipes from the first expansion was labeled incorrectly. Thus the network had some images which convinced it that the empty sea floor was part of a pipe. This is fixed for the next training session.

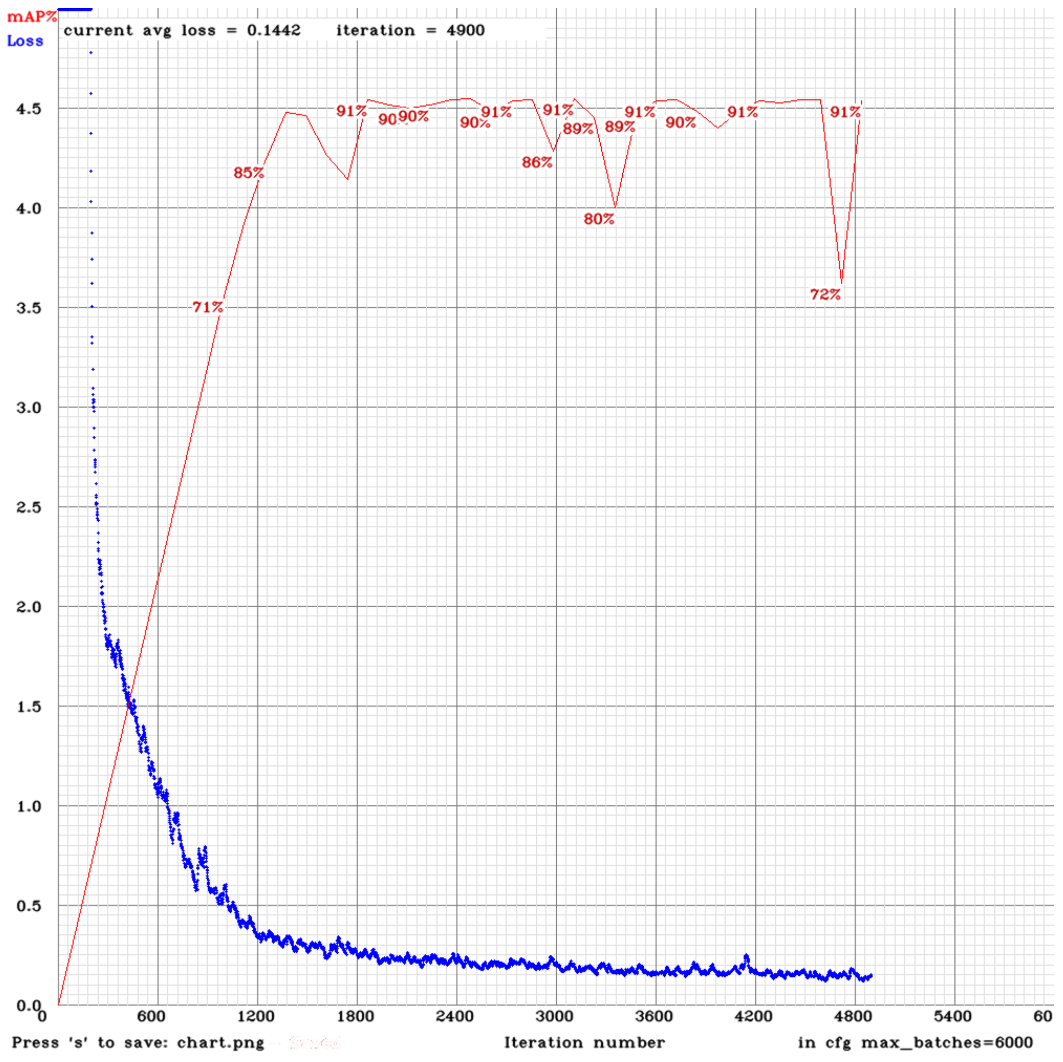
With the dataset expanded and error fixed this was decided to be the final version of the dataset. In order to evaluate the performance both the YOLOv3 network and YOLOv3-tiny network was trained for 20000 iterations. The results from this training can be found in figs. 10.21 and 10.23. From the training results to seemed that both the networks trained successfully. After the training process the best performing weights are automatically chosen and saved with the suffix _final.weights. These weights were then tested on the entire video to see how the two different networks performed.



(a) Training YOLOv3-tiny on set 1 with expansion as training for 6000 iterations.



(b) Training YOLOv3 on set 1 with expansion as training for 6000 iterations.



(c) Training YOLOv3 on set 1 with both expansions as training for 6000 iterations.

Figure 10.19: Training YOLOv3-tiny and YOLOv3 on expanded dataset.

(a) (b) (c)

Figure 10.20: Video of YOLOv3-tiny trained on the final dataset.

The QR-code in fig. 10.20a directs to a video of the final weights of the training sequence in fig. 10.21. From this video one can see that the network consistently finds the pipe. Some of the behaviour may be described as trigger happy, as the network often allocate several bounding boxes to the same pipe. Several hits for the same pipe is not necessarily an issue as the bounding boxes will later be used to classify what map points are of a certain class. In this structure it should not be an issue if the pipe get more than one bounding box classifying the same point as a pipe multiple times. There are still some false positives, though this is to a much lesser extent than previously. In order to reduce the impact of false positives this report proposes to use a sliding window technique used in the traditional computer vision approach. This should be evaluated over the neighboring frames to increase the robustness. It is also of note that these additional hits are presented with a significantly lower certainty. This could be used to employ a threshold to determine which threshold should be accepted.

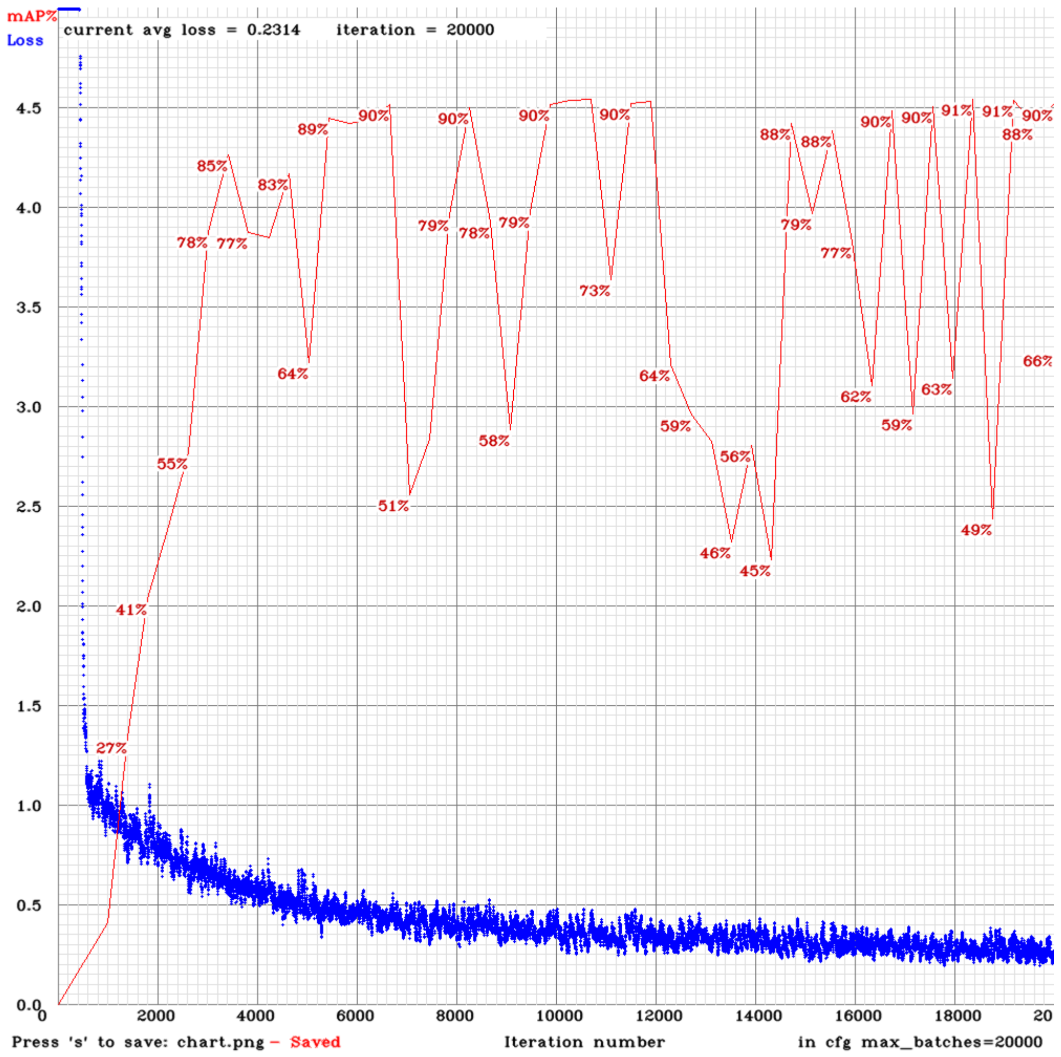


Figure 10.21: Training YOLOv3-tiny for 20000 iterations.

The QR-code in fig. 10.22 directs to a video of the final weights of the training sequence in fig. 10.23. The YOLOv3 network worked well on the entire video, though it exhibited fewer extra positives when compared to the YOLOv3-tiny network from fig. 10.21. On the flip side it experienced more false positives when compared with the network from fig. 10.21. Since the performance of the full sized YOLOv3 network could not be considered superior to its smaller counterpart YOLOv3-tiny. For this reason YOLOv3-tiny, with its increased frame rate, seem to be the best choice. YOLOv3-tiny is therefore proposed as the network of choice for the system.

Figure 10.22: Video of YOLOv3 trained on the final dataset.

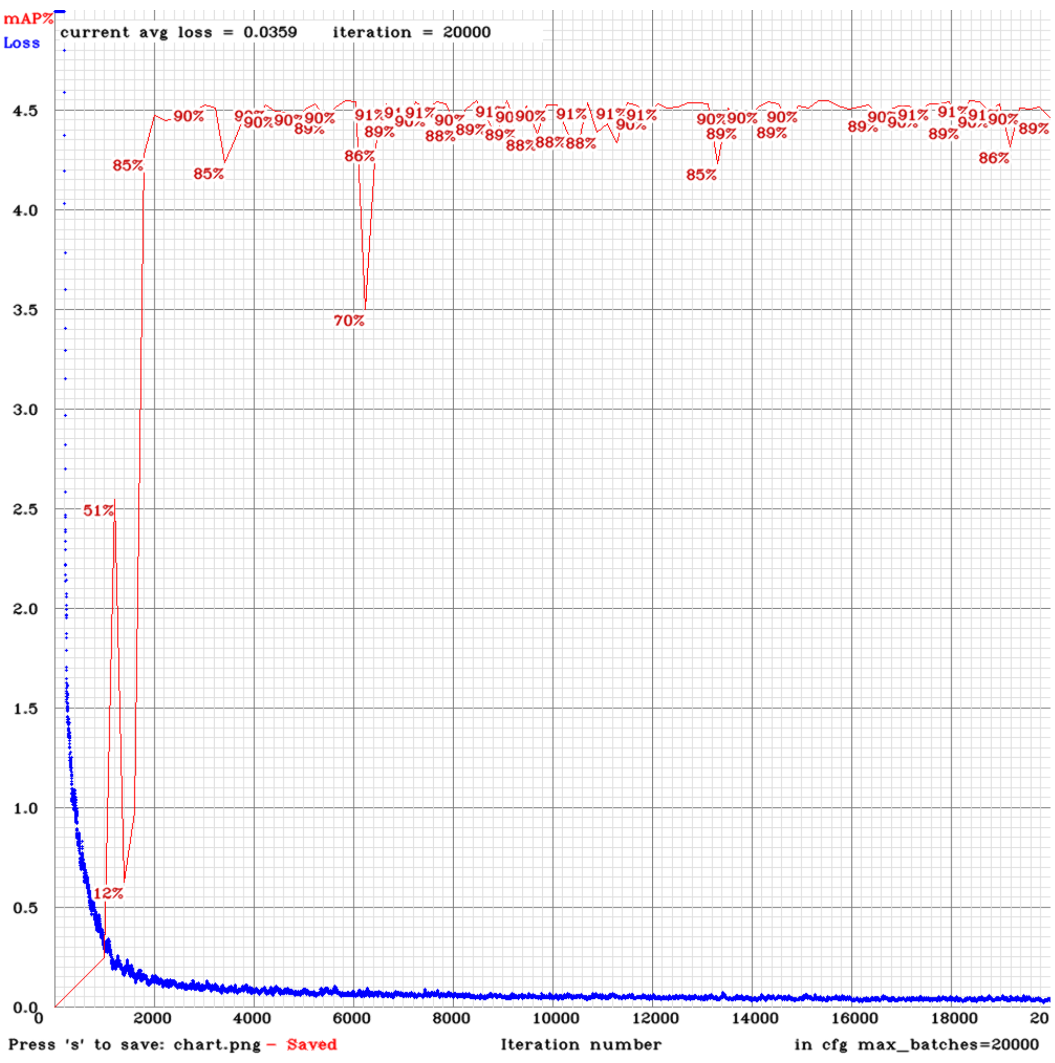


Figure 10.23: Training YOLOv3 for 20000 iterations.

Figure 10.24: Video of YOLOv3 trained on the final dataset for 100 000 iterations.

Lastly, the YOLOv3-tiny was trained for 100000 iterations to see whether or not the extreme increase in training time could give some insights into if the network overfitted or if there where any improvements by training for a very long time. The result of the training can be seen in fig. 10.25 and can be viewed with its final weights by following the QR-code in fig. 10.24. The first thing of note is that the network was much more conservative with the detections and there was a significant decrease in extra detections and false positives. Another interesting insight was that network struggles when the pipe was at very close proximity to the camera. All in all this seemed to be the best performing network. However, it is not certain if this is due to the extra training time or a spot of luck in the training sequence. Figure 10.26 show a selection of various weights tested on a shortened version of the operational video. The weights are grouped in sections of four to compare how the network performed given various number of iterations. Figure 10.26a show the weights at 2000, 4000 6000 and 8000 iterations. The weights from 2000 iterations have very few detections and often misses the pipe. At 4000 iterations, the network starts to improve significantly in the detections, while still missing a fair amount. The weights at 6000 iterations have much fewer misses, but the network is very trigger happy. At 8000 iterations the network is less trigger happy than the weights from 6000, while still having the increased amount of correct detections. Figure 10.26b show the weights at 5000, 10 000, 15 000 and 20 000 iterations. There are some noticeable improvement from 5000 to 10 000, though the weights 15 000 and 20 000 does not seem to offer a significant improvement. This seem to be an indication that the network have reached its most general performance around 5000 to 10 000 iterations. Figure 10.26c show the weights at 25 000, 50 000, 75 000 and 100 000 iterations. This was to see if there could be an improvement by letting the network train for a significantly increased time. From these weights it seemed that the weights from 25 000 iterations performed better than the weights that had trained for a larger number of iterations, indicating that a significant increase in training time does not seem to be beneficial.

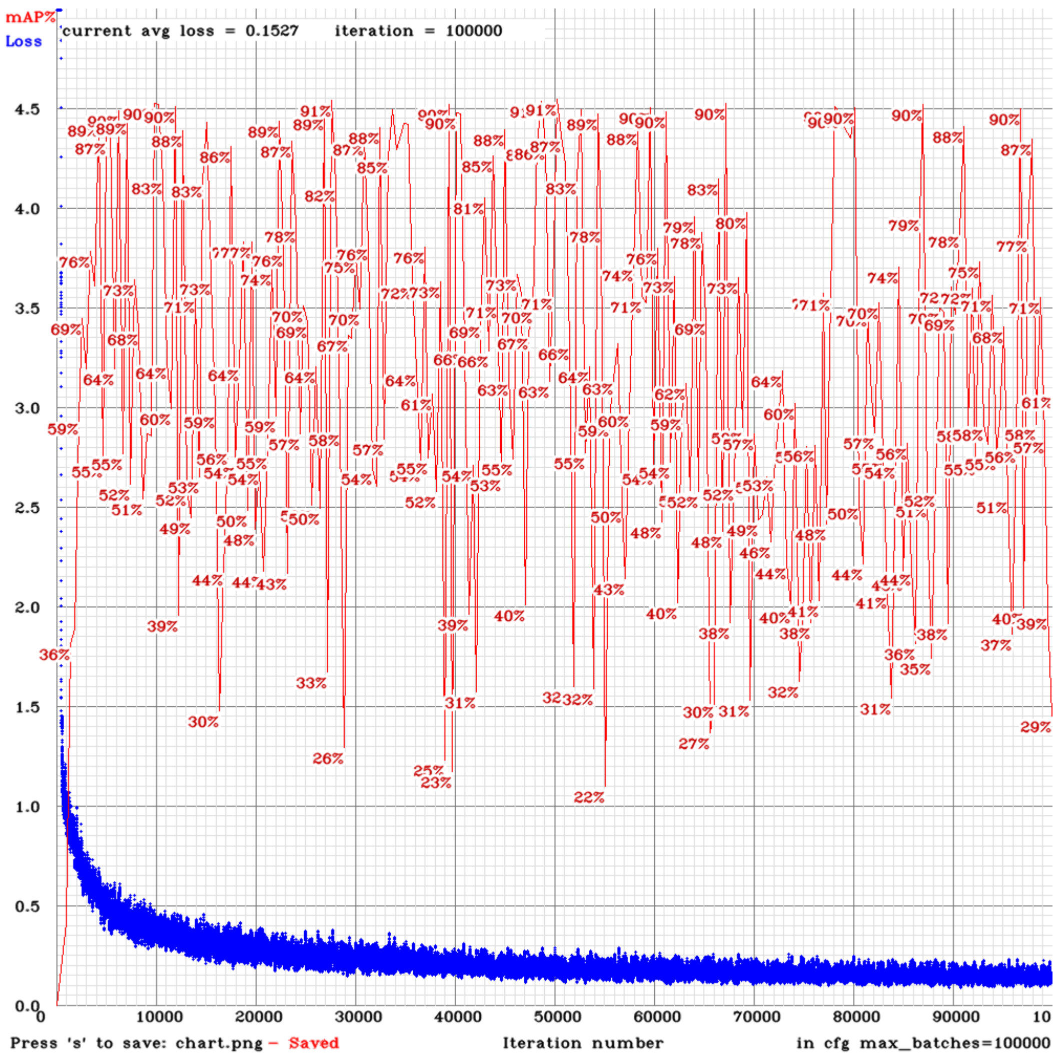


Figure 10.25: Training YOLOv3-tiny for 100000 iterations.

(a) Video of the YOLOv3-tiny trained for 2000, 4000, 6000 and 8000 iterations.

(b) Video of the YOLOv3-tiny trained for 5000, 10 000, 15 000 and 20 000 iterations.

(c) Video of the YOLOv3-tiny trained for 25 000, 50 000, 75 000 and 100 000 iterations.

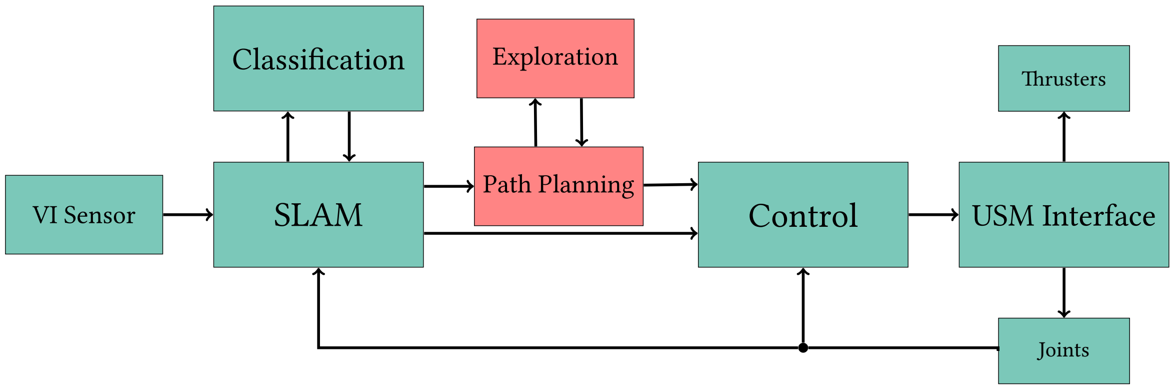
Figure 10.26: Videos of YOLOv3-tiny trained on the final dataset using weights from various iterations.

10.2 Summary

In this chapter two approaches for solving underwater pipe detections have been tried, one using traditional computer vision techniques and the other machine learning based. After working to make the traditional computer vision algorithm work adequately, this work was abandoned to make time to see if the machine learning approach could solve the complex situation of operational conditions. In section 10.1 the goal was to find a neural network that could solve the complex operational setting. The chosen network was YOLOv3-tiny and through training and observing the network, it was deemed a sufficiently good solution. A strategy for gradually improving the dataset was presented and tried to be effective. Before any autonomous inspection system can be implemented it would seem reasonable that the system may be tested in parallel with an manual operation to see how it performs. In this process the dataset can be expanded until it performs as needed.

Part V

Planning



11 | Background Theory

Solving the path planning problem requires the definition of an underlying mathematical representation. This section is heavily based on the works of Latombe, LaValle and Tsourdos et. al, see [79]–[82] for a more in-depth coverage of the topic at hand.

Similarly to many dynamical systems, the planning problem can be represented using a *state space* representation. Each state $x \in X$ contains information about the robot's pose and is part of a finite set of states, X. The state space then consists of all necessary information needed to sufficiently solve the planning problem at hand, disregarding all non-relevant information. The robot, or *agent*, transitions from one state to another specified by a *state transition function*, $f(\cdot)$. This function describes the transition from state x to state x' when the agent executes the action u, calculated by the planner [80], [83]:

$$x' = f(x, u) \ . \tag{11.1}$$

Furthermore, let $U(x)$ denote all allowed actions for each state x. The set of all possible actions over all states can then be defined as in [80]:

$$\mathcal{U} = \bigcup_{x \in X} U(x) \ .$$

The path planning problem can be generalised to calculating a path from an initial pose $\eta_{b/I,0}$ to one or more goal poses, $\eta_{b/I,g} \in X_g \subset X$, where X_g represents the set of goal states.

The ultimate goal of a path planning algorithm can then be described as calculating a sequence of actions, u, that transforms the initial state x_i to a goal state $x_g \in X_g$. In the simplified case where the environment is assumed discrete and two-dimensional, the state space X can be represented as a graph, with the vertices representing the states. In this case, there exists a directed edge between x and x' iff. there exists a $u \in U(x)$ such that $x' = f(x, u)$. In the case of a continuous state space, the assumption of a finite state space is no longer valid as X is no longer countably infinite [80] [1], and more care is needed when modelling the state space.

To be able to relate the robot's actions and state to its environment, the *workspace* $\mathcal{W} = \mathbb{R}^n$ it resides in, must be defined[2]. Let $\mathcal{O} \subseteq \mathcal{W}$ be the *obstacle region* and $\mathcal{A} \subseteq \mathcal{W}$

[1]Countably infinite set: Any infinite set which can be put in a one-to-one correspondence with $\mathbb{N}$ (countable). Cardinality: $\aleph_0$.

[2]For most cases: $n = 2$ (2D) or $n = 3$ (3D), but this may increase in the presence of joints, for example.

be the space in $\mathcal{W}$ occupied by the robot. The obstacle region is the part of the world that is occupied by static bodies, such as an exterior wall, a fence or an underwater rock formation, and will in most cases be fixed in $\mathcal{W}$. The robot, $\mathcal{A}$, will be modelled as a moving rigid-body.

In rigid body kinematics, the act of coordinate transformation is essential. Two concepts used to execute such transformations are coordinate *rotation* and *translation.* In the 3D case, a robot $\mathcal{A}$ can be rotated around any of the three defined orthogonal axes. Such a rotation about a given fixed axis is called a *simple rotation.* Furthermore, it can be shown that more complex rotations can be described as a set of *composite rotations* around the respective principal axes [17], such as shown in section 2.1.1.

The rotation matrices for each cardinal axis can be described as a three-dimensional matrix performing a counterclockwise rotation around each axis. By combining the rotation and translation of a rigid body, the subspace occupied by the transformed robot can be defined as follows [79], [81]:

Definition 11.1. Let $\mathbf{T} : \mathcal{A} \longrightarrow \mathcal{W}$ be a rigid-body transformation, as defined in section 2.1.1, that maps every point of $\mathcal{A}$ into $\mathcal{W}$, preserving the distance between any pair of points p, $q \in \mathcal{A}$ as well as the cross-product of any two vectors. The subspace of $\mathcal{W}$ occupied by the transformed robot can the be defined by

$$\mathbf{T}(\mathcal{A}) = \{\mathbf{T}(a) \in \mathcal{W} \mid a \in \mathcal{A}\} \ . \tag{11.2}$$

Based on these definitions, the set of all allowable transformations can be defined. This set is called the *configuration space*, C, also known as the C-space [79]. For a robot moving in a three-dimensional environment, its configurations, η_{Ib},[3] can be represented as $\eta = (x, y, z, w, \epsilon_1, \epsilon_2, \epsilon_3)$, where $(w, \epsilon_1, \epsilon_2, \epsilon_3)$ describes a quaternion $\mathbf{q} = w+\epsilon_1\mathbf{i}+\epsilon_2\mathbf{j}+\epsilon_3\mathbf{k}$ [15]. The Euler convention defined in section 2.1.3 is just as applicable here, but the quaternion representation is used for sake of argument below.

Quaternions provide a double covering of $\mathbf{SO}(3)$, which means that two quaternions will correspond to the exact same rotation [84]. More explicitly, this means that $\mathbf{q} \sim -\mathbf{q}$ with respect to spatial rotation. To define the encompassed space, consider a homeomorphism $f : S^3 \mapsto \mathbf{SO}(3)$ [85] with a kernel $f = \{\pm 1\}$ equal to the centre of S^3. Thus, the co-sets of $\ker(f)$ in S^3 defines antipodal pairs, each pair further defining a line between them in $\mathbb{R}^4$-space. From definition 11.2 it can be seen that this is similar to the definition of the *real projective space*, $\mathbb{RP}^3$, in terms of the antipodal map $\pi : S^3 \mapsto \mathbb{RP}^3$, indicating $\mathbf{SO}(3) = \mathbb{RP}^3$ [86].

Definition 11.2. Let $\mathbb{S}^n \subseteq \mathbb{R}^{n+1}$ be an n-dimensional unit sphere defined by $\mathbb{S}^n := \left\{ x \in \mathbb{R}^{n+1} \mid |x - y| = 1 \right\}$. The *real projective space* of dimension n is then defined as

[3]In path planning literature, q is most often used to represent the robot configuration.

the quotient space [87], [88]:

$$\mathbb{RP}^n := \mathbb{S}^n/\{\pm\} \ .$$

$\{\pm\}$ here identifies the *antipodal points* $\in \mathbb{S}^n$.

Using the definition of the $\mathbb{RP}^n$ space, the configuration space for all 3D transformations can be defined.

Definition 11.3. ***Configuration Space (C-space)***
Let $\mathcal{A}$ be a rigid body capable of rotations and translations in 3D space. The set of all 3D rotations is then defined by the antipodal subset $\mathbb{RP}^3 \subset \mathbb{R}^4$ (definition 11.2), with the set of translations equal to $\mathbb{R}^3$. The resulting configuration space for all three-dimensional transformations can the be defined as

$$C = \mathbb{R}^3 \times \mathbb{RP}^3 \ . \tag{11.3}$$

From definition 11.3 it is possible to define two other sets, namely the *free configuration space*, C_{free}, and the *obstacle region of the configuration space*, C_{obst}, simply being the complementing set of C_{free} [79]:

$$C_{\text{free}} := \{\, \eta \in C \mid \mathcal{A}(\eta) \cap \mathcal{O} \ = \ \varnothing \,\} \tag{11.4a}$$
$$C_{\text{obst}} := \{\, \eta \in C \mid \eta \notin C_{\text{free}} \,\} = C \setminus C_{\text{free}} \ , \tag{11.4b}$$

where $\mathcal{A}(\eta) \subset \mathcal{W}$ is the closed set of points occupied by the robot when transformed to configuration η.

11.1 Continuous-Space Path Planning

After defining these sets of transformations and representations, the path planning problem can be defined as the search for a set of transformations, $\mathcal{T}$, that transforms the robot from an initial pose, η_i, to a goal pose, η_g:

$$\mathbf{T}(\mathcal{A}, \eta_i) \overset{\mathcal{T}}{\longmapsto} \mathbf{T}(\mathcal{A}, \eta_g) \ .$$

This formulation is more closely related to the act of *motion planning*. For the case of basic path planning, this can be simplified to the problem of producing one or more *flyable* paths, $r(\varpi)$, connecting η_i and η_g such that [89]:

$$\eta_i \xmapsto{\pi(\varpi)} \eta_g$$
$$\eta_i(x_i, y_i, z_i, \phi_i, \theta_i, \psi_i) \xmapsto{\pi(\varpi)} \eta_g(x_g, y_g, z_g, \phi_g, \theta_g, \psi_g) \ , \tag{11.5}$$

where ϖ represents the path parameter, e.g. a length variable. A flyable path here refers to a path that satisfies the given vehicles kinodynamic constraints [82]. The path planning problem can then be boiled down to generating a collision-free, continuous path, $\pi(\varpi)$, from η_i to η_g, defined as [79]:

$$\pi : [0, 1] \mapsto C_{\text{free}} , \quad \pi(0) = \eta_i, \ \pi(1) = \eta_g \ , \tag{11.6}$$

where Π_f is the set of feasible paths defined as

$$\Pi_f := \{\pi(\varpi) \in \Pi \mid \pi(\varpi) \in C_{\text{free}}, \ \forall \ \varpi \in [0, 1] \} \ .$$

A much used strategy for solving continuous-space problems is to transform the model into a discrete-space model. The two main strategies of performing this transformation is known as *combinatorial-* and *sampling-based* planning.

Combinatorial planning here refers to the branch of mathematics called *combinatorics*, which focuses on the study of countable discrete structures, including the field of graph theory. Combinatorial planning tries to capture the information in the state space by partitioning the free space into an exact representation using discrete data structures. By using exact representations — e.g. visibility graphs — they attain the property of completeness, i.e. if the problem has a solution it will find it or correctly conclude that no solution exist. The general approach of a combinatorial planner is to first compute a representation of C_{free} without approximating, then applying an optimal search algorithm to find an optimal path.

Sampling-based planners, on the other hand, does not explicitly characterise the free space or the occupied space. These methods lets a collision detection algorithm decide whether or not a given configuration lies in C_{free}. These planners then incrementally search the free space for a path, gradually revealing more using an obstacle detector. These planners do not rely on building a complete map, meaning they don't compute more than they have to. This makes these planners more suitable for high-dimensional problems. Two popular methods based on this scheme is the rapidly exploring random tree (RRT) and probabilistic road map (PRM).

11.1.1 Optimal Path Planning

Finally, the act of path planning is usually focused on calculating the optimal path with respect to path length. In many cases, however, it is favourable to incorporate more than just path length into the optimization objective, such as travel time or obstacle clearance. In general, these optimization objectives can be described as continuous and differentiable cost functions $c : C \rightarrow \mathbb{R}^+$, assigning a cost value to each configuration, or state, $\eta \in C$

[90]. Furthermore, if a monotonic and bounded path criteria function $c_p : \Pi_f \rightarrow \mathbb{R}^+$ can be defined, the process of finding the optimal path $\pi^* \in \Pi_f$ w.r.t. c_p results in finding the path π satisfying

$$c_p(\pi^*) = \min \left\{ c_p(\pi) \mid \pi \in \Pi_f \right\} \ . \tag{11.7}$$

By using this formulation, optimal paths in relation to path length or any other definable criteria satisfying the stated preconditions, can be found.

12 | Path Planning in 3D Space

In recent years, a considerable amount of research has been conducted on robotic path planning, resulting in a multitude of proposed solutions. The earliest methods were quite computationally expensive, like the one introduced by [91], solving the piano mover's problem in doubly-exponential time, making them unsuitable for online applications. These early solutions were examples of so-called *complete* algorithms. A complete algorithm either finds a solution or, correctly, states that no solution exists.

The algorithm proposed by Schwartz and Sharir in 1983 [91], using Collins decomposition, is an example of what is often referred to as a *combinatorial* method [80]. These methods solve the planning problem by calculating paths through the continuous configuration space. By doing certain approximations, i.e. relaxing the notion of completeness, the more efficient *sampling-based* algorithms were developed. These methods reduces the computational cost by avoiding explicitly constructing the obstacles in the configuration space. Instead, these methods rely on collision detection-based sampling schemes, often based on random sampling. Although this simplification means the loss of algorithmic completeness, they manage to attain *probabilistic completeness*, as the probability that the algorithm finds a solution converges to one as the number of samples increases.

Due to their efficiency, these sampling-based methods have become the go-to solution for many modern robotic applications. This has, in turn, lead to a myriad of different variants being developed, each with specific advantages and disadvantages. Some methods obtain computational efficiency even in cluttered environments or for robots with a high number of DoF, while others are developed with the intention of obtaining faster convergence to the optimal solution. More and more modern applications, however, push for the need for efficient and safe three-dimensional planning systems. In three dimensions, many established and computationally feasible methods become impractical without modifications due to the expanded space. The specific vehicle dynamics, environment complexity, and additional operation-specific constraints, then weigh heavily in on which methods are viable.

Looking at autonomous underwater applications, the robot is ideally capable of performing long-range missions without human interaction. Planning a new path in the case of unforeseen events must happen quickly while still providing a guaranteed safe path. The planning system is, therefore, required to be *online*, while also, preferably, taking energy consumption into consideration — either directly as an additional objective, or through minimizing path length and actuator strain. The underlying assumption for the path planning, in the context of this thesis, is a partial or fully autonomous underwater robot with a SLAM system capable of creating a continuously expanding map. The main objective of this chapter is then to create a short survey of some of the most promising state-of-the-art methods in recent literature and, based on the constraints of the stated scenario, implement a path planning system able to integrate with the SLAM algorithm and control system treated in part III and part VI.

A secondary objective concerns the development of a sub-system to exhaustively explore a map in conjunction with running SLAM and object detection systems. Therefore, different strategies for autonomous environment exploration based on waypoint generation will be discussed in chapter 13, with the implementation details and simulations results of both of these modules presented in chapter 14.

This chapter is organised as follows. Section 12.1 gives an introduction to the traditional path planning problem as well as some general planning strategies. In section 12.2, different ways of representing the environment to allow for efficient planning are discussed. Several well-established path planning methods, and their more recent state-of-the-art variants, are treated in section 12.3 and section 12.4, focusing on combinatorial and sampling-based methods, respectively. Clustering algorithms are made use of throughout this part and their use-cases in the context of path planning, among others, are examined in section 12.5, before the concept of path criteria is examined in section 12.6. Finally, this chapter ends with discussing context-awareness and its relation to path planning in section 12.7, and a short summary of the most important topics is provided in section 12.8.

12.1 Traditional Planning Methods

The path planning problem formulation for practical implementation on mobile robots was traditionally rooted in two dimensions — largely due to the fact that robotic motion was simpler and consisted of actuation in a strictly 2D environment. One of the main factors contributing early in the field of path planning is the work put forth by Lozano-Perez and Wesley in 1979 [92]. They presented a planning algorithm designed for generating collision-free paths given a polyhedral object in a known, polyhedral environment. This method, while possible to extend to 3D space, was relatively time inefficient, but helped spark interest in the field, resulting in numerous planning methods being developed in the following period [91], [93], [94]. Many of these methods were so-called *complete* methods[1] and many shared the common factor of planning using a retraction of C. Solving the planning problem has, in general, been proven PSPACE-hard, with a complexity increasing doubly exponentially with the robot's number of DoF [95]–[97]. Such increases in complexity

[1]Planning algorithms are characterized as complete if they are able to find a solution or, correctly, conclude that no solution exists

can quickly become troublesome in problems of higher orders, such as when planning for multi-jointed robots or for applications where online performance is required. Especially if one tries to solve it using a C representation that represents the environment as closely as possible. The proposed solution to this was then to relax the notion of completeness, resulting in what is often referred to as *sampling-based* methods [81], sampling C instead of creating an exact representation or a retraction. This leads to the obvious loss of algorithmic completeness. They do, however, keep the property of *probabilistic completeness*, meaning that the solution converges to the optimal solution as the number of samples approaches infinity.

Many different approaches have been proposed to solving the path planning problem, therefore, only a subset of these will be more closely evaluated. Sampling-based methods, especially, have been subject to much research and have seen a lot of use in practical implementations and experiments in recent years. This is largely due to their relative simplicity, ease of modification and applicability to different problems. Due to this, sampling-based and combinatorial methods will be the main focus of this project.

Other methods that are worth briefly mentioning, for completeness, include *artificial potential field*, *numerical optimization*, and more modern *AI-inspired* methods.

Artificial potential fields

This method was first introduced by Khatib and uses a workspace representation in the form of an artificial force field [98]. In this force field, obstacles exert repulsive forces, while the goal exercises an attractive force. This attractive potential can be represented in many different ways, where the original formulation uses the straight forward potential function $U_a(\mathbf{x}) = \frac{1}{2}\rho(\mathbf{x} - \mathbf{x}_g)^2$, where $\mathbf{x}_g$ is the goal pose and ρ acts as a scaling factor.

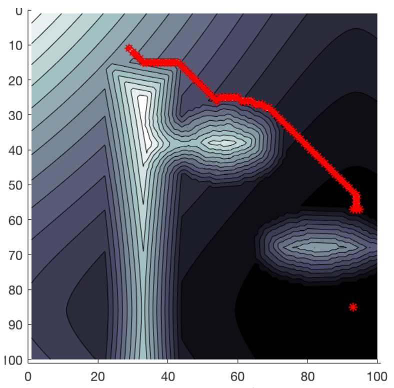


Figure 12.1: Example 2D simulation of a potential field planner getting stuck in a local minimum, failing to reach the goal.

The biggest advantage of this method is its computational efficiency, which allows for online implementations. Their main drawback, however, is being susceptible to local minima. This problem only becomes greater as the dimensionality of the problem increases. Because of this, artificial potential field planners are mainly used as local planners in modern literature [99] together with a global planner such as a visibility graph-based method [100]. Therefore, artificial potential fields will not be explored in more depth in this thesis.

Numerical optimization

Optimization methods are a natural choice for many applications when concerned with calculating optimal paths with respect to some quantifiable measure — most often path length. To solve the problem in this manner, specific cost functions are defined, for example combining the running cost and the terminal cost [101]. Minimizing such a cost function then results in a set of states and inputs based on some vehicle model. A big advantage with these types of methods is that the optimization objective can be chosen such as to minimize different quantities, such as resource management [102] and energy consumption in presence of ocean currents [101], [103], [104]. The difficulties then becomes accurately modelling the vehicle itself, as well as any environment effects that is to be considered. Of course, problem simplifications can be made to make for a more general planning system — which also reduce computation times — but this reduces the overall optimality of the solutions, which is not ideal.

AI-inspired methods

Planning methods inspired by AI and deep learning are closely related to numerical optimization, and thus share most of their drawbacks. These types of methods are more easily generalizable to a larger set of problems compared to pure optimization methods, under the assumption that large enough datasets that capture the necessary information are available. When applied to path planning problems, these networks are usually based on visual sensors, and thus consists of CNNs. These methods have been successfully applied to real-world UAV tasks, such as UAV navigation based on directly assessing monocular images [105]. Specific approaches, such as the *Motion Planning Netwrk* (MPNet) [106] show considerable promise, even outperforming certain state-of-the-art sampling-based methods in simulations. It is worth mentioning that these published MPNet results compares the network running on a powerful GPU versus the Python-implementations of the aforementioned *state-of-the-art* methods. However, evaluating such a network on underwater data would be of great interest, but is left as future work for now. DL methods have further potential use-cases related to planning through context awareness [107], which will be discussed in some more detail in section 12.7.

Probabilistic roadmaps

Probabilistic roadmaps (PRM) is one of the most popular methods based on the sampling planning paradigm, and was first introduced by Kavraki et. al. [108], [109]. This method consists of two phases: a generation phase, and a query phase.

In the generation phase, a *roadmap* (definition 12.1) is created through random sampling of C. These samples are then connected by means of a local planner — usually a straight line connection, but these planners can vary, even using control-based feedback theory [110]. In the query phase, the initial and goal states are added before the generated roadmap is searched. Being a sampling-based roadmap method it bridges the gap between the Voronoi diagram and the RRT family of planners — both of which will be discussed more closely in this chapter.

PRMs was deemed unsuitable for the case at hand due to three main drawbacks. Firstly, the roadmap generated by PRMs lack the clearance property of several other roadmap methods. Additionally, probabilistic roadmaps assume a static map, which poses problems in an exploration setting, where the map definitively is not static. Finally, and perhaps the main reason to disqualify the PRM, is the fact that there are scenarios where the generated graph fails to completely cover the environment. An example of this can be seen in fig. 12.2, especially with the subregion around $(x, y) = (150,780)$. This is not just *bad luck*, as proofs have been published guaranteeing failure in certain cases [112], which is troublesome when trying to guarantee a robust planning system.

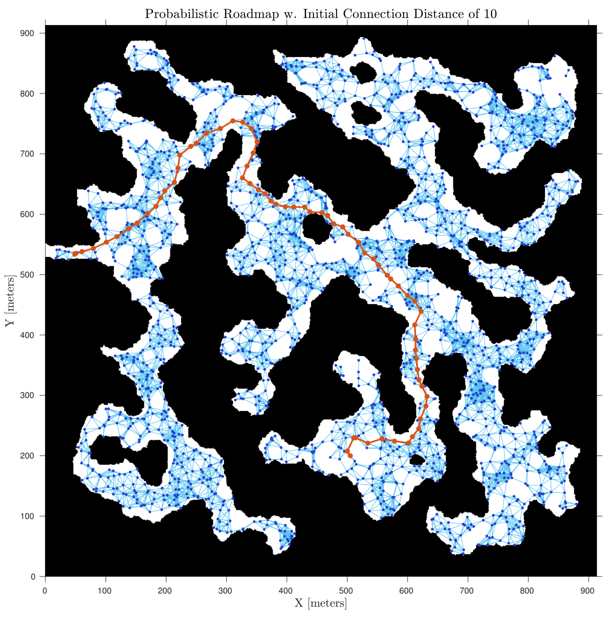


Figure 12.2: Probabilistic roadmap of an almost-closed 2D environment with the resulting solution path to the exist. Calculated using MATLAB's robotics toolbox [111].

Although admittedly having their use-cases, the last four methods will not be explored in much more depth in this thesis due to the reasons stated above. Instead, the expansive set of combinatorial methods — due to their completeness and exact representations — and specific sampling-based planners — by virtue of their relative simplicity and efficiency — will be in focus. However, the efficiency and solution quality of the path planning method is heavily impacted by the chosen environment representation. Because of this, the path planning problem can loosely be described as a two-part problem: (*i*) modelling the environment, (*ii*) calculating a path through the constructed representation subject to the given constraints of the problem. This chapter will introduce some of the most used approaches for representing the environment, before delving deeper into specific state-of-the-art planning methods that have obtained good results either theoretically and in simulations or in various real-world applications.

12.2 Representing the Environment

Accurately describing the environment the robot operates in, $\mathcal{W}$, is essential for the path planning process. The performance of the system, as well as the quality of the calculated path, is severely impacted by the manner in which the world is represented. Choosing sparse representations will result in more time-efficient planning, but the path itself can become more optimal if the representation better matches the actual surroundings. In general, these representations can be categorised as *topological* or *metric*. A topological representation describes the environment without explicit references to numeric data. I.e. the environment is modeled as a set of nodes, possibly describing features, with edges between them containing relational information. Metric mapping, on the other hand, directly utilises a certain data structure in which waypoints can be explicitly stated based on global data. Due to the nature of metric representations, they tend to favor optimal planning methods, which in turn has lead this to being the most popular way to store environment information.

Numerous representations exist, therefore the focus of this report will be on some of the more popular and promising techniques. The next sections will briefly discuss the methods most prominent in related literature, namely the *occupancy grid* and the *roadmap*.

12.2.1 Occupancy Grids

Occupancy grids [113], also known as occupancy maps, are perhaps the representation most used in robot path planning to date, largely due to their simplicity. An occupancy grid structures the workspace into a discrete grid with a specific resolution. The occupancy information is usually stored in one of two ways: *(i)* binary *(ii)* probabilistic. Binary grids store the occupancy information of each cell simply as *true* or *false* depending on whether or not the cell is occupied by some obstacle. The probabilistic map is slightly more complex and is the most common representation. Each cell in this grid is categorised as *free*, *occupied* or *unknown*, depending on the estimated probability of obstacles occupying the specific cell. Probability values sufficiently close to 1 indicate a high certainty for a cell being occupied, whereas cells with probabilities in a neighbourhood around 0.5 are marked as *unknown*. These occupancy probabilities can be initialised using a priori knowledge – e.g. from a pre-built map – and then be updated online based on incoming sensor information. In an online mapping setting, the probabilities are calculated as

$$p(m_i \mid z_{1:t}, x_{1:t}) = \prod_i p(m_i \mid z_{1:t}, x_{1:t}) \ ,$$

where $p(m_i)$ denotes the probability of a specific cell i being occupied, $z_{1:t}$ being the set of measurements up until time t and $x_{1:t}$ is the set of robot configurations, or states, up to time t.

Grid representations are not without disadvantages, however, with the main drawbacks being the denseness of the representation and the fact that it suffers from *digitisation bias* [114]. The denseness of the grid simply results in a larger number of nodes having to be evaluated during a search compared to e.g. sparse graphs. Due to this, occupancy maps structured as pure three-dimensional grids are better for smaller environments or in applications where grid resolution is of less importance. Digitisation bias[2] refers to the information loss caused by quantising the real world into grids. In other words, grid cells including just a sliver of an obstacle can be marked as either occupied or free — depending on the probability threshold — neither of which is completely true. This means the the optimality of the path, e.g. in respect to path length or obstacle clearance, is directly dependent on the resolution of the grid used.

The digitisation bias problem can be largely overcome by the use of more sophisticated data structures when generating the occupancy map. One data structure commonly used for this — that also has seen extensive use in search and optimisation algorithms, as well as in image processing — is the octree. Octrees are most easily explained through their two-dimensional equivalent: the quadtree. Quadtrees structure the data by dividing a square portion into four smaller squares, hence quadtrees, and recursively subdividing into four quadrants. In this representation, more "interesting" parts of the environment have a deeper subdivision. E.g. an image-based map with clustered obstacles can result in some parts of the quadtree being very deep with each sub-cell representing one pixel, whereas parts of the map with no obstacles might be represented by larger sub-cells of four pixels, as illustrated in fig. 12.3. Quadtrees are mainly used to partition two-dimensional space, e.g. to use as 2D navigation maps for marine surface vehicles (MSVs). To accommodate the third dimension, the octree encoding was developed [115]. This data structure is analogous to the quadtree, but partitions the space into octants instead of quadrants.

The main advantage of these structures compared to the regular grid, is that the problem of too crude a quantization is reduced due to the increased resolution of areas in, or close to, C_{obst}. As a result, the quad-/octrees generally consists of fewer cells than their grid counterparts. A comparison between a 2D occupancy grid and a quadtree representation is shown in fig. 12.3.

[2]Sometimes also referred to as *resolution completeness*, but this term will be reserved its original definition by Latombe [79] to avoid any confusing and overlapping terminology.

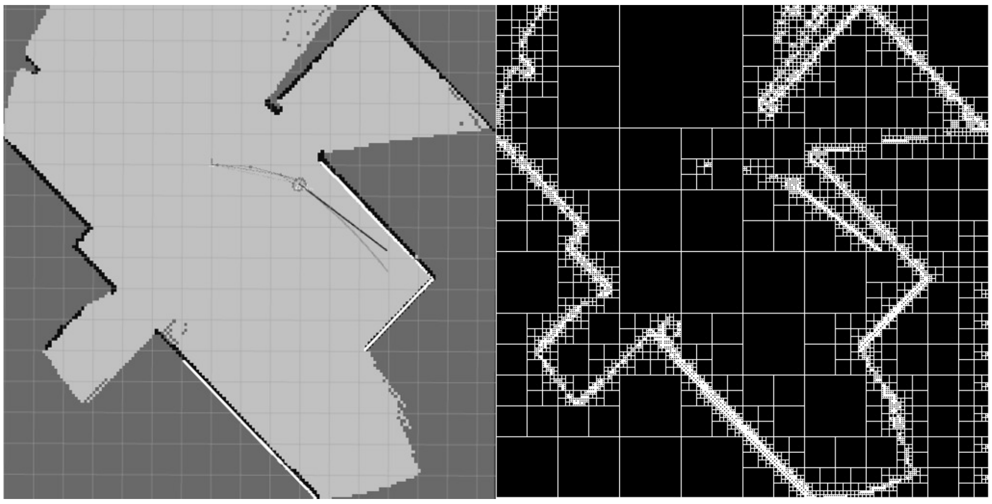


Figure 12.3: Comparison between an occupancy grid and a visualization of the correseponding quadtree representation. Empty areas results in larger cells, giving a more compact representation wrt. storage. *Left*: Screenshot from a 2D SLAM simulation based on a 2D occupancy grid. *Right*: Screenshot of the same map represented using a quadtree. (The grid overlay in the left image is not indicative of the occupancy grid resolution.)

12.2.2 Roadmaps

Graph-based roadmaps have seen much use in path planning literature. Different variants exist, but all share the common concept of building a set of edges and nodes spanning the workspace $\mathcal{W}$ while circumventing polygonal obstacles. Roadmaps can be defined as follows:

Definition 12.1. ***Roadmap***
Let $\mathcal{M}$ be a graph mapping into C_{free} and let $\mathcal{S} \subset C_{\text{free}}$ represent the set of all reachable points in the graph and η represent the robot configuration.[3] $\mathcal{M}$ is said to be a roadmap if the following conditions are satisfied:

- *Accessibility* - There exists a path $\pi : [0,1] \mapsto C_{\text{free}}$ from $\eta \in C_{\text{free}}$ to some $s \in \mathcal{S}$ $\forall\, \eta, s$
- *Connectivity* - If there exists a path $\pi : [0,1] \mapsto C_{\text{free}}$ s.t. $\pi(0) = \eta_i$ and $\pi(1) = \eta_g$, then there also exists a path $\pi' : [0,1] \mapsto \mathcal{S}$ s.t. $\pi'(0) = s_1$ and $\pi'(1) = s_2$.

From definition 12.1, the accessibility-condition implies that it is always possible to connect some initial state and a goal state to $s_1, s_2 \in \mathcal{S}$, respectively, whereas the connectivity-condition ensures algorithmic completeness by assuring no missed connections. This roadmap can be built in many different ways, with some of the most prominent methods including *cell decomposition*, (*reduced*) *visibility graphs*, and *Voronoi diagrams*.

[3]This set of all reachable points in a graph is sometimes referred to as the *swath*.

Cell decomposition methods divide the map into, mainly, vertical or horizontal cells. A safe path can then be found by choosing waypoints that coincide with the cells' centroid. While being a straightforward representation, they pose certain problems, including combinatorial explosion, limited granularity and the possibility for infeasible solutions. These complications results in methods based on cell decomposition not being discussed further in this thesis; readers are referred to [116] for a more detailed run-through of these problems.

Another strategy is to generate a graph from the starting pose to the goal by representing obstacles as polygons and adding all obstacle vertices to the graph that have visibility from previous vertices. Such a graph is called a visibility graph [117]. The reduced variant removes any edges that will never be used, i.e. edges that connects directly to obstacles. This results in an increase in the efficiency of which the shortest path can be computed. Visibility graphs do suffer one big drawback, however, namely that the vertices defining the obstacles are directly present in the graph. This means that the calculated path touches the obstacles and is in need of post processing. A roadmap method that circumvents this, is the Voronoi diagram.

Voronoi diagrams

As mentioned, the main motivation behind using graph representations for path planning, is that they reduce C_{free} into a subset of connected vertices. A search through the resulting subspace can then be performed to find an optimal path through the graph. Specific to the Voronoi diagrams is that they partition the environment into convex regions, or *Voronoi cells*. Each polygonal region as one *generating point*, p_i, and every point inside a given cell is always closest to that cell's corresponding generator point than to any other generating point. The general Voronoi diagram can be defined as in definition 12.2. In the case where the metric $d(\cdot)$ is chosen as the Euclidean distance, and obstacles are used as generator points, the Voronoi diagram becomes a *maximum clearance roadmap* [4].

Definition 12.2. ***Voronoi region*** [118]
Let

$$d(x, \mathbf{A}) = \inf \{d(x, p) \mid p \in \mathbf{A}\}$$

denote the distance between the point $x \in X$ and the subset $\mathbf{A}$, where $p_i \in \mathcal{P} \subset X$ is a generator point. The Voronoi region is then defined as

$$\mathbf{R}_k = \left\{x \in X \mid d(x, \mathcal{P}_k) \leq d(x, \mathcal{P}_j) \,\forall\, j \neq k\right\} \ .$$

Using Voronoi diagrams in path planning to represent the environment provides a graph

[4]For readers versed in the field of computational geometry, it can be mentioned that the Voronoi diagram is, in essence, the discrete form of the medial axis.

in which paths that are maximally distant from obstacles can be calculated. This results in very safe paths, which can be preferable for autonomous operations. Algorithms have been developed that generate d−dimensional Voronoi diagrams based on convex hulls in $O(n \log r)$ time for $d \leq 3$ [119], where r is the number of already processed points in the diagram. This allows efficient generation as long as n does not grow too large. For mapping and exploration in three dimensions, however, the number of points can be expected to grow quite large. Voronoi diagrams representing d−dimensional space require $O(n^{\lceil \frac{1}{2}d \rceil})$ storage space. This requirement is linear in the 2D case, but exhibit polynomial growth for $d \geq 3$.

Other methods of generation include skeletonization methods, for example through signed distance fields [120]. Planning over the generated skeleton graph can result in very efficient path planning [120], but requires efficiently processing sensor data and the possibility of incremental graph generation, which is hard to generate online, especially for larger, more complex environments.

As far as map representation goes, the main representation used in this particular project is the occupancy grid. This is largely due to the ease of which frontier regions are incorporated, as well as the fact that it needs little pre processing when combining point clouds directly with OctoMap [121].

12.3 Combinatorial Methods

Combinatorial methods have the nice property that they explicitly construct a representation of C which allows for optimal, deterministic path searches through C_{free}. This allows the method to either find an optimal path, or correctly conclude that no solution exists for the current problem at hand.

Combinatorial methods rely on explicitly calculating a retraction of the environment, most often represented as a roadmap, before performing the path search. Different methods of doing this was briefly discussed in section 12.2.2. Seeing as safety is the main priority of autonomous underwater operations, especially, an obstacle clearance-focused approach was deemed the best fit. Due to this, the Voronoi diagram was deemed a good potential planning method for underwater path planning.

12.3.1 Voronoi Diagrams in Path Planning

The Voronoi diagram has shown promising results in recent literature when applied to marine environments [118], [122], as well as showcasing its feasibility when applied to three-dimensional problems [123]. The method presented in [123] creates the Voronoi representation using the quickhull algorithm [119] by choosing the points defining obstacle bounding boxes as generator points. The diagram is further limited to a region of interest by placing points at random along the edges of the chosen region. The Yen-modification of Dijkstra [124] is then applied to find the k most optimal paths. Thus, if the optimal path is found to be inaccessible, the next optimal path is chosen. The Euclidean distance is used as the Voronoi metric, resulting in the roadmap and resulting paths being piece-wise linear. This poses a problem for the control system, as further discussed in section 12.6. Therefore, 3D Dubins paths are used to smooth the path further. For collision detection and -avoidance, this paper suggests using a local Voronoi diagram calculated around the moving obstacle. The local diagram generated in the replanning subspace connects to the global roadmap, with the local bounding box being estimated based on the *closest time of approach* and *closest point of approach* [123]. The process of replanning around moving obstacles is not relevant to this thesis, however, as only SLAM is used for map updates, hence moving obstacles will be ignored.

The applicability of a Voronoi-based planner in a 3D environment is showcased in [123], but is implemented under a very important assumption: the entire map is assumed known and static. For underwater exploration, this assumption will not hold. When obstacle positions are unknown, it is necessary to estimate individual obstacle placements online based on incoming sensor or SLAM data. This can be achieved by using a clustering algorithm to estimate obstacle clusters from a point cloud, e.g. by using DBSCAN [125]. This concept is further discussed in section 12.5.

In the complete system, the intended input to the path planner is the SLAM-map. This map is represented as a fairly sparse point cloud generated from possibly noisy data. Directly generating a Voronoi diagram from this 3D point cloud could result in a number of edges intersecting with C_{obst}, and would be in need of post-processing. Instead of directly using points in the point cloud as generator points, the graph could be generated through skeletonization techniques such as *signed distance fields*. A distance field describes the inter-surface space, with each point containing the distance from that point to the closest surface. *Truncated signed distace fields* (TSDF) are based on the relative distance between the robot sensor and the obstacle along the sensor's line-of-sight [126]. A simple example illustration of a TSDF is shown in section 12.3.1. TSDFs are relatively efficient to calculate, seeing as they only operates within a truncated area. They can therefore be used to further estimate more complex distance fields, such as the *euclidean signed distance field* (ESDF), more efficiently than calculating the ESDF directly. The ESDF is related to the Voronoi diagram as they both can be used to estimate the medial axis, providing a maximum obstacle clearance roadmap of a given environment. An approach to generating such a sparse graph through the use of TSDFs and ESDFs is presented by Oleynikova et. al. [120], [127]. The generalized Voronoi diagram (GVD) is calculated from the incrementally generated ESDF. This GVD is then further thinned, before the skeleton is extracted, preserving the best nodes and attempting to connect any unconnected subgraphs. The best nodes are chosen as the nodes in each neighborhood that are maximally distant from obstacles, and are represented using a k-D tree.

This method further demonstrates the efficiency of planning using a sparse graph combined with a heuristic graph search algorithm. Experiments exemplified in [120] show that A* combined with a sparse graph produced a path 800 times faster than the standard RRT*. The big drawback with this method lies in the graph generation. Generating the sparse graph in the way put forth in the paper, while being relatively efficient, needs offline generation of the sparse graph. A strategy to increase computational efficiency and allow for unknown obstacles could be to use efficient clustering methods together with convex hull algorithms to reduce the number of generator points in the resulting Voronoi diagram. This approach, and its implementation, will be further discussed in section 12.5 and chapter 14.

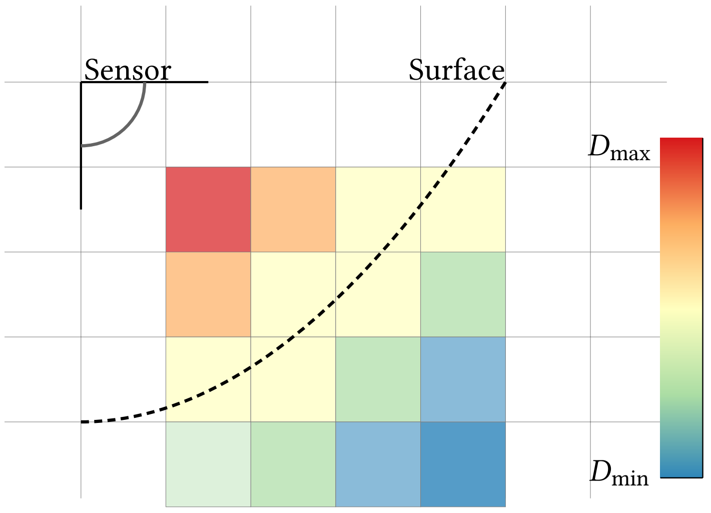


Figure 12.4: Illustration of how the TSDF allocates distance values. Red cells indicate positive values whereas cells on the blue side of the spectrum indicate negative samples.

12.4 Sampling-based Methods

As briefly mentioned previously, the main motivation behind the development of sampling-based methods was to solve the path planning problem more efficiently than the combinatorial methods were able to. Seeing as more and more applications required online planning, which in turn meant having to deal with stricter hardware limitations, the C-space formulation needed to be relaxed. The primary school of thought was then to accomplish this by performing random sampling in the chosen environment representations. Hence the name sampling-based methods. This set of methods has been of much interest in recent decades, and have thus been subject to many different approaches and seen many modifications. This section treats different state-of-the-art variants of one of the most popular sampling-based methods in literature and its variants and applicability to online, underwater path planning.

12.4.1 The RRT Family

The perhaps most popular sampling-based path planning approach to date is the *rapidly exploring random tree* (RRT) method. RRTs have seen extensive use and have been researched heavily in recent literature. These types of algorithms build a tree structure by randomly sampling leaf nodes from C_{free}, favoring the expansion towards the larges Voronoi regions currently in C [128]. The underlying concepts of the RRT are fairly simple, which has been a major factor leading to a huge amount of offspring variants being developed, some of which will be discussed in more detail later in this section.

The RRT algorithm starts by initialisizing the current pose $x_i \in C_{\text{free}} = X_{\text{free}}$[5] as the root node. After initialization, the tree expands by randomly sampling X, adding the new nodes $x_{\text{new}} \in X$ iff. $x_{\text{new}} \in X_{\text{free}} \subseteq X$ and the edge connecting x_{new} to the existing tree is collision-free. Similarly to the PRM, the sampling can be performed through straight line-sampling, or it can be *steered*. Such a steering function chooses an input u that results in an $x_{\text{new}} \in X$ that is closest to the randomly sampled state through evaluating eq. (11.1). Using such a steering function results in a tree that gives both collision-free and dynamically feasible paths according to the modelled vehicle dynamics. By nature of the performed expansion, the RRT algorithm is biased towards unexplored parts of C [128]. To increase the convergence to the specified goal state, a goal bias that forces the tree expansion towards the goal at certain intervals can be included. A nice feature of the RRT tree is that it is always connected as it always expands from the current tree. This results in a less memory requirements as compared to the PRM, due to the fact that PRM computes the complete graph before finding a path. This means that the method of choice is, of course, dependent on the intended application. It is worth noting that, if looking at the datastructures themselves, a tree structure is easier to maintain than a graph with respect to memory requirements. As a side note, this argument also holds for the graph-version of RRT, the *rapidly-exploring random graph* (RRG) [129]. Additionally, regarding the intended application, RRTs are closely connected to environment exploration, as mentioned. This will be further treated in section 13.1.

Furthermore, seeing as the RRT builds upon a state space formulation, it allows for the inclusion of specific vehicle dynamics with relative ease, resulting in trees that conform to kinodynamic constraints. RRTs suffer at least one drawback, however, namely that the ideal metric ρ is not easily estimated [130]. This is not specific to the RRT-family, however, as

[5]It is worth noting that $C_{\text{free}} = X_{\text{free}}$ for basic path planning

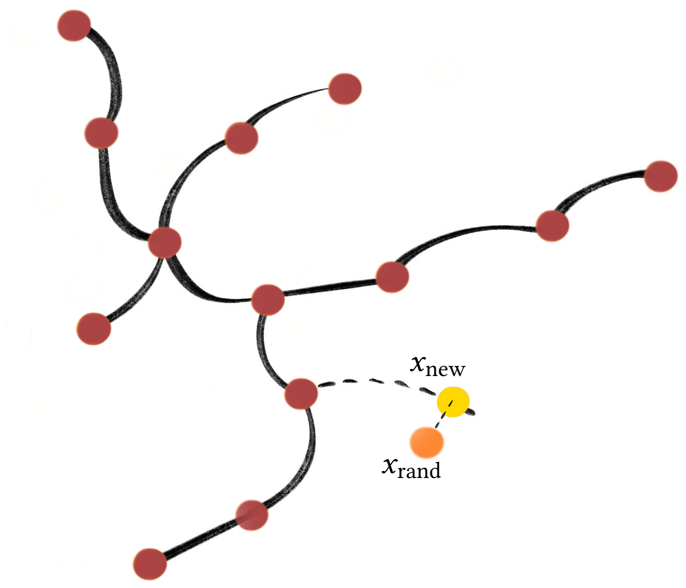


Figure 12.5: Illustration of the RRT expansion process. The random sample x_{rand} (orange) is found by following the steering function. The new node x_{new} (yellow) is found to be the point along the steered line closest to the random sample. This node is then added to the existing tree.

this is true for all randomised sampling algorithms. See [80] for a more in-depth discussion of different metrics and their requirements.

As mentioned previously, the RRT method sparked numerous variants. These mainly focused on the efficiency when applied to higher order problems, increasing the solution convergence, and on fixing some of the introduced problems such as planning in narrow passageways. The list of offspring algorithms have grown quite large, and as a result, only a select few will be treated here. Several literature surveys have been published that treats a large number of the variants published to date, thus the reader is referred to [131]–[133] if an even broader overview is of interest.

One of the first extensions to the basic RRT algorithm was the *bi-directional RRT*. This was developed mainly to overcome the problem of almost-closed environments — also known as the bug-trap problem — and does so by starting two trees. One from the start state and one from the goal state, before alternating the expansion from each of them. Although being a simple extension, it has proven itself to often be more efficient in practice than the base RRT [81]. The more interesting RRT variants, however, come from the endeavor of trying to increase the optimality and convergence of the solution paths.

Sampling-based methods are known to be probabilistically complete, but they are not guaranteed to converge towards the optimal path unless only one solution exists. The first major RRT upgrade to guarantee *asymptotic optimality*, was the RRT*. RRT* builds upon the briefly mentioned RRG algorithm, but includes a near-neighbor search and tree-rewiring to guarantee *asymptotic optimality* [134]. The neighbor search is used to find the best parent node within a spherical area with radius ρ from the new node. The radius is determined by $\rho = \gamma \frac{\log n}{n}^{1/d}$, where n is the total number of nodes currently in the tree, d is the spatial dimension of the problem, and γ is an environment-specific parameter [129]. This same radius is used for tree-rewiring, updating the edges between the encompassed nodes to minimize the edge weights between nodes. This procedure is illustrated in the first three sub-figures in fig. 12.6. These two additions results in increased path quality compared to RRT, due to the local optimizations performed underway. The main drawback of RRT* is the increase in runtime introduced by the additional node evaluations. Through simulations, this increase was roughly tripled, leading to a weighting between path optimality and computational efficiency. Even though RRT* evaluates nodes for rewiring, it does not revisit the already built tree to check for any new obstacles. In the case of a dynamic environment, this can be a problem. In an attempt to address this, while also reducing the linearly increasing memory usage of RRT*, the *RRT*Fixed-Nodes* (RRT*FN) and its dynamic extension (RRT*FND) [135], [136] were introduced.

The name fixed-nodes refers to the fact that the total allowable number of nodes in the tree is static and bounded. By doing this, the linearly increasing memory requirement of RRT* is reduced. This algorithm grows the tree in the same manner until the maximum allowed number of nodes is reached. At this point, leaf nodes are removed as long as the leaf node in question is not the last node on the current solution path. The expansion process is illustrated in fig. 12.6. To allow for dynamic environments, the algorithm will restart if a collision is detected. Instead of restarting with an empty tree, however, the algorithm removes the already visited nodes, as well as the collision nodes, before using the rest in the new expansion process. The simulations performed in [136] indicate that the RRT*FN and RRT*FND outperforms RRT* in both runtime and success rate in dynamic environments. While RRT*FND seem to give significant improvements in runtimes for complex environments it does only obtain marginally faster runtimes for simpler scenarios — even, at times, having a slower runtime than *RRT** due to the initial overhead. A drawback that comes to mind with the fixed-node algorithms, is precisely the fact that the number of nodes must be assigned. This requires some degree of knowledge about the environment such as to have enough nodes to reach the goal. If the number is chosen too small, there is a risk of never reaching the goal simply due to spatial distance.

Instead of focusing on the reduction of nodes to increase runtimes, another approach is to increase the rate of convergence by guiding the search to more quickly obtain a solution. One such example is the *Informed RRT** [137] which makes use of informed sampling, as opposed to the original stochastic sampling, to steer the state sampling. *Informed* refers to the fact that, when a solution is found, little is gained from sampling a new state which results in a less optimal path. Thus, information about the current solution is used to further

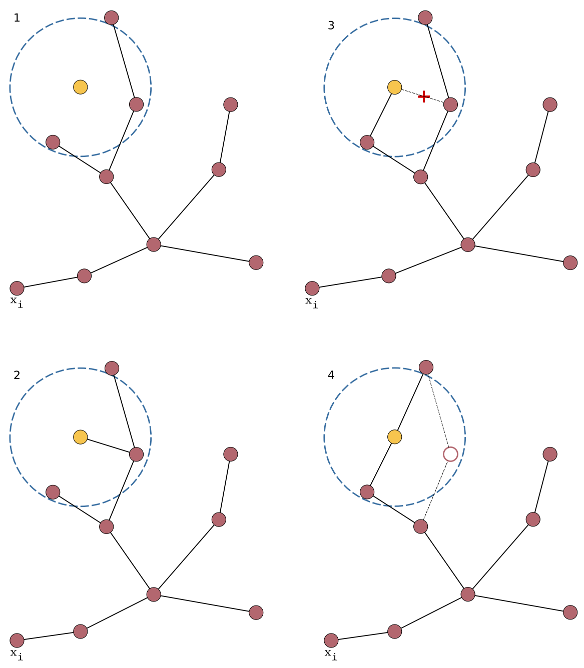


Figure 12.6: Illustration of the initial tree expansion of the RRT*FND algorithm. **1)** A CIRCULAR AREA encompassing the NEW NODE is searched. **2)** The NEW NODE is added to the EXISTING TREE. **3)** & **4)** The rewiring is done by optimising the edges connecting the nodes, before any child nodes connecting a path with a higher cumulative cost is removed.

drive the search. This sampling scheme works by exploiting the fact that, after an initial solution is found, all possible improvements to that solution lie within an ellipse. As the solution improves, the ellipse shrinks. This ellipse is called the *informed subset* [137], and can be defined as

$$X_I := \{\mathbf{x} \in X \mid c(\mathbf{x}) \leq c_{\text{best}}\} \ , \tag{12.1}$$

where $c(\mathbf{x})$ is the cost from x_{init} to x_{goal} and c_{best} is the current best path cost. Obstacles are not encompassed by this informed sampling definition, therefore all states inside X_I can be guaranteed to result in a lower cost. This is the case for informed sampling in the real world, since obstacles are almost always present. However, even though lower costs are not guaranteed for all states inside the informed set, this still allows for a much denser expansion inside a subset of the complete state space. Overall, this results in a greatly increased probability of finding more optimal paths. An example of the different trees generated by RRT, RRT*, and Informed RRT*, is shown in fig. 12.8.

Another method that improves on the RRT* even more is the *Batch Informed Tree* (BIT*). This algorithm also makes use of informed sampling, but improves on Informed RRT* by evaluating part of the state space in batches. BIT* divides the problem into j batches, each with k samples. These batches implicitly define a random geometric graph on which a

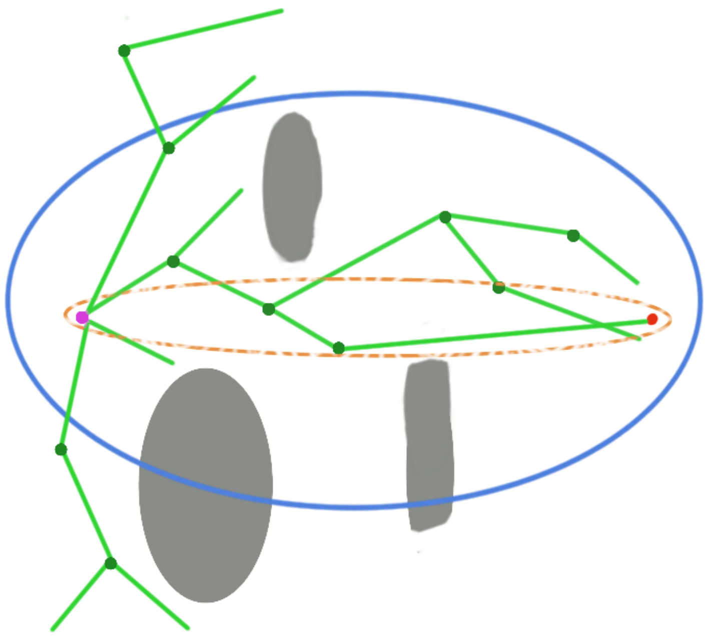


Figure 12.7: Example of how the informed set can evolve through the path searching process. The current tree (green) samples the state space while avoiding obstacles (gray) inside the current informed set (blue). As the process continues and the paths converge closer to the optimum, the informed set shrinks (orange). In this case, the final informed set would converge to the straight line connecting the starting point and the goal (red).

heuristically driven search can be performed to find the optimal path from x_{init} to x_{goal} in the given batch. When the first batch is processed and the current optimal path is found, a new batch consisting of a denser selection of states is drawn before a new heuristic search is initiated. This procedure continues until the solution stops improving or when no more traversable edges exist. These informed samplers were the ones most closely investigated in this thesis. Even more recent variants, such as RABIT* [138], combining local optimization through *covariant hamiltonian optimization for motion planning* (CHOMP) [139] and BIT* for an even further increased convergence rate. Though promising, they were left as possible future extensions.

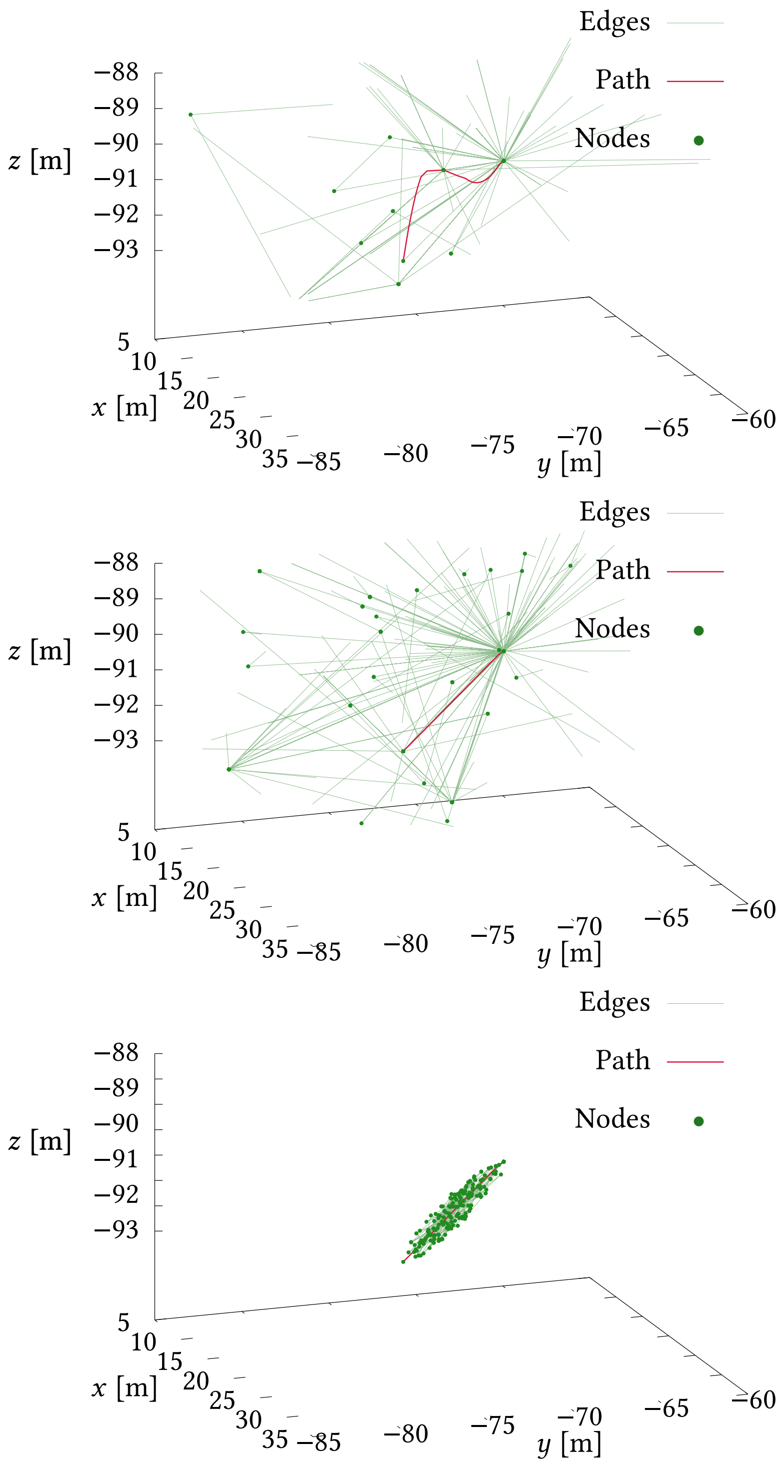


Figure 12.8: Comparison between the resulting tree structure after running RRT, RRT* and Informed RRT*, respectively, in an environment without obstacles (only a subset of the RRT and RRT* graphs are shown). In the same case, due to graph pruning, the BIT* graph reduces to the straight line entirely (not shown).

12.5 Clustering in the Context of Planning

Clustering methods play an important role in many different fields of research, including computer science and bioinformatics, for example, and has a wide variety of use-cases. Clustering is probably nowadays most often associated with machine learning (ML) and falls under the category of *unsupervised learning*. In this sense, clustering seeks to find a *natural structure* in a set of unlabelled data, and then to use this to divide the data into clusters. This dividing structure is extrapolated based on the algorithm's sense of *similarity*. The notion of similarity can have many interpretations, especially when analysing high-dimensional data. In many cases this similarity measure is based purely on the distance measure, i.e. the $L2$-distances between the data points; samples close to one another is then labelled as members of the same class. Seeing as clustering methodology has been around for many decades — see, for example, the work of Cattell from 1943 [140] for a related case from psychology, or MacQueen's [141] or Lloyd's[142] paper for some of the original k-means papers[6] — it is of no surprise that the number of clustering methods rival that of the number of path planning methods. The main reasoning behind this large number of methods is that there is no one universal clustering algorithm that can solve any and all problems. Clustering methods are divided into specific groups based on their method of clustering and what assumptions they make. For example, k-means [141] is an example of a partitioning, or *squared error*-based, clustering method that works by continuously updating a centroid estimate under the assumption that the clusters are inherently spherical. There are obvious drawbacks with this method, but it has still seen much use in literature and performs well under the right circumstances [144], [145]. Due to the nature of these clustering methods, several papers have been published surveying clustering algorithms, trying to answer the question of which clustering algorithm is the most optimal one. The answer, however, is always *it depends* [146]. This holds true for clustering methods, even more so than for path planning methods. Therefore, the choice of algorithm requires good knowledge of the data in question.

Seeing the numerous applications of clustering methods in literature, it comes as little surprise that it also has its uses in path planning. One such example comes from the use of clustering to approximate optimal, discrete paths for mobile robots [147]. This clustering is based on an approached the authors named Γ-clustering and uses a graph to represent the environment. The parameter Γ refers to a ratio variable defined by the ratio between the minimum weighted edge leaving or entering the cluster and the maximum weighted edge already inside the cluster. The practical notion of Γ the becomes a measure of separation

[6]To the authors' best knowledge, the k-means algorithm was independently developed by different authors in different fields, with the earliest documentation being the work of Steinhaus [143] (in French) in 1956, but the name wasn't coined until 1967, by MacQueen [141].

between the vertices in the cluster and the remaining vertices in the graph. While providing a fascinating use of clustering and an interesting take on the path planning problem, this procedure is deemed too computationally expensive in its current state to facilitate online planning.

Another example, more akin to traditional path planning and some of the methods discussed in ??, is the use of clustering to provide a model of the C-space [148]. The idea behind this is to use clusters to calculate an abstract representation of the C-space which acts as something in between combinatorial and sampling-based planning methods. This is also the motivation behind one of the planning methods proposed throughout in this report (see section 14.1), although using a slightly different approach which also takes inspiration from the work of Candeloro and Lekkas [123].

As previously mentioned, different clustering methods work better in certain circumstances given certain assumptions. Under the working assumptions of this report, the perception and SLAM systems provide a 3D point cloud representation of the environment. Furthermore, while clustering methods such as k-means are simple and relatively efficient, obstacles cannot be assumed to be spherical in shape. Due to this, density-based clustering was deemed an appropriate clustering technique. Each of these clusters then represent a single obstacle $o_i \subseteq \mathcal{O} \subseteq \mathcal{W}$. Depending on the perception system and the environment, these clusters can contain thousands of points. If, for example, a Voronoi diagram is calculated based on these clusters — which is essentially the same as just calculating the diagram based on the input cloud directly — the resulting graph would become very dense and extensive pruning would be necessary to be able to use the diagram. As proved in previous research [123], a simpler diagram can be extracted by using boundary points as generator points, simplifying the process immensely. Preceding works do, however, assume a priori knowledge about obstacle positions and their bounding boxes. Therefore, this report seeks to circumvent this assumption, and instead find boundary points incrementally as the robot explores. This can be achieved by using the extracted clusters as input points for a convex hull algorithm. An example of a calculated convex hull from a single point cloud cluster using CGAL's implementation of the quickhull algorithm [119] is shown in fig. 12.9. By then using the points defining the convex polygon as the Voronoi generator points, much fewer points will have to be processed. This strategy is tried illustrated in fig. 12.10 using built-in MATLAB functions. In fig. 12.10a, the Voronoi diagram is generated using all points present in the obstacles. This results in a larger number of evaluations as well as a larger number of edges within the diagram, most of which is present inside the obstacle itself. This would require much pruning to remove all unwanted edges. Instead, fig. 12.10b showcases the Voronoi diagram generated when just evaluating the convex polygon points, and results in a much lower number of edges overall. The added benefit of this is that the convex hull directly represents the obstacle which makes the removal of unwanted edges fairly

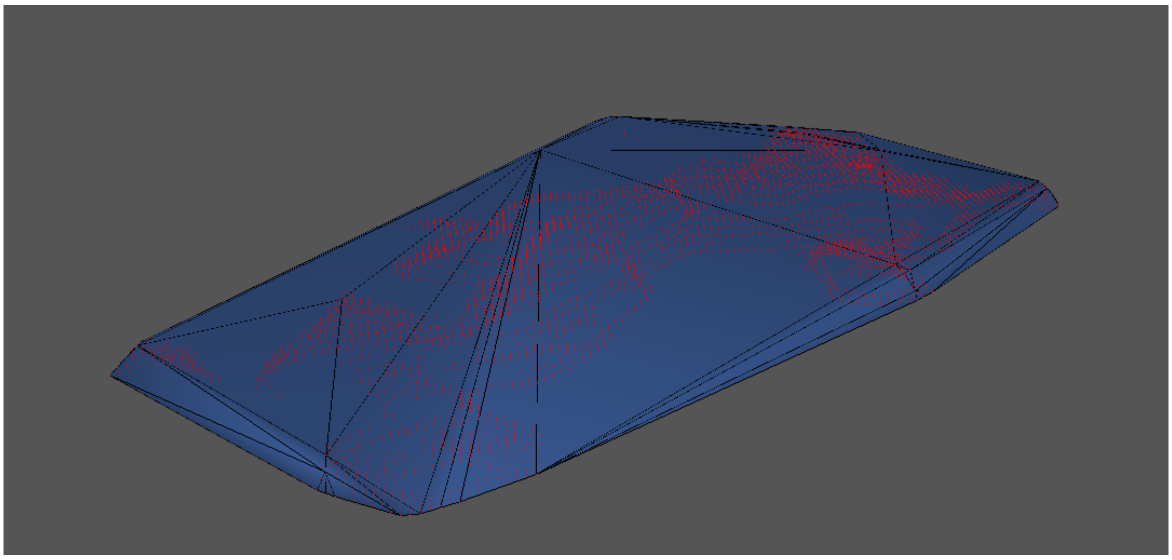
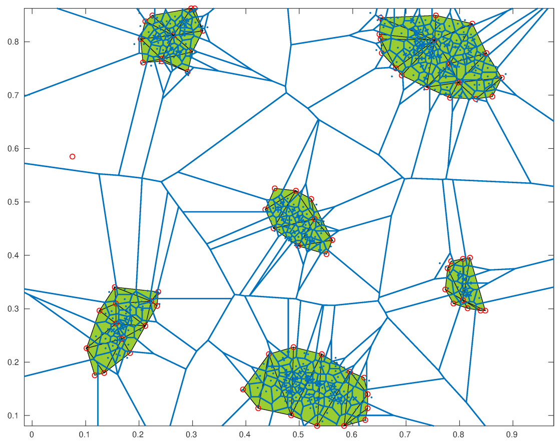
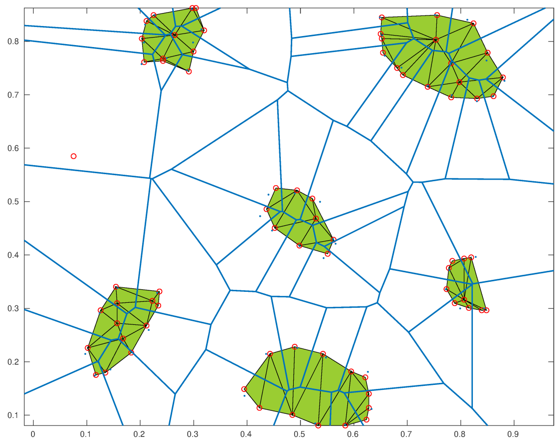


Figure 12.9: Convex hull of a single point cloud cluster calculated using the quickhull algorithm.

straightforward.



(a) Voronoi diagram generated from all observed points.



(b) Voronoi diagram generated based on the polygon points.

Figure 12.10: Example roadmaps showing the 2D Voronoi diagram generated from two sets of points representing the same environment.

It is worth noting, however, that when calculating the convex hull of the clusters, the obstacle representation is simplified. This results in a loss of information and is, in part, the reason behind why this method of planning would fall in between the realms of combinatorial and sampling-based planning. Furthermore, the *tuning* of the clustering algorithm must be addressed. The particular cluster used in fig. 12.9 was found using a Euclidean distance-based clustering method. These types of clustering methods rely on a user-defined tolerance value, which essentially states how far two samples can be apart and still be considered part of the same cluster. As this parameter will depend on the given application, it is necessary to perform practical tests to conclude with a reasonable value. An example clustering using DBSCAN is shown in fig. 12.12.

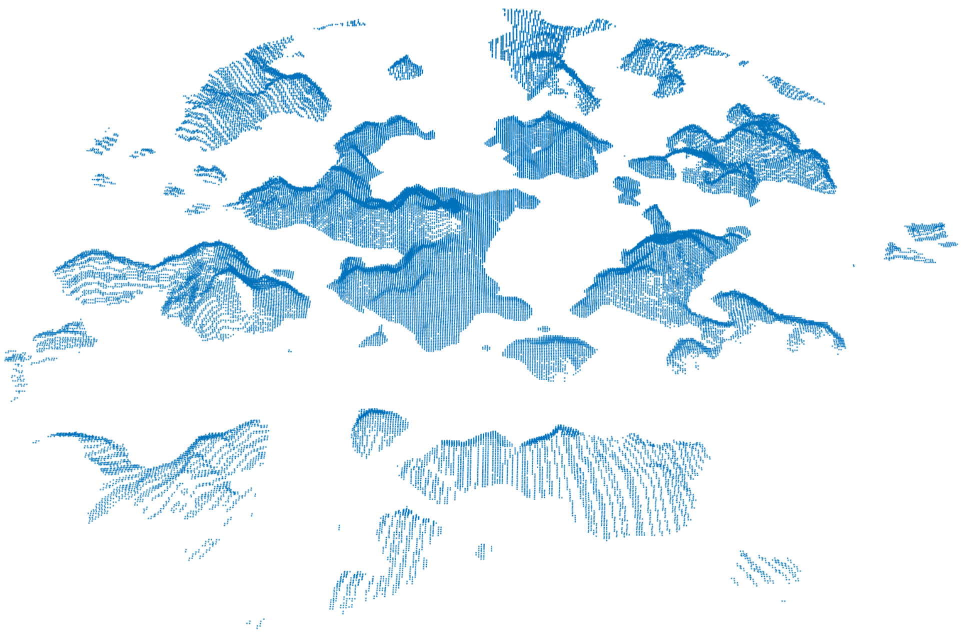


Figure 12.11: Point cloud from a simulated underwater environment with the seafloor filtered out.

With a roadmap generated, the path to the goal state can be found by applying an optimal search algorithm, such as A*. The resulting path will consist of piece-wise linear segments, and is in need of post processing. Different ways this can be achieved, and their pros and cons will be discussed in the following section.

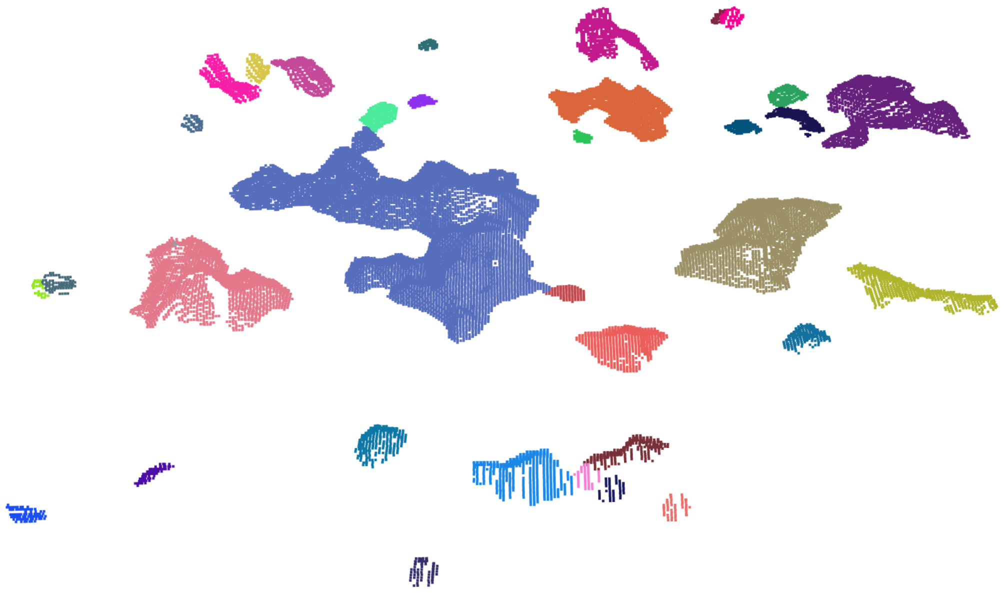


Figure 12.12: Example clustering of the point cloud shown in fig. 12.11 using DBSCAN. Some of the information is counted as noise due to the more sparse point measurements in the lower left corner, especially.

12.6 Path Criteria

Before discussing the details surrounding the implementation and testing of path planning methods, the path itself needs to be discussed. In essence, the traditional outputs from a path planning system is a set of piece-wise connected and a *flyable* path. A path is flyable if it satisfies the vehicle kinematics and remains collision-free given the vehicle's geometry. Seeing as the path calculated by most planning methods consists of a set of waypoints connected by straight lines, post processing is often needed to generate a smooth path. An often used criteria for evaluating paths is their *continuity*. Generally, continuity is divided into *geometric* (G) and *parametric* (C) continuity. Geometric continuity is based on the geometric properties of the curve, thus, the first couple degrees of geometric continuity can be defined as [89]:

- G^0 : All curves are connected
- G^1 : The path-tangential angle is continuous
- G^2 : The center of curvature is continuous

Parametric continuity, on the other hand, takes a more analytic approach:

- C^0 : All curves are connected
- C^1 : The first derivatives (velocities) of the curve are continuous
- C^2 : The second derivatives (accelerations & curvatures) are continuous
- C^n : The nth derivative of the curve is continuous

In chapter 11, the notion of continuity was implicitly stated in the path definition (eq. (11.6)). To achieve feasible paths, different *path smoothing* techniques have been developed. One of the first method to handle this problem was the Dubins path [149] which calculated the shortest path between two points by representing the path as a set of straight lines and circular arcs, given a maximum curvature constraint. These resulting paths satisfies C^1 continuity, but are not curvature-continuous (C^2) due to the discontinuities at the start and end of the circular arcs. If looking at the solution path from a control point of view, the control system responsible for regulating the attitude of the robot will be subject to a discontinuous reference signal due to it relying on second derivatives. This can lead to unwanted behaviour, e.g. due to coupled dynamics in underactuated vehicles, and excessive strain on the actuators. Therefore, curves satisfying at least C^2 continuity is favoured [82], [89].

To bypass the discontinuities present in Dubins paths, attempts have been made to combine them with *clothoids*, which have linearly changing curvatures. While good in theory, clothoids have no analytic solution [89], which can lead to longer computation times. This lack of a closed form structure renders them unsuitable for online planning and navigation. Additionally, this property makes clothoids difficult to use in the presence of obstacles, as it can reduce the algorithmic completeness or optimality of the planning system [150]. A much more promising interpolation method is the use of *splines.* A spline is a piecewise-defined function consisting of a number of polynomials. Their piecewise description allows for combining segments of lower-degree polynomials such that the combined path *appears* smooth [151]. For three-dimensional curves, C^2 continuity is required, but C^3 is often preferred. I.e. the curve needs to have continuous curvature, while continuous torsion is preferable in certain cases. C^3 curves can, for example, be obtained by constructing a *shape-preserving*[7] C^1-G^2 cubic spline with a correction term of variable degree [152].

Other interesting interpolation techniques satisfying the C^2 criteria, while being less computationally heavy, include the *Fermat spiral* (FS) and the aptly named *docking spiral* (DS). Fermat spirals were introduced in 1636 by Pierre de Fermat as a variant of the Archimedean spiral. The original parametrization is based on the square root of the path parameter:

$$\underbrace{r = a\sqrt{\theta}}_{\text{polar}} \quad \underset{\mathrm{T}(x,y)}{\overset{\mathrm{T}(r,\theta)}{\rightleftharpoons}} \quad \underbrace{\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + au\cos(\rho u^2 + \chi_0) \\ y_0 + au\sin(\rho u^2 + \chi_0) \end{bmatrix}}_{\text{cartesian}} , \tag{12.2}$$

where $u = \sqrt{\theta}, u \in [0, \theta_{\max}]$[8], a is a parameter that defines the spiral turning (scaling factor in Cartesian), $\rho = \{\pm 1\}$ is the turning direction and χ_0 is the initial tangent angle [118]. The transformation $\mathbf{T}(\cdot)$ here indicate a transformation between polar and Cartesian coordinates which allows different initial positions $(x_0, y_0)^\top$ and initial tangent angles χ_0 [153].

This alternative parametrization is what guarantees the necessary C^2 continuity, although the tangential angle $\chi(\theta)$ suffers from discontinuities at $\theta = \pi/2$. This is overcome, however, by writing the tangential angle as

$$\chi(\theta) = \theta + \arctan(2\theta) ,$$

which provides a continuous coverage of $\theta \in [0, \pi]$. To then obtain full 2π coverage, the mirrored spiral curve is used. This use of a mirrored curve is what allows the spiral to have segments with zero curvature at the start and end of the path. An example of such a spiral curve is shown in fig. 12.13; here the parametrization in eq. (12.2) has been augmented by including a z component as a function of the number of turns needed and a given maximum slope value.

[7] A shape-preserving curve satisfies certain collinearity, convexity and coplanarity criteria.

[8] $\frac{d\kappa}{d\theta}(\theta) = \frac{\|\dot{r} \times \ddot{r}\|}{\|\dot{r}\|^3} = 0 \;\Rightarrow\; \theta_{\max} = \sqrt{\frac{\sqrt{7}}{2} - \frac{5}{4}}$

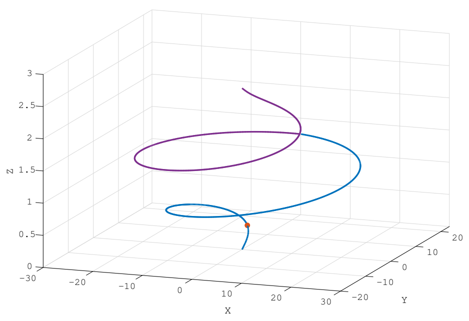


Figure 12.13: Illustration of a simple z-modification to Fermat's spiral that conserve zero curvature wrt. the xy-axes, but not for xz. The blue line is the Fermat spiral and the purple is its mirror. Red indicates the first point of maximum curvature.

The docking spiral is essentially a combination of the Fermat spiral and a logarithmic spiral. Combining these two spiral types allows the curve to reach a designated goal with zero curvature while having an approaching angle that is limited by some $\theta_{\max}$. This makes for safer docking of underwater robots, as the robot nears the docking station straight wrt. the docking station orientation. The main drawbacks of these two spiral methods, however, is that a 3D generalization that keeps their properties have yet to be performed

When discussing path criteria, another property that is useful to discuss — especially in scenarios where obstacles are present — is the path *allowance*. The allowance, $\mathfrak{a}$, is a measure of how much the smoothed path differs from the original connected path segments [89]. In 2D, this is analogous to the magnitude of the cross-track error between the smoothed and the original path. Thus, if the smoothed curve deviates too much from the calculated path, the robot run the risk of colliding with obstacles. In light of this, it is favorable to use a smoothing method that produces curves with small allowances. In general, Fermat's spirals are found to have a smaller allowance compared to clothoids, but larger than that of Dubins paths, hence

$$\mathfrak{a}_{\text{Dubins}} \leq \mathfrak{a}_{\text{Fermat}} < \mathfrak{a}_{\text{Clothoid}} \ .$$

Hence the choice of smoothing method depends on how far the modified path is allowed to stray from the calculated piece-wise linear segments.

More specific path requirements can be deduced from the specific kinematics of the given robot, of course. In the case of a multi-jointed USM, the USM kinematics imposes certain restrictions on the curvature and torsion. These can be defined as follows:

Definition 12.1. Kinematic Path Feasability
Let $\kappa_\pi, \tau_\pi \in \mathbb{R}$ be the curvature and torsion of the planned path, respectively; and $k_\kappa, k_\tau \in \mathbb{R}$ and be the curvature and torsion limits imposed by the USM. If the constraints $\kappa_\pi \leq k_\kappa$ and $\tau_\pi \leq k_\tau$ hold, then the path is considered *kinematically feasible.*

By evaluating a multi-jointed USM consisting of j joints, q_j, with intermediate links of length l_i wrapped around a circle in 2D space with a certain radius R centered at O, the specific curvature criteria can be developed. Then, by defining the angle θ_i as the angle between two subsequent joints, i.e. $\theta_i = \angle q_i O q_{i+1}$ as shown in fig. 12.14, the curvature can be defined as

$$\kappa_i = \frac{2\sin(q_i)}{l_i} \; .$$

From fig. 12.14, it is evident that there is an offset from the previous link that is equal to

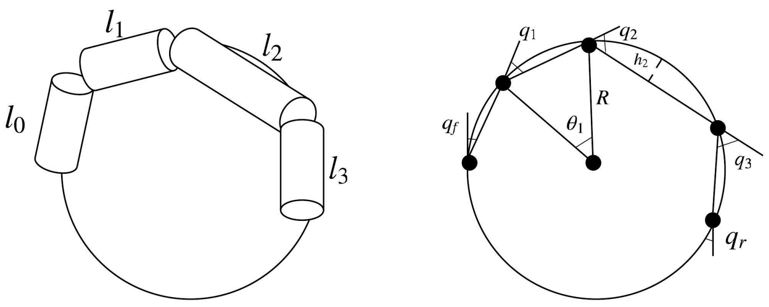


Figure 12.14: To the left, a image of the links of the USM wrapped around a circle. To the right, the joint angles q_i (with front and rear end-effector q_f, q_r) inscribed to visually demonstrate the propagated angular offset from the previous link.

the joint angle of that link. Thus, given a USM with a maximum joint angle $q_{\text{i,max}}$, the path curvature κ_π must satisfy the following constraints:

$$\begin{aligned} q_1 &= \arcsin\left(\frac{l_1}{2\kappa}\right) + q_f \;, \; \frac{\theta_1}{2} \le q_{1,\max} - q_f \\ q_2 &= \arcsin\left(\frac{l_2}{2\kappa}\right) + \frac{\theta_1}{2} \;, \; \frac{\theta_2}{2} \le q_{2,\max} - \frac{\theta_1}{2} \\ &\cdot \\ &\cdot \\ q_n &= \arcsin\left(\frac{l_n}{2\kappa}\right) + \frac{\theta_{n-1}}{2} \;, \; \frac{\theta_n}{2} \le q_{n,\max} - \frac{\theta_{n-1}}{2} \; . \end{aligned} \tag{12.3}$$

If any of these limits fails to be satisfied, it is kinematically impossible for the USM to follow the circular path exactly. The proof of this is shown in appendix C.

As a consequence of this, the calculated path is required to be at least C^2, with a maximum curvature κ_π satisfying the USM kinematics. How this is accomplished will be further discussed in chapter 14.

12.7 Context-aware Planning

Most of modern path planning methods tend to focus specifically on generating a collision-free path, lumping all perceived obstacles in C_{obst}. By doing this, any information regarding the specific obstacle is completely neglected. In many situations it can instead be beneficial to extract additional information regarding the specific obstacles. Consequently, the robot can adapt to the environment at hand by performing actions based on the type of obstacle encountered. Recent literature presents different means of performing context-segmentation — as well as various ways of utilising it. One such method, based on a mobile robot in the 2D plane, is presented in [154]. The authors' main motivation behind this work was focused on improving reasoning in service robots as well as the *social aspect of navigation*. This essentially means improving the robot's decision-making abilities in human-made environments, as well as having the robot operate more orderly in areas where humans are present. These core concepts can easily be transferred to submarine robots, with less focus on the social aspects, however.

In submarine environments it can be valuable to distinguish between natural and human-made structures. Human-made constructions might be of interest for inspection or following, or their dimensions can be extrapolated from incomplete sensor data in an exploration setting. These extrapolated obstacles can then be used to plan paths that avoids the estimated obstacles which can make for safer and more efficient planning, as expensive re-planning can possibly be avoided. Natural obstacles, on the other hand, might include submarine vegetation which can interfere with actuators and cause damage, and might therefore need additional clearing.

Accomplishing this type of context-aware planning is the main idea behind linking together classification and path planning. This can be achieved object detection through image or point cloud classification or segmentation. In the case of a USM performing IMR operations, this context-aware planning can be used to help guide the robot along a pipe to perform an inspection task, for example. This is especially useful in cases where the pipe is covered by natural substances, or in the rare case where the pipe has drifted. By employing an exploration driven inspection strategy, assisted by semantically labelled environment information, these sort of problematic occurrences can be overcome. See section 16.1 for a more detailed walk-through of the implementation specifics of connecting the planning and classification modules — exemplified with a pipe inspection case — and ?? for results from simulation tests.

In this thesis, the classification is performed based on images. Methods for classifying and segmenting unstructured three-dimensional point cloud data exists, but are not thoroughly investigated in this project. If interested in this subject, the reader is referred to appendix D for a brief introduction to the concepts and surrounding literature.

12.8 Summary

Path planning, and the overlying theme of task planning in general, is a central subject in robotics going back to the dawn of modern mobile robotics. In turn, many different approaches of handling this problem. Some of these methods have been tried exemplified here.

Combinatorial planning attempt to solve this problem by generating an, often, sparse graph which efficiently store information on the environment. Efficient search algorithms can then be applied to the generated graph, e.g. using A*, for efficient planning. These methods tend to suffer from a few drawbacks such as complex graph generation procedures and the possible difficulty of handling the information of a continuously changing environment.

Probabilistic, or sampling-based, planning have in recent years been the go-to approach for online path planning in a variety of applications. This is in large part due to their relative simplicity and efficiency. Examples from the RRT family have been put forward to cover most of the apparent planning problems in robotics, with their applicability to so many different scenarios being a result of their straightforward implementation and the relative ease of which vehicle kinematics can be included. They are not without their problems, however, as they are probabilistic in nature and lose some of their optimality in exchange for performance.

Other approaches have also been briefly discussed, such as artificial potential fields and optimization methods. With the advent of artificial intelligence and deep learning, it is only logical to attempt to use this to solve planning tasks. So far, not much research have been done directly on this, but very promising examples such as MPNet [106] exists. By basing the planning on AI/DL, it is not unreasonable to imagine a network combining the task of planning with additional objectives, such as possibly incorporating semantic information directly.

Seeing as most planning algorithms result in a set of connected waypoints, post processing is most often needed to ensure the feasibility of the path wrt. the guidance and control systems. A set of path critera, including continuity and allowance, was therefore discussed, and a couple of techniques for generating paths of a sufficient degree of continuity introduced. Based on the evaluated literature, Fermat's spiral seem a promising method if generalized to three dimensions while preserving the properties of the 2D parametrization. The same argument goes for methods such as the briefly touched upon docking spiral, which have the additional nice property of providing straight curves towards the goal. Seeing as these methods are only developed for 2D motion, the well established method of cubic spline interpolation, and its variants, were studied in stead. An approach based on natural cubic splines was used in the actual implementation (chapter 14), partly also due to the way the control system was implemented.

Finally, the concept of introducing contextual information to the planning procedure was briefly investigated. The idea behind the combination of the classification system from part IV and the planning module was presented, which will be further illustrated in chapter 16.

The question of how to dynamically set new goals remains, however. To shed some light on this problem, the next chapter will discuss different approaches for autonomous generation of goal states for continuous and exhaustive environment exploration.

13 | Autonomous Exploration

Modern robotics is in many ways driven by the goal of attaining robust, autonomous systems. Key tasks that are essential in many such systems are exploration, localization and mapping. Accomplishing this allows the robot to be able to estimate its pose while simultaneously seeking to exhaustively explore and keep track of the environment. This process is widely known as *simultaneous localization and mapping*, or SLAM for short [10] (see chapter 7). Traditionally, this entails teleoperating the robot to build a map and run localization, or even building the map first then attempting localization separately. Performing such tasks this way introduces the need for a human operator which is not ideal, especially for longer missions where communication delays might become significant — planetary exploration is a case in point [7], [8]. Attempting to run these processes autonomously, however, establishes a need for the system to, independently, decide where to go next based on the current sensory information and map estimates. This problem of deciding the best exploration goals, and in turn planning paths to reach them while, is often referred to as *Active SLAM* [155].

There are many ways of deciding such exploration objectives, one such being focusing on *loop closures*. Loop closing connects previously explored parts of the environment with newly discovered ones and updates the map estimate with the combined information. This is an important procedure in SLAM as it helps obtaining consistent environment maps and reduces the uncertainty of the mapping, as it allows for the correction of any inevitable drift that has occurred [10]. Thus, one exploration strategy could be to exhaustively explore the environment while aiming to minimize the map uncertainty. Deciding a satisfactory exploration goal then becomes the main problem.

This chapter will discuss different exploration strategies that has been put forth in previous literature, as well as approaches proposed in recent theoretical and practical research. Different strategies have been developed, each based different objectives and methodologies, but they are often broadly categorized as either *frontier-based* or *next-best-view* (NBV) planners. This is the overarching categorization that will be used in this chapter. Each of these schools have their advantages and disadvantages, both of which will be examined.

13.1 Exploration Strategies

Path planning in combination with environment exploration entails having only partial knowledge of the complete surroundings. Therefore, the system needs to be able to characterize the unknown areas of the map and from that information calculate a suitable exploration goal state. Early exploration methods were relatively primitive, such as those based on wall-following [156], which continuously follows obstacle edges to complete the map — similar in essence to how the BUG-family of planning algorithms work [157]. This approach has some obvious difficulties arising when applied to larger and more open areas, which can lead to large areas of the map being unexplored. Improved methods were quickly developed, with the simplest being *nearest frontier*, which in turn has been extended with *cost-utility* functions and *behaviour-based* and *hybrid* approaches.

13.1.1 Frontier-based Exploration

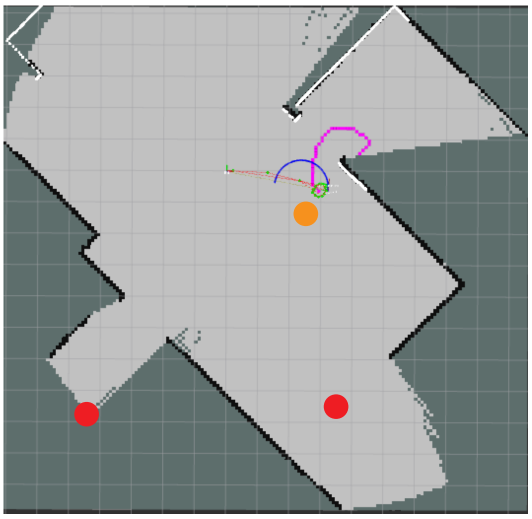


Figure 13.1: Example from a 2D SLAM simulation using an occupancy grid representation with example frontiers marked in red and the calculated path (pink) from the current robot position to the closest frontier (orange).

In 1997, Yamauchi presented the quintessential exploration strategy, namely the nearest frontier method [158]. This method works by dividing the map into three distinct categories: *explored space*, *unexplored space* and *frontier space*. This is similar to the occupancy grid representation, where the frontiers are characterized as the boundary regions between the explored and unexplored space. Based on the frontier regions, centroids of sufficiently large frontier clusters are calculated, before the centroid closest to the robot's current position is chosen as the next exploration goal. The practical execution of this method is illustrated in fig. 13.1, where the nearest frontier exploration approach is applied on a 2D exploration simulation using ROS and a simulated differential-drive turtlebot [159].

This strategy has the obvious advantage of being simple and its possibilities for modifications and improvements. Such examples include the use of topological vision-based methods [160] and feature-based methods [161] as search strategies. Although the method itself is pretty straight forward, it also allows for exploration in narrow spaces [158]. The simplicity of this method is not only positive, however.

Seeing as the nearest frontier method always chooses the closest frontier, it ignores any extra information that might help decide better goals. In some cases it might also be advantageous to have a notion of *safeness* regarding the path towards the possible exploration goal, partly to help lessen the effects of any environment and vehicle modelling errors. In an attempt to circumvent this, an exploration method was developed which combines the closest frontier approach [158] and a path cost evaluation based on the *path transform* of [162]. This method, presented in [163] and dubbed the *exploration transform*, bases the exploration on minimising both the path length and the path risk. This is formulated as

$$\Psi(x) = \min_{x_g \in \mathcal{F}} \left\{ \min_{\pi \in \Pi_x^{x_g}} \left\{ l(\pi) + \alpha \sum_{x_i \in \pi} C_{\text{danger}}(x_i) \right\} \right\} , \tag{13.1}$$

where $\mathcal{F}$ is the set of all frontier cells, $\Pi_x^{x_g}$ is the set of all paths from x to x_g, $l(\pi)$ is the length of the path π, $c_{\text{danger}}(x_i)$ is the cost function for the risk of entering cell x_i and $\alpha \geq 0$ is a weighting factor. C_{danger} was originally stated in [162], but this formulation essentially enforces a repulsive force on the robot from the obstacles no matter the distance between them. To bypass this, the cost function was reformulated as

$$C_{\text{danger}} = \begin{cases} \infty, & \text{if } d < d_{\min} \\ (d_{\text{opt}} - d)^2, & \text{else} \end{cases} , \tag{13.2}$$

where $d_{\min}$ represents the closest allowable distance between the robot and obstacles and d_{opt} is an estimated optimal, or encouraged, clearing [163]. The main advantages of this approach are that the path safety is taken into account and that it finds the most appropriate path without the risk of running into local minima. In practical implementations, this has shown good results for 2D mobile robots. If applied to a three-dimensional problem this might not be the case due to the potentially large increase in evaluations needed. This method can be considered a version of a cost-utility method, as it combines the frontier search with cell and path costs. Another cost-utility based way of choosing the next waypoint would be to estimate the utility, or information gain, of a candidate point.

Information gain, in this sense, represents the amount of information about the environment gained by reaching a certain point in space. The expected information gain can be defined in terms of *entropy* [164]. The entropy can be approximated as the joint entropy of the path $\pi(\varpi) = \mathbf{x}_k = x_{1:k}$ and the map $\mathcal{W}_m$, given a series of control inputs $\mathbf{U}_k = u_{0:k-1} \in \mathcal{U}$ and a set of observations $\mathcal{Z}_k = z_{1:k}$. The total entropy then becomes [165]

$$H(\mathbf{x}_k, \mathcal{W}_m \mid \mathbf{U}_k, \mathcal{Z}_k) \approx H(\mathbf{x}_k \mid \mathbf{U}_k, \mathcal{Z}_k) + H(\mathcal{W}_m \mid \mathbf{U}_k, \mathcal{Z}_k) , \tag{13.3}$$

where the individual entropies are defined as in [165]

$$H(\mathbf{x}_k \mid \mathbf{U}_k, \mathcal{Z}_k) \approx \frac{1}{k} \sum_{i=1}^{k} \ln \left(2\pi \exp \left[\frac{n'}{2} \right] \right) |\Sigma_{ii}| \tag{13.4}$$

$$H(\mathcal{W}_m \mid \mathbf{U}_k, \mathcal{Z}_k) = -w^2 \sum_{c \in \mathcal{W}_m} \{ p(c) \ln p(c) + [1 - p(c)] \ln[1 - p(c)] \} \ , \tag{13.5}$$

where n is the size of the state vector and Σ represents the covariance matrix. This formulation is based on pose-SLAM, i.e. $\mathcal{W}_\mathrm{m}$ is represented as a pose graph (see ??). To find the input, or action, that maximises the expected information gain one can instead find the input that minimises the joint posterior entropy $H(\mathbf{x}', \mathcal{W}_\mathrm{m} \mid \mathbf{U}_k + \mathbf{U}', \mathcal{Z}_k + \mathcal{Z}')$ [165]:

$$\mathbf{U}'* = \operatorname{argmin} \{ H(\mathbf{x}', \mathcal{W}_\mathrm{m} \mid \mathbf{U}_k + \mathbf{U}', \mathcal{Z}_k + \mathcal{Z}') \} \ . \tag{13.6}$$

Applicable actions **U** include exploration to specific waypoints, but can also combine actions where the robot tries to perform loop closure.

The advantage of such an exploration method is that the resulting algorithm gives good results regarding loop closure and frontier exploration, since actions can be chosen to either reduce pose uncertainties by exploring the environment or to reduce path uncertainties by looping back to known locations. This expected information gain can further be combined with an evaluation of the path length to then be able to weigh the information gain against travel cost. Related to this, focusing on minimising landmark uncertainty, is the method presented in [166]. This method aims to plan towards minimising the uncertainty of landmarks and robot location in the map by weighting the two uncertainties. I.e., if the robot location has a high uncertainty, the next waypoint should be chosen based on nearby landmarks with low uncertainty, and vice versa.

The majority of exploration strategies in modern literature have, similarly to path planning techniques, been focused on two-dimensional problems. Only fairly recently have research targeted the extension of frontier exploration to three dimensions. These approaches often convert 3D point cloud data to an octree, from which frontiers are extracted [167], [168]. While being based on octrees, this shows the promise of real-time frontier extraction in 3D exploration due to the efficient processing and look-up provided by, for example, OctoMap [121]. The main problem of just extending the same methods to 3D, is the fact that the number of frontiers and their possible scattering renders these methods inefficient. To help reduce the problem of increased dimensionality, efficient clustering algorithms can be used to select frontier candidates. One example of this is through the use of k-means divisive clustering [169]. This method runs divisive — also known as hierarchical clustering — within a specific sensor FOV to cluster the frontiers in chunks viewable by the sensor. This, and the approach presented in [167], where new frontiers are added to the stored set only if they contain new information, show that simple extensions of traditional approaches — by means of efficient point cloud clustering and frontier caching — results in improved exploration strategies.

After a new waypoint has been set, for example using the nearest frontier approach, there is the possibility of the robot having sufficiently explored the region associated with the surrounding frontier. To account for this, the method of *repetitive re-checking* has been proposed [170]. If the chosen frontier at any point during execution ceases to be a frontier cell, it is dropped and a new waypoint is chosen. Results show that this addition does not increase the computational burden given the constant-time look-up to determine if it is still a valid frontier. This simple addition can, however, help decrease the over-all path length and exploration duration by decreasing the time spent in already mapped terrain. Another problem that can occur, if the explorations goals are not chosen optimally, is that semi-closed parts of the map can be visited multiple times. In 2D, the analogy to this is that a room the robot explores is left before being fully explored, due to a goal being set in the adjacent room. The suggested solution is then to segment the environment based on the Voronoi diagram. Frontier cells can then be associated with the nearest Voronoi region. Frontier cells lying in another Voronoi region is then only chosen if the set of frontiers in the current Voronoi region is empty. Thus, the robot will not leave its current enclosed region until it is fully explored. In the case of exploring sparse underwater environments, this is unlikely to be as big of a problem. If the robot is to be utilised in underwater caverns or similar, however, this might need to be taken into consideration. Nevertheless, this extension is very simple and can decrease the overall path length, and since it is based on Voronoi diagrams, should be relatively easy to extend to 3D if using a Voronoi-based planner.

13.1.2 Next-Best-View Exploration

Another strategy that is closely related to the frontier-based approach, but separated for clarity, is *next-best-view* (NBV) exploration. NBV methods seek to find the minimal set of sensor poses that covers a given scene. To appropriate this for online exploration, receding horizon NBV methods (RH-NBV) have been developed [171]. RH-NBV creates a set of voxels from the occupancy grid that are visible from the current configuration, but remain unmapped. These voxels are also chosen such that the direct line-of-sight does not intersect with C_{obst}. Given the initial configuration, RH-NBV then builds a tree using RRT. Hence, RH-NBV combines planning and exploration by exploiting the RRT's Voronoi bias, favouring large Voronoi regions.

Each node in the tree is given a score based on the score of the parent node, $s(\mathbf{x}_p)$ the current information gain, $g(\mathbf{x})$, and their cost of traversing to that configuration, $c(\mathbf{x})$. This can be expressed as $s(\mathbf{x}) = s(\mathbf{x}_p) + g(\mathbf{x}) \exp[-\lambda c(\mathbf{x})]$, where λ is a tuning parameter penalizing long paths — i.e. a large λ makes the robot explore its close surroundings carefully before moving on [171]. At each iteration, or replanning, the best path segment is chosen and executed, before the remaining tree is used to initialize the next search. This method allows for efficient inclusion of sensor dynamics which can result in good map coverage, even in 3D environments.

This method have two main drawbacks, however. The first is that it, as presented in literature, relies on RRT. This means that the solution path is not guaranteed to be near the optimal path, and is likely to stray further from the path as the exploration goals are chosen further away from the current configuration. The second problem has to do with map scale. If the map is very large, it is likely that exploration goals will stray further and further away from the robot as time goes on. Due to the tuning factor λ, goals far away from the current position may never be chosen if the score becomes low enough. This is because of how the method terminates the exploration task, stopping if the best score becomes lower than a specified threshold, which can result in premature termination.

The traditional frontier-based methods is not without problems either, with the biggest drawback being the time it takes to give full coverage. Very recently, however, there was presented a method that combines frontier-based exploration with RH-NBV exploration, named the *Autonomous Exploration Planner* (AEP) [172]. This approach uses RH-NBV as a local exploration strategy, while frontier exploration is used globally, ensuring more efficient handling of large-scale environments. Through experiments, this has been shown to outperform both frontier-based and RH-NBV exploration strategies. Being developed mainly with UAVs/MAVs in mind, the resulting paths are not directly feasible for UUVs. This could potentially be improved upon with modifications, however.

13.2 Summary

All the aforementioned exploration strategies are tried and tested in terrestrial or aerial surroundings, where sensor information is much more reliable, and usually contains more depth information. By virtue of noisier data and environmental disturbances — among other complications — underwater exploration and mapping have been shown to be much more challenging than its above-ground counterpart. Methods that have shown promising results has been developed [173]. However, these are performing view-based exploration based on both sonar and cameras. In this thesis, only visual sensors are available for gathering environment information. In future research, however, it could be interesting to combine sonar data with the camera(s) to give a larger spatial horizon in which the longer-ranging sonar could be used to estimate potential camera views for increased map coverage — with the added benefit of better collision avoidance.

Seeing as many underwater robotic vehicles are fairly restricted in roll, pitch and also yaw angles, view-based exploration strategies were deemed unsuitable, unless fundamentally changed in the tree generation. This is fully doable, as the inclusion of vehicle dynamics is supported by the RRT algorithm. However, the problem of preemptive termination, as discussed briefly in section 13.1.2, poses a larger problem as the potential workspace can grow quite large in underwater environments. This can, as mentioned, be circumvented using the method proposed in [172], combining RH-NBV and more traditional frontier-based exploration. This could be a very interesting extension to look into in further work.

Autonomous exploration share the same main drawback as many path planning methods, namely that the problem complexity increases immensely as the problem dimension expands. Due to this, frontier exploration in 3D environments suffer from the sheer amount of possible frontier cells present in the map. This results in traditional nearest frontier methods [158] being rather inefficient without efficient frontier detection [169] and storage in between measurements. For this thesis, an exploration strategy that combines traditional frontier-based methods with an altered frontier evaluation function and efficient point cloud clustering was implemented. This is further discussed in section 14.4.

14 | Planning & Exploration Method

Different combinatorial and sampling-based planners, along with their applicability to dynamic underwater mapping and exploration, have in previous chapters — as well as in the work performed in [1] — been investigated. As previously mentioned, combinatorial methods have the advantage of being algorithmically complete and, in most cases, optimal with regard to the stated optimality conditions. Due to this, combinatorial methods are often preferable if the method is applicable to the problem at hand. On the other side — disregarding fuzzy methods and algorithms based on numerical optimization [7], [106], [174]–[176] due to computational costs — sampling-based methods have proven very reliable for online robot path planning [81], [130], [136], [137], [177]. To examine the applicability of combinatorial and sampling-based methods for underwater mapping, inspection, and exploration, both a Voronoi-based planner and a planning system based on an RRT-variant and informed state sampling were implemented.

This chapter will delve more deeply into the implementations referred to in chapter 13 chapters 12 and 13. The next couple of sections discuss the implementation of a maximum clearance path planner based on Voronoi diagrams and an asymptotically optimal informed sampling-based planner, and discusses their suitability for use online in an autonomous mapping and exploration operation.

14.1 Incremental Voronoi-based Path Planning

Previous research has indicated that the Voronoi diagram is a very useful concept in path planning, especially when robot safety with respect to static obstacles is essential. Several studies have shown its use in the 2D plane, both for terrestrial mobile robots [116] and for marine surface vehicles [178]. These concepts have been tried applied in three-dimensional environments in different ways, either through the calculations of signed distance fields [120] or by bounding box-approximations of the obstacles present in the environment [123], to name a couple. Both of these, however, include a number of assumptions or simplifications which makes them infeasible for the goal of this work. In the work presented by Oleynikova [120], for example, the signed distance field calculations are performed fairly efficiently, but perform the graph generation offline. This can be a good solution in many cases, e.g. in the case where the robot performs much work at a small area of operation before moving on, giving it time to perform back-end graph calculations — which is not necessarily the case for the problem at hand. Several previously published studies work under the assumption that all static obstacle positions and shapes are known, and in some cases also assume that the total workspace boundaries are known — which is the case with the work presented in [123], among others. To try to alleviate the necessities of these kind of assumptions, the work presented here tries to perform the obstacle discretization and Voronoi diagram generation iteratively, to allow for continuous environment expansion through exploration and mapping. The first steps in trying to accomplish this, was to achieve a simplified obstacle shape estimation through the use of clustering techniques. These simplified shapes can then be used as generator points for the Voronoi diagram.

14.1.1 3D Point Cloud Clustering

As touched upon in section 12.5, numerous clustering methods have been developed in the last decades, each with concrete advantages and disadvantages. Choosing the right clustering method is very dependent on the properties of the data at hand. Due to this, a few different clustering methods were applied on environments simulated through Gazebo [179] and the use of a UUV simulator [20] as mentioned in section 14.5. The simulated environment consisted of a fairly uneven underwater submarine terrain, that to some degree mimics an actual underwater environment. Due to the nature of the simulated point cloud data, clustering methods assuming a static number of clusters, as well as algorithms based on shape assumptions, were discarded. This includes algorithms such as k-means due to the underlying assumption of spherical objects, even though there exist modifications that estimate the number of clusters [180]. The methods most closely evaluated and tested were Euclidean distance clustering — or density-based clustering (such as DBSCAN [125]) — and the *region growing* segmentation technique. Clusters in this context can be defined as follows.

Definition 14.1. Let δ define a specific similarity measure. Furthermore, let $\mathcal{P}$ be a given point cloud model containing points $\mathbf{p}$ defining a number of obstacles O — or classes — where a single cluster $O_k \subseteq O$ contains a single obstacle. I.e. a single cluster can be defined as

$$O_k := \{\mathbf{p}_k \in \mathcal{P}\} \;\; \forall \;\; \mathbf{p} \;\; \text{s.t.} \;\; \delta \; ,$$

where δ is dependent on the method and similarity measure used.

Density-based clustering

Clustering points based on density is a fairly straight-forward approach where the similarity measure is based on a distance measure between points, with the Euclidean distance (L_2-norm) being used most often. The most used spatial distance-based clustering algorithm — and probably the most used clustering algorithm overall — is the *density-based spatial clustering of applications with noise* (DBSCAN). Two slightly different implementations were tested, where the first one was based on the *Point Cloud Library* (PCL) and computes the closeness of points using the L_2-norm using the closeness definition stated as

$$\delta : \min \|\mathbf{p}_i - \mathbf{p}_j\|_2 \geq d_{\text{th}} \ , \tag{14.1}$$

where d_{th} denotes a maximum distance threshold [181] with $\mathcal{P}$ stored as a k-d tree. Based on this, all points around the query point $\mathbf{p}_k$ is said to belong to cluster O_k if they fall within the sphere with radius $r = d_{\text{th}}$ centered at $\mathbf{p}_k$. Based on this, the clustering works by going through the points in $\mathcal{P}$ and, if the neighboring points fall within the threshold radius, adding the neighbours to the same cluster. The simplified pseudocode is shown in algorithm 4.

As with most clustering methods, slight tuning was needed to fit the simulated data. Point cloud data was fed forward from the occupancy grid generated from the simulated environment (fig. 12.11); the resulting clusters are shown in fig. 14.1.

Algorithm 4 DensityClustering ($\mathcal{P}$, d_{th}, M)

```
    Input: Point cloud $\mathcal{P}$, distance threshold $d_{\text{th}}$, minimum cluster size $M$
    Output: List of clusters $C$
 1: $\mathcal{T}$ = PointCloudToBinaryTree ($\mathcal{P}$)    // Convert $\mathcal{P}$ for efficient neighbor look-up
 2: C = ∅                                                   // List of clusters/obstacles
 3: Q = ∅                                                   // Queue containing query points
 4: for $\mathbf{p}_k \in \mathcal{P}$ do
 5:     Q = Q ⋃ $\mathbf{p}_k$
 6:     for $\mathbf{p} \in$ Q do
 7:         N = FindNeighbors ($\mathbf{p}_k$, $d_{\text{th}}$)
 8:         if |N| < M then
 9:             continue
10:         for each neighbor $\mathbf{q} \in$ N do
11:             if not PointHasBeenProcessed ($\mathbf{q}$) then
12:                 Q = Q ⋃ $\mathbf{q}$
13:     C = C ⋃ Q                                           // All points in queue processed, add to C
14:     Q = ∅
    return C
```

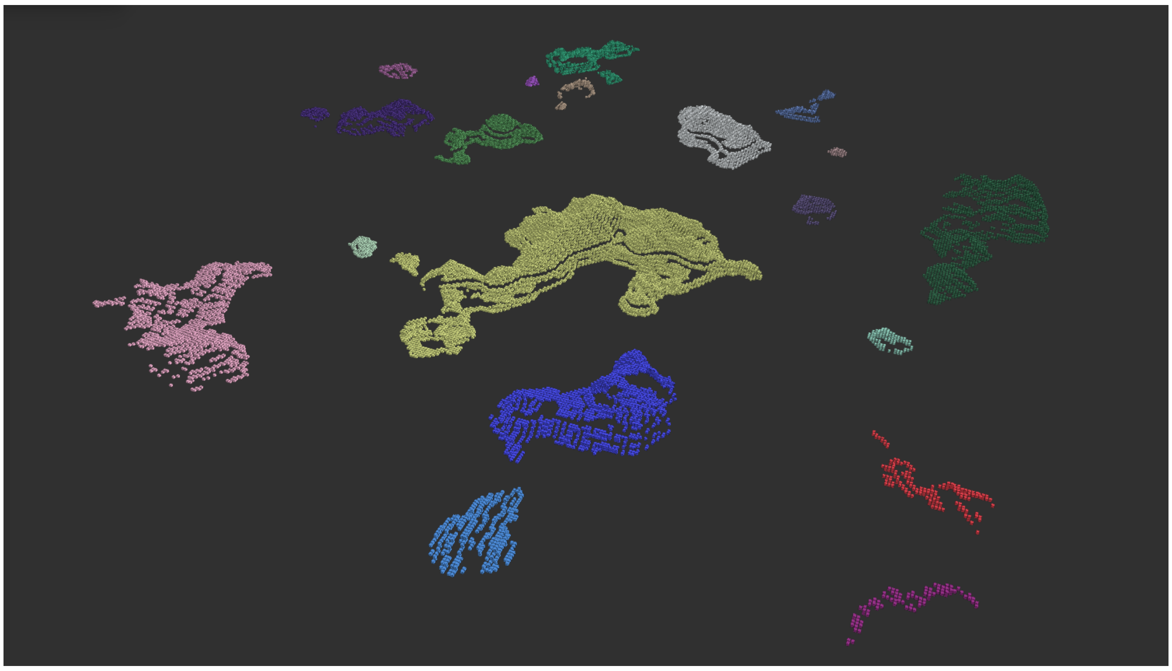


Figure 14.1: Resulting obstacle clusters after running density-based clustering on the data shown in fig. 12.11.

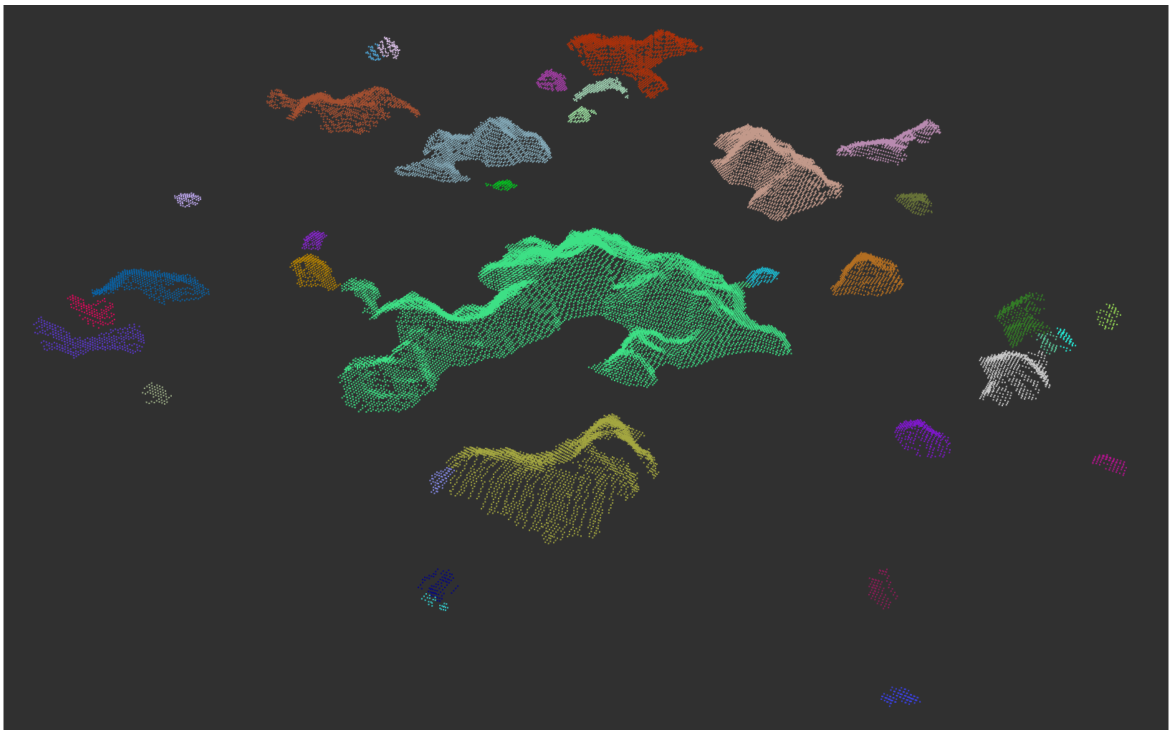


Figure 14.2: Resulting obstacle clusters after running density-based clustering on the data shown in fig. 12.11 using the SciKit Python implementation with a different space partitioning method.

The other density-based algorithm that was tested was Scikit-learn's DBSCAN implementation [182] using a *balltree*[1] [183] for space partitioning. Through multiple tests, all with $\delta = L_2$-norm and the same minimum cluster size and density, it became evident that the Scikit variant using balltrees was more sensitive to small clusters, whereas the PCL implementation with k-d trees was more prone to write off smaller point collections as noise. This slight difference can be seen in figs. 14.1 and 14.2.

Region-growing

Clustering by region-growing can be seen as a slight extension to pure distance-based clustering. This method assumes that objects are differentiable based on other features than just spatial closeness. These features might include surface smoothness, texture or color. Color was deemed insufficient for distinguishing differences under water; object smoothness was instead chosen as a possible similarity measure.

This implementation is present in PCL also, with the similarity chosen as the angle between the surface normals of two points $\mathbf{p}_i$ and $\mathbf{p}_j$:

$$\delta \, : \, \arccos\left(\langle \mathbf{n}_i, \mathbf{n}_j \rangle\right) \leq \theta_{\text{th}} \ , \tag{14.2}$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product, $\mathbf{n}_k$ is the surface normal at the point $\mathbf{p}_k$, and θ_{th} defines the maximum angle threshold [181]. By letting $\mathbf{p}_k$ indicate the current seed point, if the angle between the normal vectors of $\mathbf{p}_k$ and the current query point $\mathbf{p}_j$ is less than θ_{th}, then $\mathbf{p}_j$ can be included in O_k. This requires the estimation of the surface normal vectors of the points $\mathbf{p} \in \mathcal{P}$. This is done through calculation of the normal vector of the tangent plane S fitted to the point $\mathbf{p}_k$ and its neighbors through least-squares optimization [184]. I.e. fitting a plane $S = n_x x + n_y y + n_z z + d$ to the neighborhood $\mathcal{N}$ (containing m neighbours as well as $\mathbf{p}$ itself) by solving

$$\min_{\mathbf{b}} \|[\mathcal{N} \ \mathbf{1}_{m+1}] \ \mathbf{b}\|_2 \ , \tag{14.3}$$

where $\mathbf{b} = \left[\mathbf{n}^\top, \, d\right]$. See [181], [184]–[186] for a more in-depth coverage of surface normal estimation.

Additionally, as discussed in [181], the surface curvatures can be analyzed to guide the region growth in directions of lower curvature before evaluating higher curvature directions. The curvature can be estimated directly using the neighborhood around the point $\mathbf{p}_k$, and a check similar to eq. (14.3) can be performed using a curvature threshold κ_{th}. For a closer look at the implementation specifics, the reader is referred to [181].

[1]A geometric data structure similar to a k-d tree, but based on hyper-spheres as opposed to cubes, resulting in a Voronoi-like partition of the data.

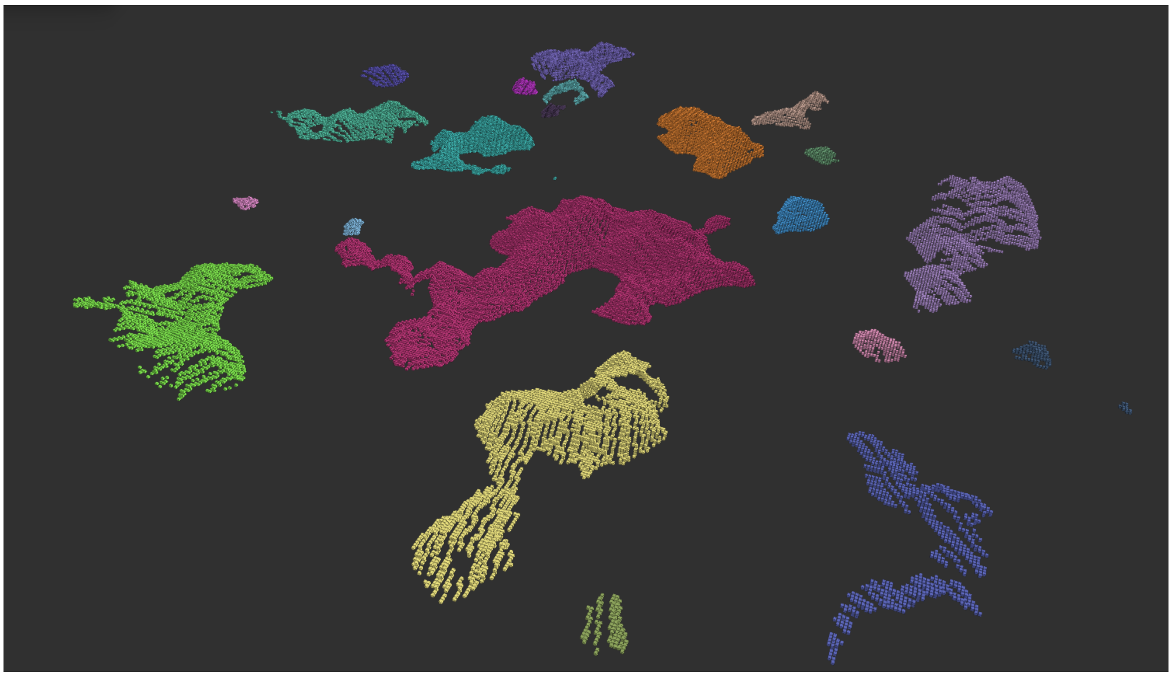


Figure 14.3: Point cloud clustering using the principles of region-growth applied to the simulated terrain shown in fig. 12.11.

An example clustering of the environment shown in fig. 12.11 using region-growing is presented in fig. 14.3. Region-growing has been shown to, in some cases, achieve a higher accuracy when used for object segmentation than density-based clustering [187]. These differences are by no means extreme, and no major quantifiable differences were apparent in simulated tests. Through simulations, however, visual inspection indicated that region-growing was a suitable method and was therefore favored over pure Euclidean clustering for simulation tests. This needs re-evaluation, and re-tuning of parameters, in practical tests before deployment.

As mentioned, these algorithms were present in PCL and integrated using their open source framework. After extracting the clusters, they were fed sequentially through a convex hull algorithm.

14.1.2 Abstracting Obstacle Shapes Using Convex Hulls

Convex hull algorithms work by encapsulating a set of input points using a convex polygonal shape. A convex hull $\mathbb{C}$ of n points $p \in \mathcal{P}$ is the smallest convex set containing all points, and can be defined as [117]

$$\mathbb{C} := \left\{ \sum_{i=1}^{n} \lambda p_i \, : \, \lambda \geq 0 \, \forall \, i \, , \, \sum_{i=0}^{n} \lambda = 1 \right\} \, .$$

There exist numerous convex hull algorithms, but two algorithms was deemed more fitting than others for the task at hand: *i)* Chan's algorithm [188] and *ii)* the Quickhull algorithm [119].

Chan's algorithm

Chan's algorithm is an example of an optimal *output-sensitive* convex hull algorithm based on the divide-and-conquer concept [188]. Output-sensitivity refers to the algorithm's runtime being dependent on the output. This algorithm has a runtime of $O(n \log h)$, where h is the number of vertices in the output hull. Chan's algorithm was developed for 2D and 3D points and remains the theoretically optimal algorithm in these cases, with the original author's implementation apparent in its first publication [188].

An attempt was made to test this algorithm on the simulated test-case. While being a relatively simple algorithm, conceptually, much work would have to be done to create a robust implementation to handle degeneracies and the like. This was concluded outside of the scope of this thesis. Due to this, the more robust implementation of the Quickhull algorithm available through the Qhull library [119] was chosen.

Quickhull

The original Quickhull algorithm is conceptually similar to the Quicksort algorithm in the way it utilizes the divide-and-conquer strategy. In the two-dimensional case, the algorithm starts by finding the minimum and maximum x values p_a, p_b, and then uses the line between them to partition the point set in two subsets. Points encapsulated by the triangle can then safely be ignored in further processing. The algorithm then recursively checks lines generated between the defining points of the new subset (e.g. using the line connecting p' and p_b) until all points are processed.

The extension of this algorithm to higher-dimensional problems, presented by Barber et. al., is based on the 2D version together with the inclusion of the *Beneath-Beyond* algorithm [189]. This facilitates the generation of convex hulls in d dimensions with a theoretical runtime of $O(n \log n)$ [119]. The pseudocode shown in algorithm 5 is a slightly simplified variant of the implementation presented in the Quickhull paper, and is based on the three-dimensional case.

The Quickhull algorithm is readily available through the Qhull library [119]. While providing robust and efficient implementations of several convex hull-related algorithms, its C++ interface is still under heavy development. Because of this, the interface available through the *Computational Geometry Algorithms Library* (CGAL) [190] was used instead. An example of a resulting convex hull is shown in fig. 12.9. Even though this algorithm has a slower runtime than Chan's algorithm in the average case, the Quickhull algorithm was used due to its robust implementations and it being well established in the field of computational geometry.

Algorithm 5 Quickhull ($\mathcal{P}$)

```
    Input:  Point cluster $\mathcal{P}$
    Output:  Convex hull $\mathbb{C}$
  $\mathbb{C} = \varnothing$
  $\mathbb{C}$ = GetMaximalTetrahedron ($\mathcal{P}$)          // Based on max $x, y, z$ values
  for $f \in \mathbb{C}$.facets do
      for each unassigned point $p \in \mathcal{P}$ do
          if $p$ outside $f$ then $f$.AddToOutsideSet ($\tilde{p}$)
  for $f \in \mathbb{C}$ with $f$.OutsideSet $\neq \varnothing$ do
      $p'$ = GetFurthestPoint ($f$.OutsideSet)
      V = $f$                                    // $V$ is the current visible set
      N = GetVisibleNeighbors ($p'$, V.facets)
      V = V $\bigcup$ N
      H = GetBoundary(V)                         // Get horizon ridges
      for ridge $r \in H$ do
          $f'$ = GenerateFacet ($r, p'$)
          ConnectNewFacet ($\mathbb{C}$, $f'$)           // Link this facet to its neighbors
      for each new facet $f'$ do
          for each unassigned point $q \in f$.OutsideSet $\in V$ do
              if $q$ outside $f'$ then
                  $f'$.AddToOutsideSet ($q$)
      V = $\varnothing$
  return $\mathbb{C}$
```

14.1.3 Generalized Voronoi Generation From Convex Hulls

Having calculated a set of bounding polygons, the scene is set for generating the Voronoi diagram. The convex hull vertices are passed as generator points, with the diagram generation itself being based on the Qhull library. The generator points was passed to the SciPy wrapper of Qhull's Delaunay-based Voronoi (*qvoronoi*) which allows for quick diagram generation. An illustration of the process from which the diagram is calculated — which is based on convex hulls and the lifting transform [191] — is shown in fig. 14.4.

As previously exemplified — see fig. 12.10 — the generated diagram can include edges that lies within C_{obst}. Therefore, the generalized Voronoi variant is required, in which the infeasible edges are pruned. To accomplish this, the edges were checked, and if found to be within a convex hull, marked for removal by storing their indices. To check if points were enclosed by a convex hull — or obstacle — ideas from optimization theory was used.

By assigning **P** to be the points constructing the convex hull and **q** to be the query point.

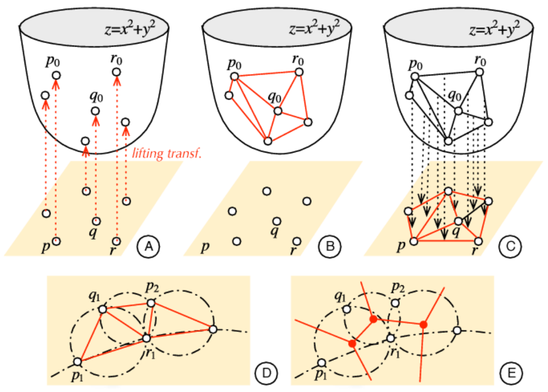


Figure 14.4: Illustration of the Voronoi diagram generation process. *A:* The generator points are projected up one dimension onto a paraboloid using the lifting transform. *B:* The convex hull is generated from the project points using the quickhull algorithm. *C:* The Delaunay triangulation (DT) is calculated and projected down into the original plane using the inverse lifting transform. *D:* The resulting DT in the original plane. *E:* The finished Voronoi diagram is generated by calculating the dual of the DT. [Image credit: Candeloro et. al. [123].]

The in-hull check can then be formulated as an LP problem:

$$\begin{aligned} &\min\ \mathbf{c}^\top \mathbf{x} \\ &\text{s.t. } \mathbf{A}\mathbf{x} = \mathbf{b}\ , \end{aligned} \tag{14.4}$$

where

$$\begin{aligned} \mathbf{A} &= \begin{bmatrix} \mathbf{P}_{N\times 3} & \mathbf{1}_{N\times 1} \end{bmatrix}^\top , \\ \mathbf{b} &= \begin{bmatrix} \mathbf{q} & 1 \end{bmatrix} , \quad \mathbf{c} = \mathbf{0}_{1\times N}\ . \end{aligned} \tag{14.5}$$

The idea behind this is to check if the end points of the Voronoi edges can be expressed as a convex combination of the points defining the convex hull. If a solution to this LP problem exists, the point lies within the convex hull. Furthermore, some of the graph edges are what is known as unbounded edges, meaning edges that extend from a vertex towards infinity. These edges will only be evaluated if there exist no other path alternative. Therefore, these can be safely removed, which helps memory requirements. An example diagram calculated based on this method using the simulated environment shown in fig. 14.13, with a simplified representation of the full Voronoi planning pipeline shown in fig. 14.5. Results from simulations running this cluster-hulling approach is included in section 14.5.1.

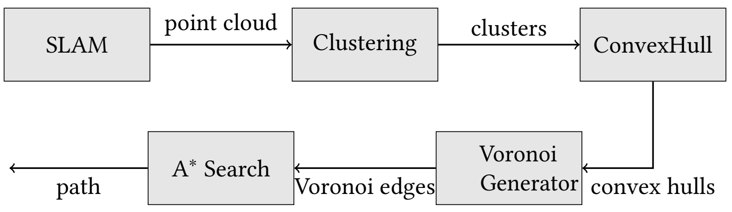


Figure 14.5: Crude overview of the process for planning based on dynamic Voronoi diagram generation.

14.2 Sampling-based Planning With Kinematic Constraints

Implementations of path planning algorithms, especially when it comes to sampling-based methods, are not scarce. Several tried and tested frameworks exist, including *Search-Based Planner Lab* (SBPL), the *Open Motion Planning Library* (OMPL), and the *MoveIt! Motion Planning Framework*. Due the ease of which it can be customized, added to, or otherwise modified to suit different contexts and problems, OMPL [192] was used as a stepping stone in this thesis for the implementation of a sampling-based path planning system. Coincidentally, OMPL also includes implementations of a select few informed planners, courtesy of the author behind the Informed RRT* and BIT* algorithms [137], [193], which was the main basis of informed algorithms used. The choice of an informed planner is largely based around the main task being pipe inspection. As this most often occurs in very open environments, waypoints along the pipe are quite often placed fairly straight forwards with respect to the previous waypoint and the current orientation. By nature of the informed sampling scheme discussed in section 12.4, the sampled path then quickly converges to the straight line connecting η_i and η_g. Example trees from solving a single-query planning problem are shown in fig. 12.8.

For these tests — and planning in general going forward — the main path optimization objective was minimum path length, i.e. finding the path $\pi^*(\varpi) \in \Pi_f$, as defined in eq. (11.7), where $c_p(\pi)$ is the length of the path π. This optimization objective is straight forward to modify to a given scenario. For example, if the robot detects an imminent collision, the planning module will tell the control system to halt before attempting to find a collision-free path. In this specific replanning stage, the optimization objective is modified to include obstacle clearance to help reduce the number of future collision states. This is done by by modifying the objective function to evaluate the path length **and** the distance between the collision model and the current known environment. The path cost then becomes

$$\begin{aligned} c_p(\pi) &= w_l S(\pi) + w_c C(\pi) \\ C(\pi) &= [\,\min \|\mathcal{A}(\mathbf{x}) - \xi\|\,]^{-1} \quad , \end{aligned} \tag{14.6}$$

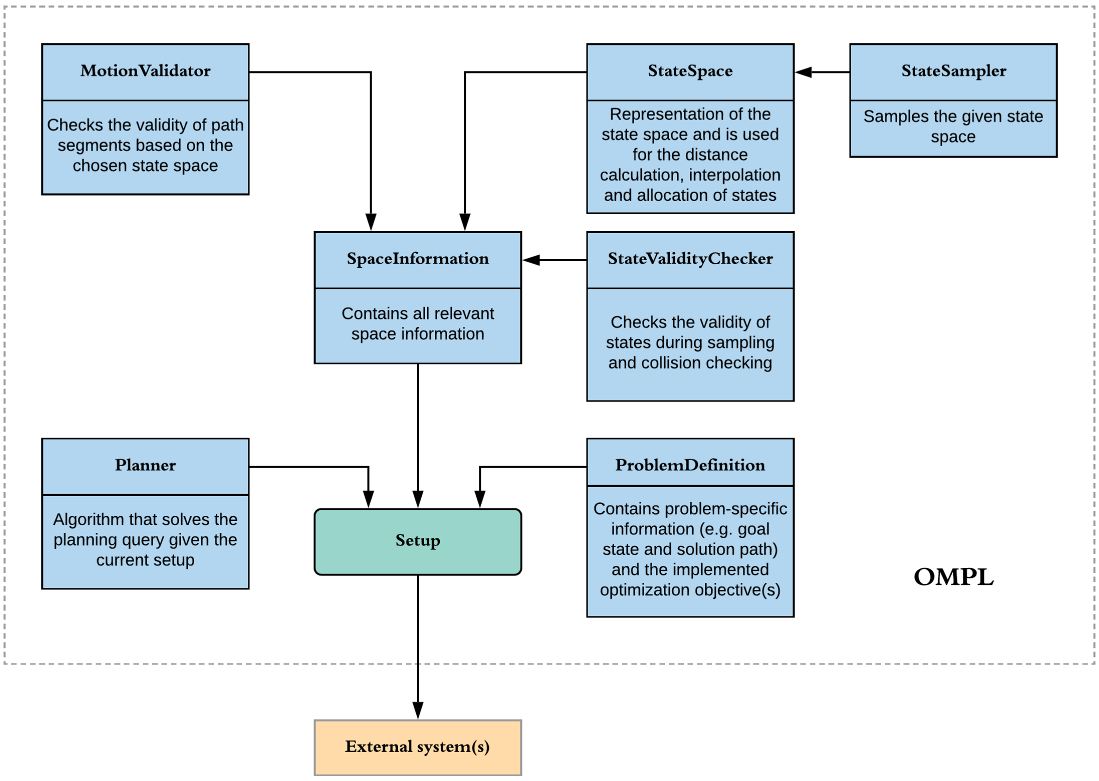

MotionValidator
Checks the validity of path segments based on the chosen state space
StateSpace
Representation of the state space and is used for the distance calculation, interpolation and allocation of states
StateSampler
Samples the given state space
SpaceInformation
Contains all relevant space information
StateValidityChecker
Checks the validity of states during sampling and collision checking
Planner
Algorithm that solves the planning query given the current setup
Setup
ProblemDefinition
Contains problem-specific information (e.g. goal state and solution path) and the implemented optimization objective(s)
OMPL
External system(s)

Figure 14.6: Illustration of the most noteworthy OMPL-modules that needed implementation or modification and how they fit together. [Figure is essentially an excerpt of OMPL's own hierarchy illustration [192].]

where $S(\pi)$ denotes the path length, ξ the closest portion of the currently known environment and w_l, w_c the relative weights on each of the cost values. $\|\mathcal{A}(x) - \xi\|$ defines the distance between the robot in state x and the surrounding obstacles, which is then inverted such that paths that have low clearance become more costly. The calculated clearance is also numerically capped such as to avoid paths with unnecessary large clearance, seeing as much of the environment is potentially unknown. Just how the distances and collision detection is performed is discussed further in section 14.3.

To solve the path planning problem, OMPL provides a modular interface which can be adapted and expanded upon to meet application-specific requirements. How this interface fits together is illustrated in fig. 14.6. During execution, the planner samples the state space $X = C$ and propagates the states $x \in X$ according to a defined model. In the case of a 3D environment with no specific constraints, the state space equals the three-dimensional Euclidean space, i.e. $X = \mathbb{R}^3$, and states are sampled uniformly and interpolated using straight-line interpolation. The resulting path is then comprised of a number of piece-wise linear segments. Due to this, careful path smoothing is needed to obtain paths that satisfy the continuity constraints — for this, natural cubic splines are used due to the implementation specifics of the control system. This of course also holds for the case where rotations around all axes are allowed, with the state space defined as $X = \mathbf{SO}(3) \times \mathbb{R}^3 = \mathbf{SE}(3)$. This approach works well for fully actuated robots with free movement in all axes, such as UAVs. Even though the USM is also manoeuvrable in 6 degrees of freedom, it is not necessarily advantageous to always utilize all of these motions. An assumption made regarding the control system, for example, is that roll is near-zero due to active roll stabilization. In situations where mapping is performed based on visual sensors, large fluctuations in roll (and to some degree pitch) is unwanted. Furthermore, since many underwater vehicles are fairly restricted in roll and pitch motion, it is beneficial to allow for a higher or lower number of DoF depending on the application.

In the case of a 6-DoF system, the state can be defined as $\boldsymbol{\eta} = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}$, consisting of the 3D position and orientation. The kinematic model can then be described as

$$\dot{\boldsymbol{\eta}} = \begin{bmatrix} \mathbf{R}_b^n(\phi, \theta, \psi) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{T}_\Theta(\phi, \theta, \psi) \end{bmatrix} \boldsymbol{\eta} \,, \tag{14.7}$$

where $\mathbf{R}_b^n$ and $\mathbf{T}_\Theta$ are the BODY-to-NED rotation matrix and the angle rate transformation matrix [15]. If one assumes a near-zero roll and pitch angles ($\phi = \theta \approx 0$), the model simplifies to a 3-DoF model:

$$\begin{bmatrix} \dot{x} & \dot{y} & \dot{\psi} \end{bmatrix}^\top = \begin{bmatrix} v\cos(\psi) & v\sin(\psi) & w \end{bmatrix}^\top \tag{14.8}$$

where v and w are the surge and turn rate, respectively. This is a much used model and provides a simple, yet good approximation for most surface ships as well as many underwater vehicles [15]. Keen-eyed readers might recognize this as nearly equivalent to the kinematic car model used as the basis for Dubins path generation. As a result of this, it was of interest to attempt state sampling directly in a Dubins path-based state space to include yaw (and pitch) kinematics into the planning procedure such as to guarantee that the maximum curvature constraint discussed in section 12.6 is upheld. To be used in 3D path planning, however, an extension is needed.

14.2.1 3D Dubins State Space

In its original forumlation, a Dubins path refers to the shortest path connecting two points in $\mathbb{R}^2$, consisting of a set of straight lines and circular arcs of bounded curvature [149]. Dubins path interpolation is a much researched and used tool in robotics.

The original approach was to calculate each of the path segments, and then compare segments to obtain the shortest path. This is a time-consuming process, and it would be preferable to instead extract the shortest path directly. An approach to achieve this, through the definition of the Dubins set and a logical classification scheme, was presented by Shkel and Lumelsky [194]. This allows for the computation of only the segments that are present in the final shortest path, which significantly reduces the computational time (see [194] for an in-depth derivation of this classification approach). Reducing the Dubins calculations in this way is the basis for how the Dubins state space has been implemented previously [192]. To be used for planning in three-dimensional space, however, the dimensionality needs to be increased.

Extending a Dubins path to 3D space is well known, and can be realized in different ways. One such method of incorporating the z-axis, motivated by fixed-wing UAVs, first calculates the 2D Dubins path between the start and goal configurations, before performing ascending or descending spiral motions to reach the desired height [195]. This approach has been simulated and implemented on fixed-wing UAVs [196], but is deemed more appropriate for cases where the vehicle can reach a certain cruising altitude for large sections of the path, and for situations where large parts of the environment is known. Instead, a similar approach to the one in [197] was implemented. Again, it is based on the 2D Dubins path, but allows continuous changes in depth. This is done by evaluating the depth of each sampled state and assigning a change in depth based on the observed depth and the sampled state's distance along the path:

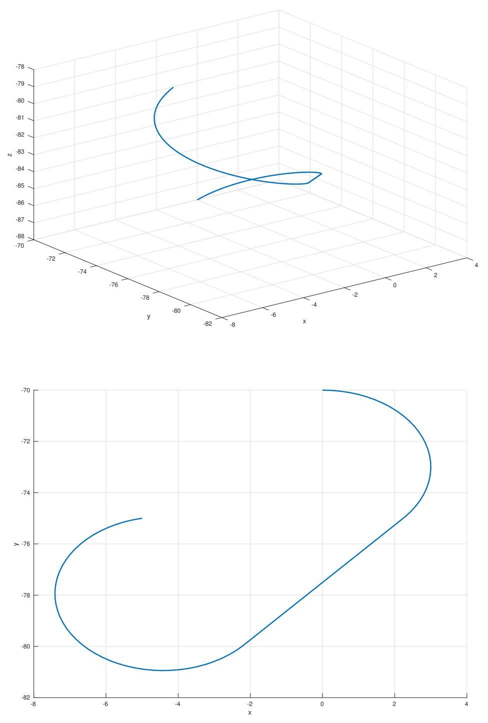


Figure 14.7: Simple solution path from $(x, y, z, \psi) = (0, -70, -88, 0)$ to $(-5, -75, -80, \pi/12)$ based on sampling and interpolation in X_D.

$$z = z_i + \frac{s(x, y)}{s(x_g, y_g)} \Delta z \,, \tag{14.9}$$

where $s(x, y)$ defines the arc length in the down-projected 2D space from the initial position (x_i, y_i) to a point (x, y) along the path and Δz is the difference in depth between the initial state $(x_i,\ y_i,\ z_i)$ and the goal state $(x_g,\ y_g,\ z_g)$. This extended Dubins state space, X_D, is then defined by the robot's 3D position and its yaw angle: $(x,\ y,\ z,\ \psi) \in X_D = \mathbb{R}^3 \times \mathbf{SO}(2)$. The problem is then reduced to finding the collision-free path $\pi(\varpi)$, disregarding roll and pitch, such that

$$\eta_i(x_i, y_i, z_i, \psi_i) \xmapsto{\pi(\varpi)} \eta_g(x_g, y_g, z_g, \psi_g).$$

A simple example of such a path is shown in fig. 14.7.

Similar approaches can of course be done with more complex formulations, such as basing the state space on clothoids or Fermat spirals, for example; this can further reduce the need for path post processing. The Dubins approach was used in this work mostly because of its computational efficiency due to fast path calculation. Furthermore, it is worth noting that, if planning without constraints on the kinematics, the informed subset takes the shape of an ellipsoid — as mentioned in section 12.4.1. With included constraints, this subset can become much more complex, resulting in less efficient sampling. This can be circumvented, however, by that the Euclidean distance between two states, s_1 and s_2, is always shorter than, or equal to, that of the constrained distance between the same two states. I.e.

$$\|s_2 - s_1\|_2 \leq \|s_2 - s_1\|_{\mathcal{K}} \ ,$$

where $\| \cdot \|_{\mathcal{K}}$ indicate the distance metric in some set $\mathcal{K}$ defining a constrained state space. Thus, the informed subset contains the constrained subset, given that the optimization objective is minimizing path length. By doing this under-approximation of the distance metric, the state sampling can be performed more efficiently. This is also the reason for the straight-line tree segments apparent in fig. 12.8 and fig. 14.16.

Finally, before being forwarded to the control system, the path is simplified — by removing superfluous waypoints[2] — and smoothed using natural cubic splines, ensuring the required C^2 continuity.

14.3 Collision Detection

In most robotic applications, path collision checks are essential. These checks are used to determine if a state is collision-free, i.e. if $(x, y, z)^\top \in \mathcal{W} \setminus \mathcal{O}$. This is needed to determine if a candidate state is acceptable as part of the solution path, e.g. when performing state sampling. An underlying assumption for collision checking is that the environment is known. In situations where the environment can change by courtesy of dynamic obstacles, for example, or in exploration settings where the map is only partly known, this assumption does not hold. As such, the entire path needs to be checked for collisions as new information is gathered. To this end, it is also favorable to not just check that the state (x, y, z), but the whole configuration η stays collision-free, i.e. $\eta \in C_{\text{free}}$. To be certain that a given configuration η is indeed valid, a new set was defined. Namely the bounding volume of the

[2]Such as waypoints that are close and together add up to a straight line.

robot, $\tilde{\mathcal{A}}$, which encapsulates $\mathcal{A}$ with additional safety margins such that $\mathcal{A} \subsetneq \tilde{\mathcal{A}}$. This bounding volume is exemplified in fig. 14.8.

For performing such collision checks, the *Flexible Collision Library* (FCL) [198] was used. FCL is a C++ library for collision and proximity queries, providing 3D collision checking for the whole robot configuration with different map representations. Using this library, the robot model was simplified as a rectangular box (in the case of ROV simulations) or a cylinder (if simulating the snake robot), with the map being represented using FCL's built-in OctoMap support. Additionally, these models could also be implemented as multi-jointed if planning for each link is favored.

When sampling the state space, OMPL requires a state validity checker to be implemented. This was implemented as a function

$$\text{bool isStateValid } (\eta) \ \left\{ \text{return isCollision(} \tilde{\mathcal{A}}(\eta) \text{)} \right\}$$

which determines if the sampled state results in a collision with the received SLAM map — stored as an OctoMap OcTree — based on the state's pose using FCL's object-environment-collision checking. Furthermore, FCL provides *continuous collision detection* (CCD), which checks the robot object for collision when moving along a continuous path. This continuous collision detection allows for determining if the robot collides at any point when traversing the path given updated environment information, while also providing the time and point of contact. OMPL, which was used as the underlying planning interface, allows for the implementation of custom motion validity checkers — often referred to as local planners. Using FCL, this was implemented as a custom class inheriting from OMPL's *MotionValidator* class. With FCL's CCD, a member function

$$\text{bool checkMotion } (\textit{state1, state2, last\_valid}) \ \{ \text{ return is\_collide } \}$$

which is called for each pair of states in the current path, was implemented. This function transforms the robot's calculated poses along the path and returns true if a collision happens, as well as the time of collision and the last valid state along the path[3]. For increased safety, $\tilde{\mathcal{A}}$ was chosen sufficiently larger than the actual robot dimensions. See fig. 14.8 for a simple illustration of the concept. By checking for collisions using an expanded bounding volume, the space around the path can be guaranteed safe. This allows for some deviations in the path following while still assuring a collision-free path. For a more detailed run-through of implementation specifics and documentation, see the published paper [198] or the open-source GitHub repository[4].

[3]The last_valid argument is stored as a pair of values, with the first being time of collision on the interval [0, 1] and the second being the last valid path state before collision.

[4]FCL GitHub repository: `https://github.com/flexible-collision-library/fcl`

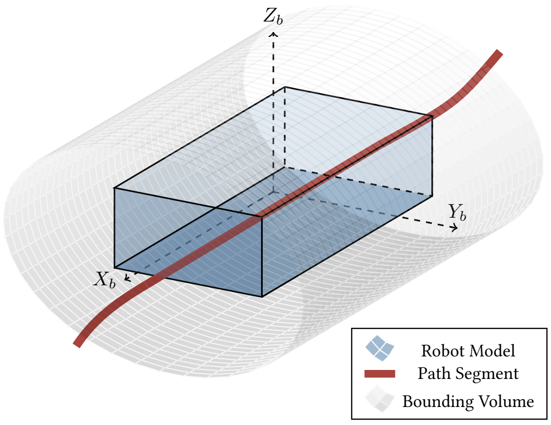


Figure 14.8: Illustration of the added safety around the calculated path. For illustrative purposes, the robot is modelled as a rectangular ROV-type vehicle, and the bounding volume is modelled as a cylinder. The bounding volume is projected along the calculated path segments and checked for intersections with the environment using the robot's pose.

14.4 Environment Exploration

The chosen exploration strategy is a frontier-based one, used primarily due to its capability of handling large environments without the risk of premature termination. The traditional nearest frontier [158] was used as a starting point. Instead of continuously choosing the nearest frontier, however, a cost function that incorporates path distance, change in orientation, depth and potential information gain was used to evaluate frontier candidates. To find the candidate frontiers, the current occupancy grid is searched. Cells that are not occupied and has neighboring cells that are free and unknown are marked as frontier cells. These cells are stored and converted to a point cloud. Candidate cells are then found through clustering of the frontier point cloud, similarly to what was discussed in section 12.5 and what was done in [167], [169]. After the frontier clusters are extracted, goal point candidates, p_c, needs to be calculated. The most used method is to use the cluster centroids as candidates. This was tested, but due to the potential impact of outliers in the centroid calculation, the geometric median was used instead. Calculating the geometric median was done using Weiszfeld's algorithm [199], which is an iterative procedure seeking to decrease the sum of distances between samples. It essentially works as an *iterative re-weighted least squares* algorithm, updating the guess p_c based on weights calculated from the inverse distances, and can be summarized as

$$p_{c,i+1} = \frac{\sum_{j=1}^{n} \frac{x_j}{\|x_j - p_{c,i}\|}}{\sum_{j=1}^{n} \frac{1}{\|x_j - p_{c,i}\|}}, \tag{14.10}$$

where x_j are samples from the points in the evaluated cluster and $p_{c,i}$ is the estimated geometric median at iteration i. This was implemented as a C++ function

void geometricMedian (*cluster*, *& candidate*) ,

with *cluster* being a vector of points — or more specifically octomap::OcTreeKey structures — and *candidate* being the calculated median, or centroid if the algorithm does not converge.

While being a much used method for geometric median calculations, there are rare cases where the algorithm fails to converge. In these cases the cluster centroid is used instead [5]. These candidate cells are then checked for proximity to obstacles, through neighboring cell evaluation, and are dropped if they are found to be too close to an obstacle for the robot to safely approach.

When choosing among the found frontier candidates, an evaluation function is used that can be formulated as

$$c_f = \min \{ w_l \|p_c - p_r\|_2 + w_d \Delta d + w_\theta \theta - w_I I \} \ , \tag{14.11}$$

where P_f is the position of the frontier candidate, p_r is the current robot position, Δd is the depth difference between the robot and the frontier candidate, α is the change in orientation between the current steering angle (ψ) and the frontier candidate, I is the potential information gain, and c_f is the cost of the best frontier candidate. The values w_l, w_d, w_α, and w_I, are used to weight spatial distance, orientation change, depth change and potential information gain, respectively. These weights can be specified by the user, but for the simulations discussed in section 14.5.3, the weights were set as $w_d < w_l < w_I \sim w_\theta$, slightly favoring w_θ.

Using this frontier weighting scheme forces the robot to favor exploration routes that increases the amount of known information in the map, while simultaneously keeping it from deviating too much from the current depth level and keeping the exploration goal in, or close to, its FOV, and keeping the paths at minimum length given the constraints. This lessens energy consumption by reducing the amount of steering needed and results in paths that are reminiscent of environment coverage-paths, favoring exploration that maximizes new information before performing relatively costly descending or ascending control actions. By favoring small changes in yaw, the assumption of straight pipes — at least over relatively short distances — is implicitly built into the exploration strategy. How this can be useful is more closely discussed in e.g. section 16.1.1.

To estimate the potential information gain, the number of unknown compared to known cells inside a bounding volume centered at the frontier candidate is used. This gives a

[5]For potential future implementations, the algorithm presented by Cohen et. al. provides a rapidly converging method for calculating the geometric median without this risk of failure to converge [200].

measure of how much is to be gained by visiting that specific frontier. Information gain and relative angle change is also weighted higher than, for example, closeness due to the fact that closer cells might be observed on the way to more *rich* candidate frontiers. A simplified illustration of this is shown in fig. 14.9.

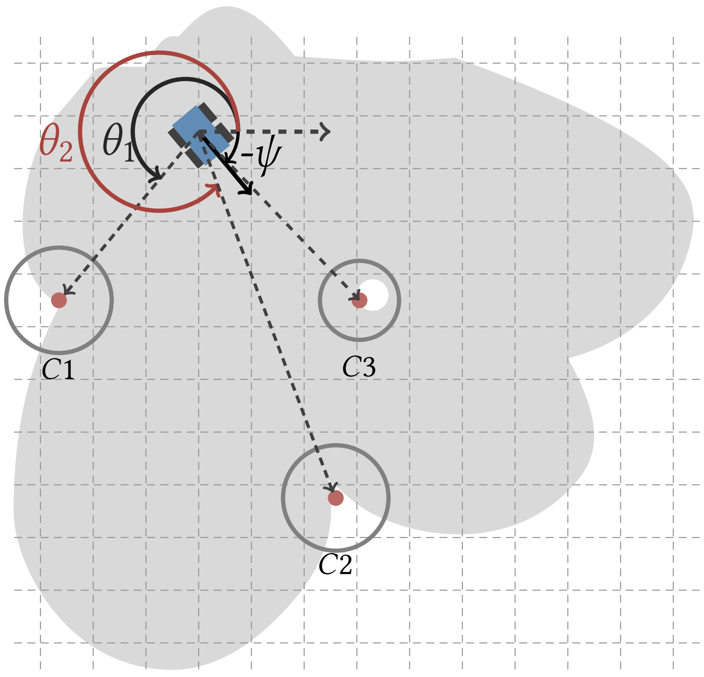


Figure 14.9: Illustration of a very simplified exploration case. Gray indicate known parts of the environment, whereas white indicate unknown. Given three candidate frontiers ($C1, C2, C3$), their position relative to the robot as well as their relative orientation and information gain is evaluated. In this specific case, the robot would choose $C2$ due to the high increase in information and the small yaw change required, even though $C1$ is closer and $C3$ is more straight ahead.

Furthermore, since the environment not necessarily consists of natural boundries, a *geofence* was established to bound the exploration space. This limits the space in which frontiers are examined and evaluated. To start the exploration procedure, an exploration request is needed, consisting of a spatial definition of the geofence and a specified exploration action. The exploration states are currently limited to

IDLE EXPLORING INSPECTING RETURNING HOME

at this time, with actions received along the exploration request altering the exploration states as illustrated in fig. 14.10. If frontiers are found, the best candidate is chosen and sent as an exploration goal to the path planner ROS node.

It is worth mentioning that autonomous exploration was studied as a future extension

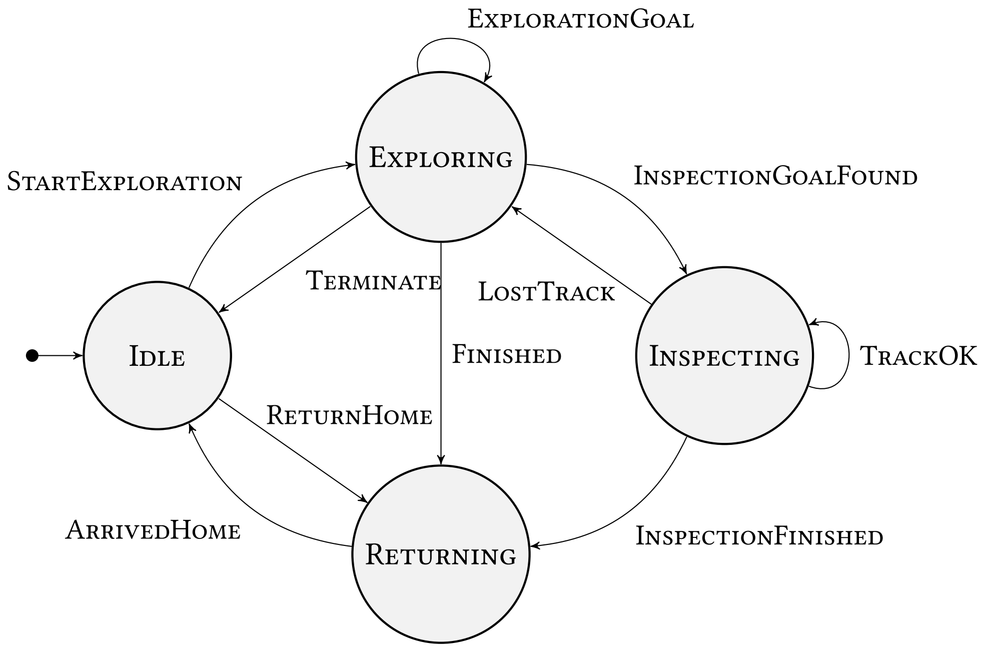


Figure 14.10: Simplified state diagram of the exploration process. The exploration and inspection goals are forwarded to the path planning module. Inspection goals are found through evaluation of semantic point cloud information received from the classification module — e.g. a pipe-labeled point cloud. A received Terminate command will return the robot to Idle at any time, similarly with a received ReturnHome.

for the USM and as a tool for testing more complete autonomous systems by combining different vehicles with different SLAM algorithms, such as autonomous mapping using an inspection class ROV. As far as USM's are concerned, autonomous and procedural goal selection should be based on the immediate task at hand and the observed surroundings. For the case of a pipe inspecting snake robot, equipped with a stereo setup, objective goals should be set based on the estimated pipe position and direction, e.g. using semantic information. Thus, waypoints could be set along the pipe found through object classification (part IV) and forwarded to the path planning module. This will result in the robot being able to autonomously perform pipe following/-inspection, as illustrated in fig. 14.11. Including exploration can also be favorable in such a scenario, however. How this can be done, and more specific details on how semantic information can be included in planning for inspection tasks, is treated in section 16.1.

14.5 Simulations

To gather data and test implementations it was desirable to set up a simulation environment. Many different types of simulation software exist, but seeing as ROS interfacing was the primary concern, ROS-native environments were preferable. Simulations were therefore

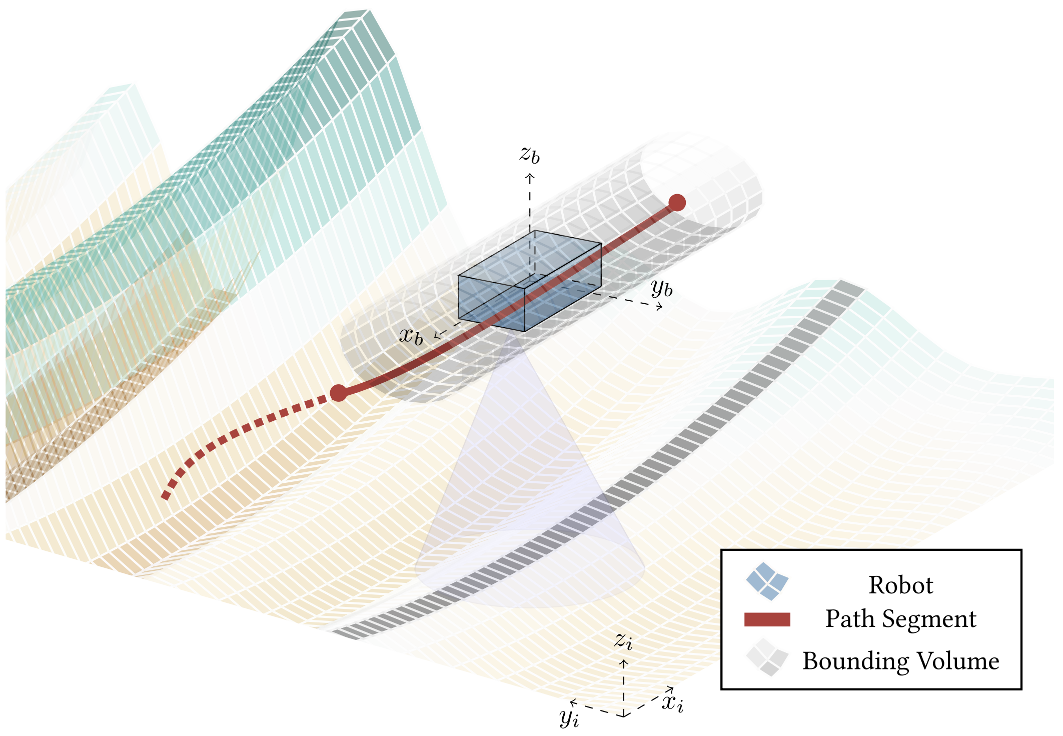


Figure 14.11: Illustration of a pipe following scenario showing the robot ($\mathcal{A}$), the bounding volume ($\tilde{\mathcal{A}}$), part of the current path ($\pi(\varpi)$), as well as a pipe section (gray).

set up using the *UUV Simulator* [20], a Gazebo-based package with good ROS interfacing [201] created to simulate unmanned underwater vehicles (UUVs). This simulator has built-in plugins for different sensors, such as cameras, position sensors and Doppler velocity logs, among others. (See appendix A for more details regarding the UUV simulator.) The idea behind using this system was then to create/modify a setup consisting of an underwater vehicle model with a range of sensors to produce RGB image and point cloud output. Combining cameras and sonars, it should be possible to detect and distinguish between simulated natural structures, such as outcroppings and mountains; and man-made constructions, such as docking stations and pipelines such that this setup could also potentially be used together with the classification module. One such sensor plugin was the imaging sonar plugin made available on SMaRC's open source Github repo.[6], which was used to setup a case with idealized SLAM input for map generation. Modifying the sensor setup and robot model can easily be done by altering the xacro-files to obtain a composition for the specific simulated case. An example setup is shown in fig. 14.12, in which the robot was set to map a small underwater environment based on sonar input.

The simulated environment used was a rendering of an underwater setting with many natural structures and a pipe running through parts of it. It is also worth mentioning that the UUV simulator is still very much under development. Therefore, additional testing on a

[6]Swedish Maritime Robotics Centre. GitHub URL: `https://github.com/smarc-project`

real-world setup should be performed before any actual tests is to be carried out.

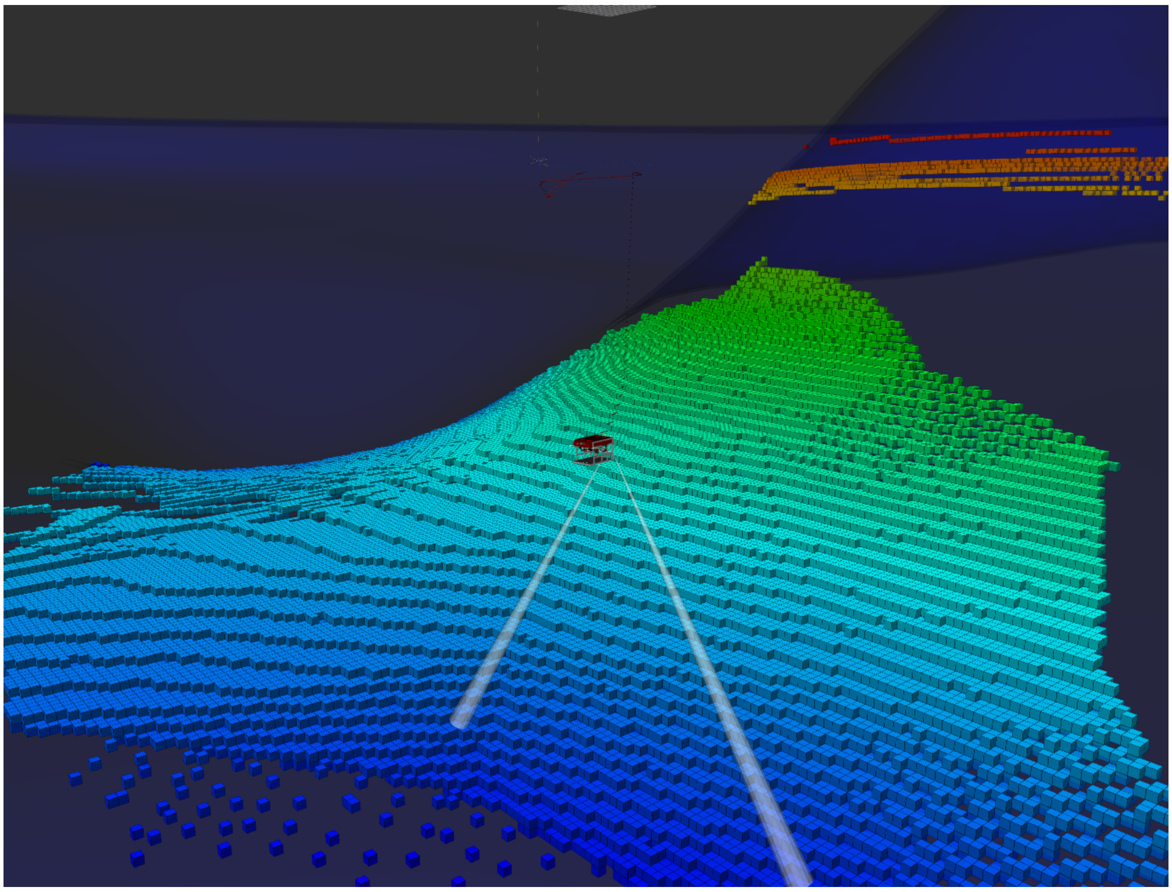


Figure 14.12: Screenshot from running the simulator using multiple sonars to generate a 3D occupancy grid using OctoMap [121].

14.5.1 Voronoi-based Planning in Partially Known Environments

The Voronoi diagram generated using the convex hulls as generator points, as presented in section 12.3.1, is shown in fig. 14.13, whereas the diagram generated with additional virtual generator points is shown in fig. 14.14.

It is evident from fig. 14.13 that the resulting diagram gives sufficient clearing from obstacles in the xy-plane. Some problems are noticeable when examining the side view, however. The first problem that arises involves the overall path quality. Since the underwater environment is very sparse, it has the undesirable effect of producing graphs in which neighboring nodes have quite large jumps in z-values between them. This further demands much more post processing to generate smooth paths that does not include unnecessary large fluctuations in the desired trajectory, which in turn adds unwanted strain on the actuators and the control system.

This was tried avoided by the addition of virtual generator points along certain depth levels, akin to the region bounding performed in [123]. Adding such points forces the generated diagram to give better paths over obstacles, as well as paths that have smaller jumps in depth value between segments, while still preserving as much of the Voronoi properties as possible. An example Voronoi diagram using these virtual generator points is shown in fig. 14.14. Adding such points, however, is not to be done lightly, as their placement would have to be correctly coordinated with the robot's motion and the continuously expanding map. The number of points added would also have to be chosen wisely, as adding too many would quickly lead to a large increase in pruning time. The perhaps biggest drawback of this type of augmentation, is that their depth is dependent on the the environment at hand, and if points were added wrongly, the whole diagram would require recalculation. Secondly, the addition of these kind of virtual generator points fundamentally alters the maximum clearance guarantees provided by the Voronoi diagram. These points could then affect the diagram in such a way that the edges would not necessarily be maximally distanced between obstacles. In most cases, however, they would still be sufficiently far away, but without provable guarantees.

One last complication, which most heavily affects the practical implementation, is the issue of computation time for the graph pruning. While the pre-computation of the convex hulls results in easy in-hull checks, the sheer amount of edges and obstacles result in a very time-consuming process. When running on a 3.2 GHz 12 core i7-8700 CPU, the time average for pruning clocked in at 3 ± 0.2 minutes when running on a occupancy map covering an area roughly 150m^2. Although not fully optimized, this pruning process was deemed too computationally expensive for a continuously expanding environment, even when incrementally building the diagram.

Due to the mentioned drawbacks encountered through implementation and simulation tests, as well as the fact that the assumption for this thesis is that the map is unknown and procedurally built, the Voronoi diagram method was finally deemed unsuitable for practical implementation on a robot.

Furthermore, since the environment is naturally unstructured, it can be argued that it is not always valid to assume that the environment — or the obstacles therein — can be approximated using convex hulls without the chance of encapsulating more than intended, rendering the representation far from exact. This therefore adds another drawback to the aforementioned method building on incremental Voronoi diagram generation, rendering it unsuitable for autonomous exploration of unknown environments. These situations are more often encountered in aerial and terrestrial applications, however, but is still a valid point for consideration. Nonetheless, in cases where the diagram can be pre-computed, the Voronoi diagram is an excellent choice of algorithm for safe and autonomous underwater navigation.

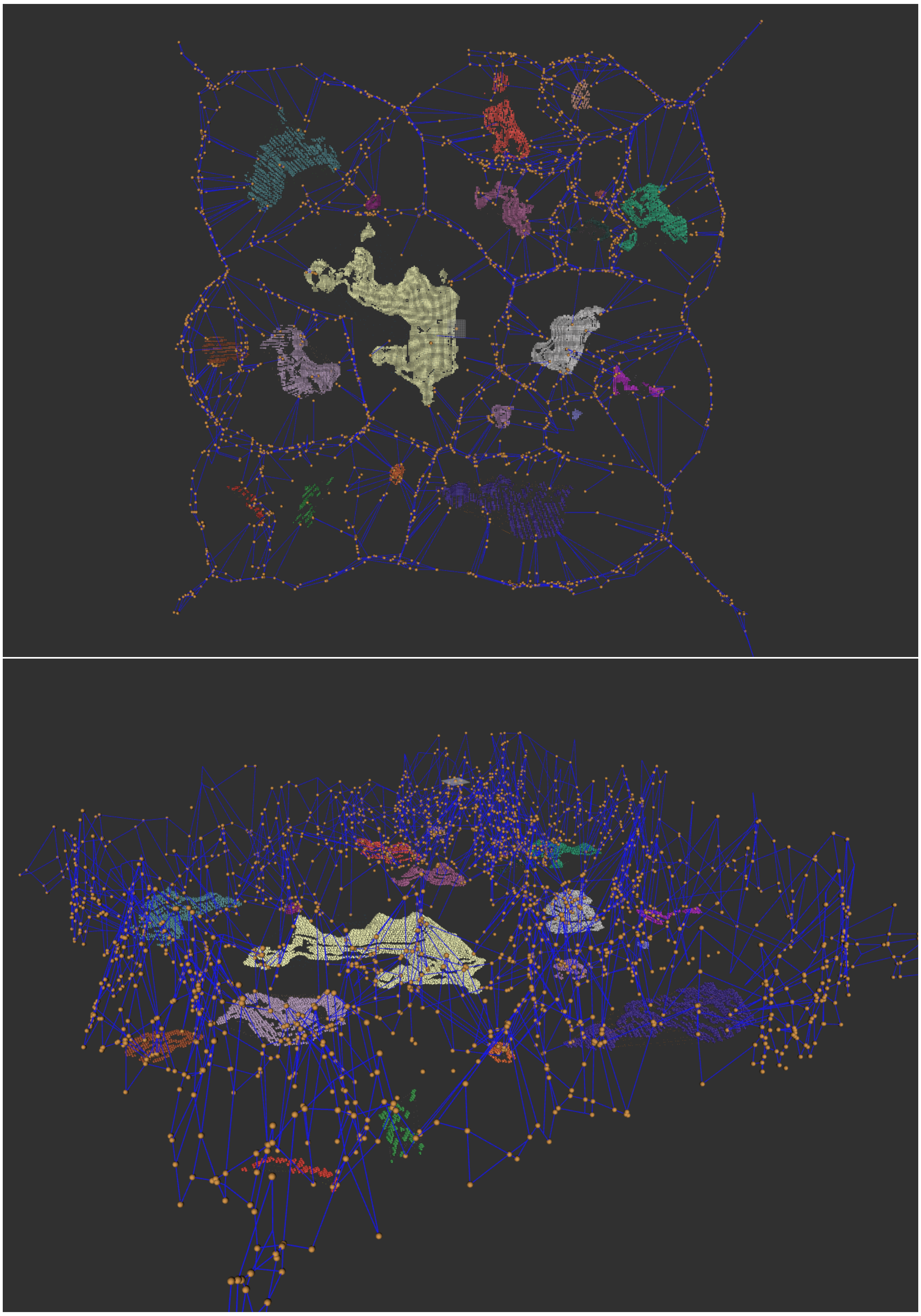


Figure 14.13: Top-down and side view of a Generalized 3D Voronoi diagram based on the environment and convex hulls shown in fig. 14.3 and fig. 12.9. Voronoi edges are shown in blue and Voronoi vertices are shown in orange.

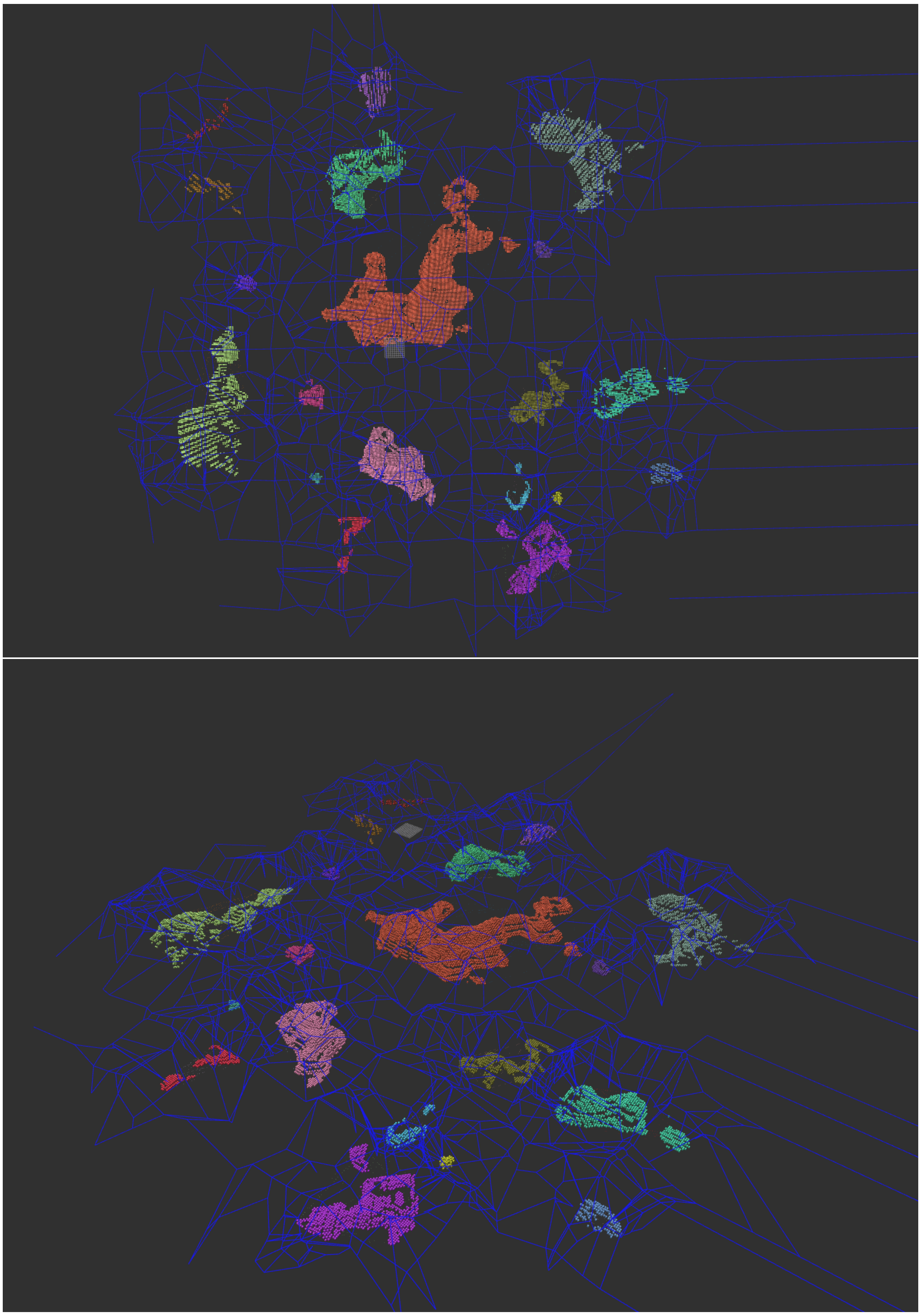


Figure 14.14: Top-down and side view of a Generalized 3D Voronoi diagram based on the same case as in fig. 14.13, but with added virtual generator points.

14.5.2 BIT* and Collision Avoidance

After generating the occupancy map, the planner was given several starting and goal states to test path generation that included easy cases, where the path was almost straight with minor obstacles; to harder cases which included larger obstacles with more change in depth needed for the final path. An example from a simulated planning query using BIT* and the Dubins state space approach discussed in section 14.2 on an ROV in an underwater environment is shown in fig. 14.15. Here, the empty parts of the map are currently unobserved. The resulting path keeps sufficient clearance of the environment while not including too steep changes in depth.

Figure 14.16 shows a similar but slightly easier case which includes the tree structure generated by the planner. This exemplifies the informed manner of the planner and showcases its efficient sampling of X_D.

The collision detection and avoidance part of the planner — while far from the most advanced — was tested repeatedly during exploration simulations. To exemplify its functionality, however, a simple test case was set up. The robot was assigned to plan a path in a fairly empty environment, resulting in an almost-straight path due to the assigned starting and ending orientations. After following the path for a brief time, the map was updated to include a relatively large mountain-structure. The motion validator then reports a collision at some point, indicated by the red dot and yellow rectangle in fig. 14.17, which in turn activates the replanning. In the replanning stage, the planning objective is set to include the path clearance, resulting in a path that weighs both path length and clearance. The resulting paths from this collision test is given in fig. 14.17, with the initial path given in blue, and the recalculated path given in red.

14.5.3 Autonomous Exploration

To test the exploration method, a few different scenarios were constructed. These were set up to evaluate the resulting total path lengths, coverage rates[7], and path quality of the respective exploration strategies.

Firstly, the combined approach discussed in section 14.4 was tested up against the nearest frontier approach, as well as exploration purely based on information gain. This test was conducted in two slightly different environments. They were initially run in a fairly small environment of 375 m^2, without any large obstacles. The results of this is shown in fig. 14.18. The resulting path and map from running nearest frontier is shown in fig. 14.18a. Here it is evident that the frontiers chosen rarely is the best choice available. This results in much backtracking initially and towards the end, especially, and much harder turns.

[7]Coverage rate, in this sense, refers to the time it takes to sufficiently map a given environment.

Comparing this to fig. 14.18b, it is apparent that the information gain approach provides much more efficient coverage of the bounded environment. It is also interesting to note that, through several tests, the information gain approach tended to give paths similar to that of a lawn mower pattern. This is because this gives near-optimal coverage, with the skewness of the resulting path being a result of the wide FOV of the sensor.

By combining the information of the environment and current pose of the robot when choosing the next goal, the path ended up as shown in fig. 14.18c. The resulting path has less sharp turns over all, and gives a higher total coverage overall due to longer straight paths following regions close to the edge such that the sensors obtain information form outside the stated geofence. Furthermore, by better covering the edges, the number of smaller pockets of space that is left unexplored due to them being to small to be assigned cluster status, is reduced, resulting in a better complete coverage, at the expense of a slightly longer mean path length. It is worth noting that this combined approach gives a slightly slower coverage rate than the information gain approach, but it does, in turn, result in a path that imposes less strain on the actuators.

Something to note regarding fig. 14.18 is that the holes evident in the map that is within the assigned bounds were in fact observed, but a minor bug in the sensor-to-occupancy grid conversions using the simulator resulted in small patches being deleted. This is, however, not the case for fig. 14.20, where the holes present are due to the respective frontier candidates being deemed not safe enough to be approached closely enough to be fully observed.

When comparing just the paths, it is apparent that the path lengths vary quite a bit. The nearest-frontier approach tend to give the longest paths, whereas the pure information gain strategy gave the shortest paths by some margin. The path lengths from several tests are summarized in table 14.1. This shows that the information gain variant gave the shortest paths, with the combined method falling in between, but with the smallest variation in length. Although these tests were conducted in a fairly small environment, running similar tests in larger, more complex scenarios, made it clear that these results hold.

Table 14.1: Mean path lengths when exploring a small environment using three different strategies for weighting, pure nearest frontier or information gain, or the combined approach described in eq. (14.11).

Exploration Strategy	Nearest Frontier	Information Gain	Combined Approach
Mean Path Length [m]	113.81 ± 21.88	99.68 ± 17.09	104.40 ± 15.22
Median Path Length [m]	116.34	99.39	109.83

Figure 14.19 compares the obtained coverages more directly. Here, the calculated map coverage each of the three mentioned strategies are plotted against the total mission time. From this, it is clear that, while gaining much ground quickly, the nearest-frontier approach has a much slower convergence rate than the two others. This early increase is due to the

excessive turning to accomodate for the close frontiers chosen. Focusing on information gain does result in the fastest convergence. Although seeing as the combined approach has an almost identical convergence rate, the fact that the resulting path is smoother wrt. the turning angle, it was deemed the best approach. Running an exploration task with the combined approach in a larger environment, including larger variations in the surroundings, resulted in the map shown in fig. 14.20.

14.6 Summary

In this chapter, the implementation specifics regarding both a Voronoi-based and an informed sampling-based planner, a simple collision detection scheme, and an autonomous exploration strategy were discussed. These methods were then tested and evaluated on various simulation cases in an underwater environment.

Seeing as the environment is naturally unstructured, the convex hull approximation used for the generation of the Voronoi diagram is not always valid. If traversing an underwater cavern, for example, this approach would not work. However, in cases where this approximation holds, the Voronoi diagram can be a powerful tool for navigation in mostly known environments. Using Voronoi diagrams for exploration tasks ended up being discouraged due to the pruning time, the possible additional points needed on the z-axis, and the fact that the maximum clearance property of the Voronoi diagram gives no guarantees wrt. unexplored parts of the map. I.e. that paths that seem sufficiently safe in the known map might cut short, or even collide, with unknown obstacles. In cases where a static map is known beforehand, for example when traveling between known subsea structures, this method shows promise. How to better handle the distance calculations based on depth level to avoid overly oscillating graph edges is left as future work. Also, to better cluster obstacle points, a robust implemetation of Chan's optimal clustering algorithm [188] would be preferred to the Quickhull algorithm, due to its better runtime.

The biggest benefits of using the sampling-based planner were its relative simplicity and the ease of which additional constraints could be added. This resulted in a planning module capable of producing nearly-optimal collision-free paths that coincided with the constraints imposed by the control system. Included into this was a simple collision detection and -avoidance scheme built around the FCL library which provided efficient collision detection and validity checking when performing state sampling and path following.

A state-based exploration strategy for autonomous environment exploration was implemented. This method used efficient point cloud clustering to group frontier cells from which candidate goals could be extracted, which were optimal given the specific weighting of the evaluation function while satisfying certain safety requirements wrt. obstacle proximity. Through simulations this method provided fairly efficient map coverage while also easing strain on rudders. A future extension to this could be to include sonar measurements with the provided SLAM map to better estimate optimal view points for safer and more efficient exploration.

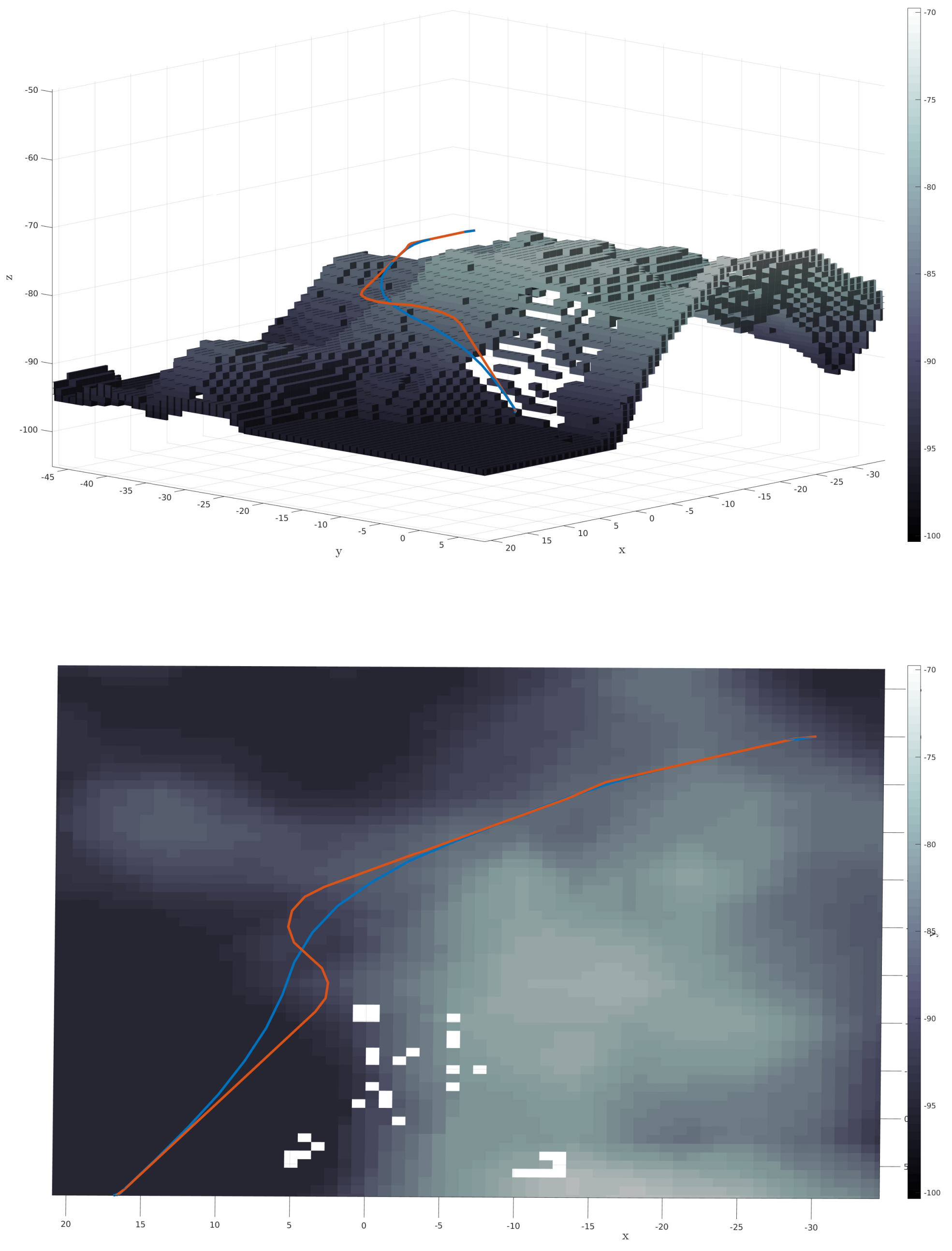


Figure 14.15: **Top**) Resulting calculated path from using BIT* with state sampling from the 3D Dubins state space (red) and its cubic spline-smoothed variant (blue). **Bottom**) Same paths shown in a top-down view.

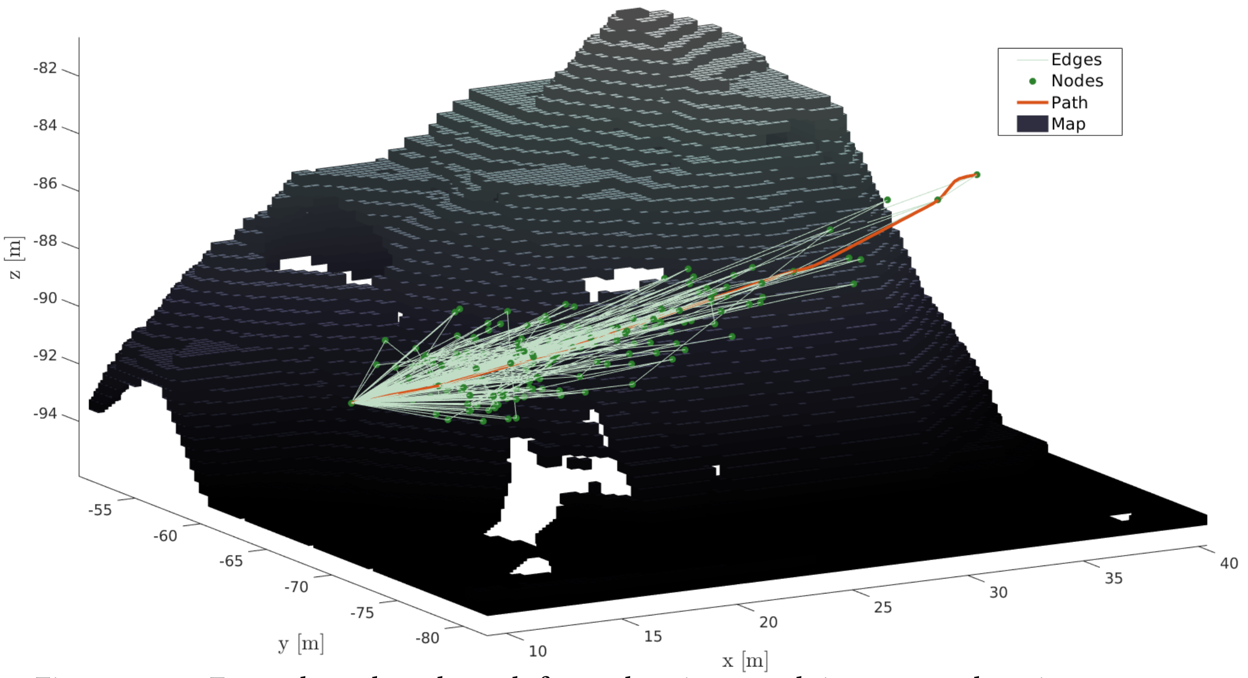


Figure 14.16: Example path and graph from planning a path in a mapped environment, as further discussed in section 14.5.3 (fig. 14.20).

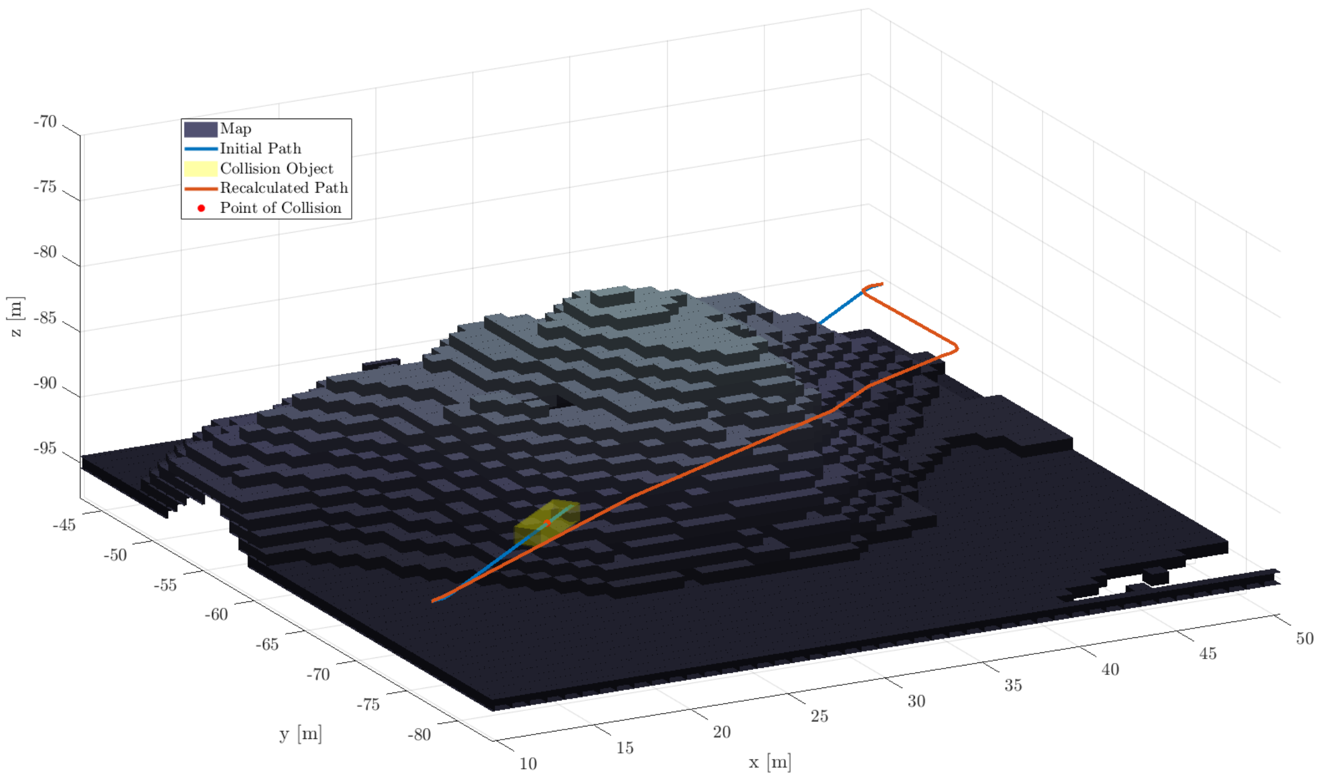
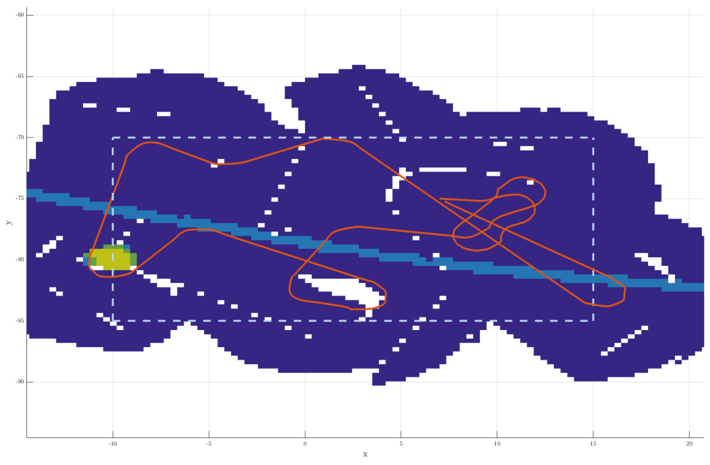
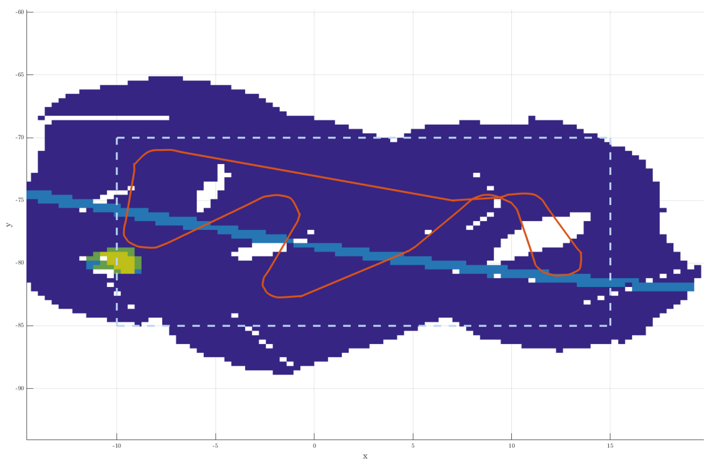


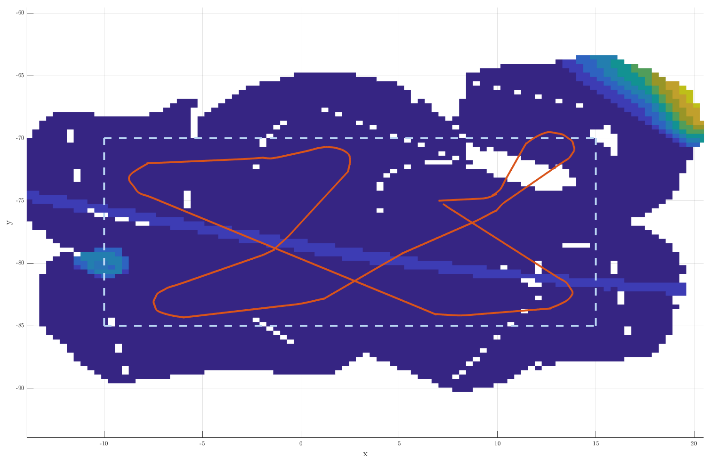
Figure 14.17: Example path resulting in a collision. When collision is detected between the robot and the environment along the original path (blue), replanning is initiated using the previous safe state, resulting in a new safe path (red). The point of collision (red) is at the center of the robot model (yellow) translated along the initial path. [The blue path would never ocurr, this scenario was forced by suddenly inserting the mountain in front of the robot.]



(a) Nearest-frontier path and map.



(b) Information gain-based path and map.



(c)

Figure 14.18: Example paths and maps from simulating exploration of a small area using three exploration variants. (**a**) Nearest frontier. (**b**) Information gain. (**c**) Combined approach.

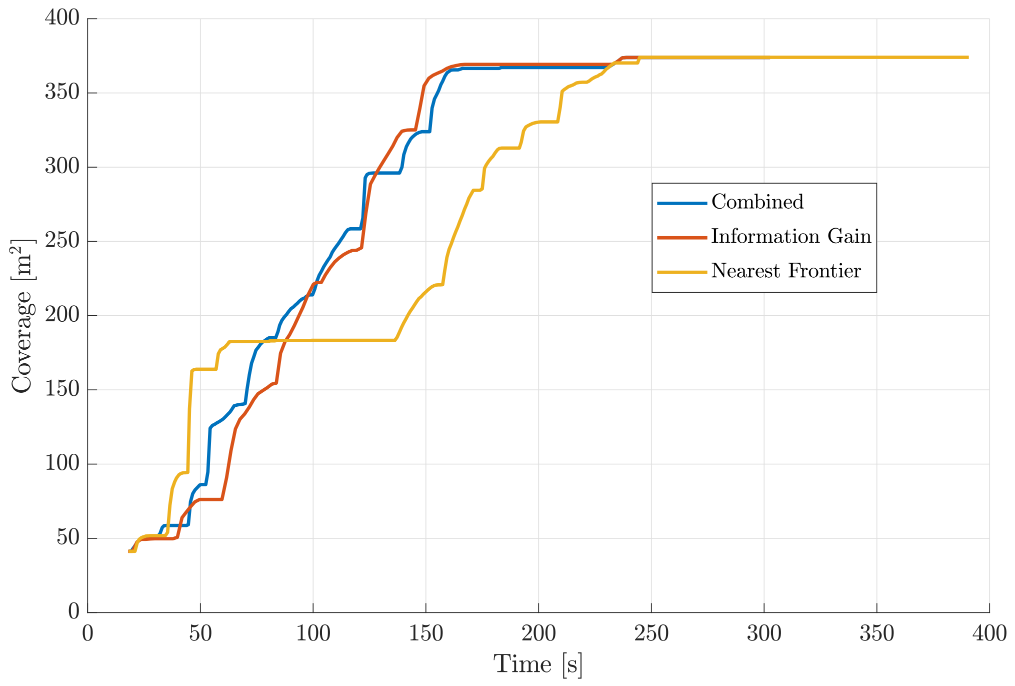
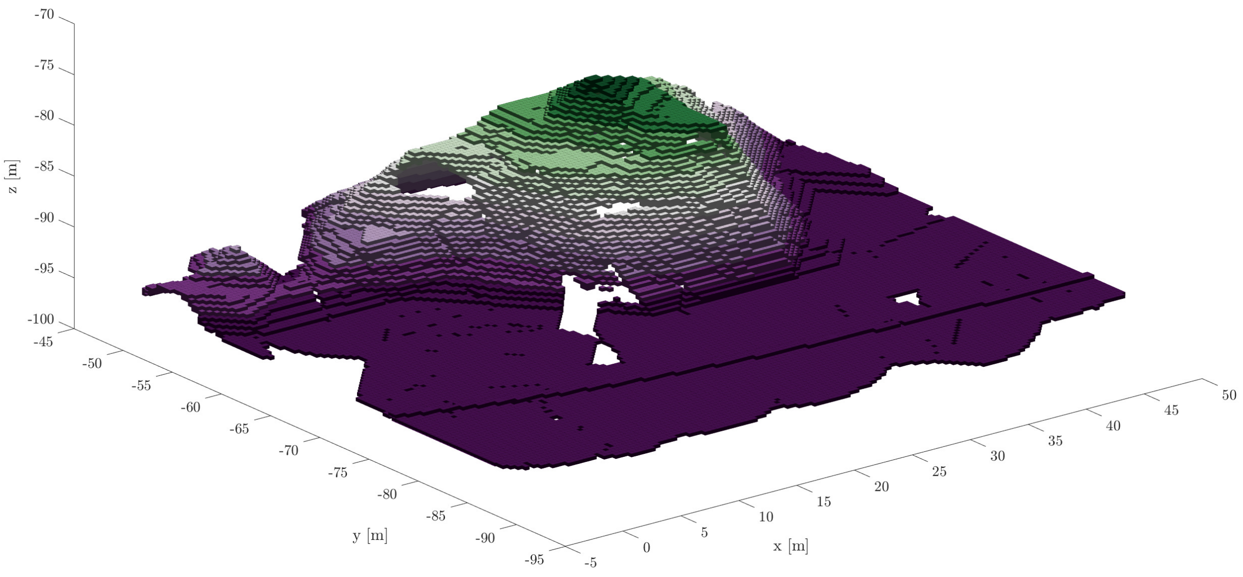


Figure 14.19: Comparison of the obtained coverage using the three different exploration variants when exploring a small environment covering 375 m^2.



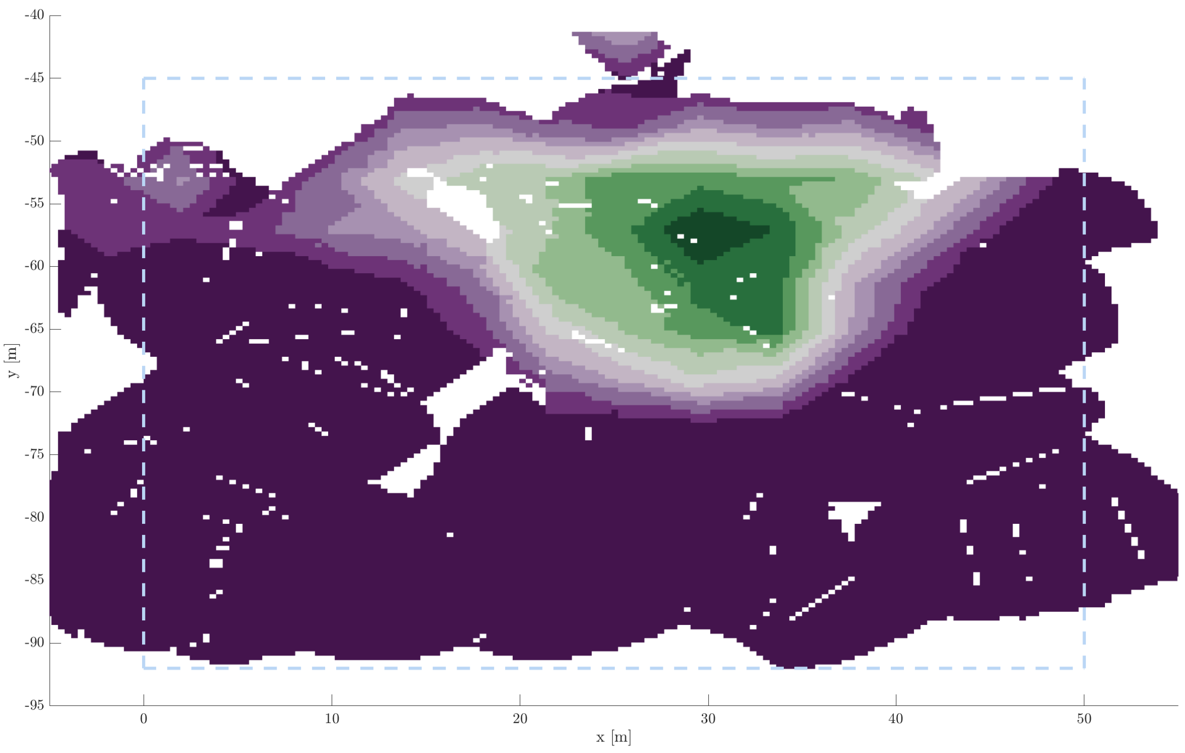
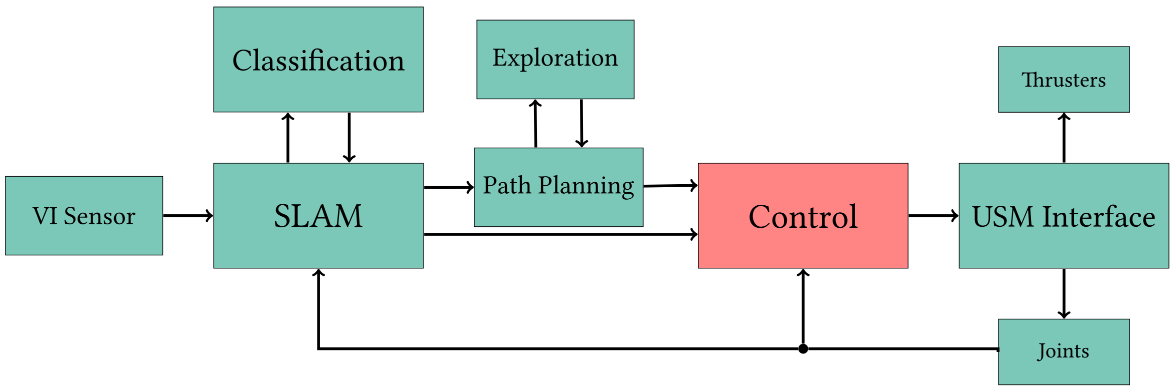


Figure 14.20: Example map from exploring the area within $x \in [0, 50]$, $\quad y \in [-45, -90]$ seen from the side and top-down.

Part VI

Control



15 | Control System

THE main goal of a control system is to ensure that the robot is able to position and maneuver in accordance with a set of specifications. In general, such specifications may include path following, dynamic positioning, trajectory tracking, or specific steady state maneuvers, see e.g. [15] for a presentation of these topics. With respect to the overall autonomous architecture presented in section 3.1, an USM is presented with the task of following pipes. Pipe following is a specific task that, in large, open, underwater environments could be solved with straight line path following — because the pipes stretch over large distances without bending too much. Such a solution, however, is hand-crafted towards this particular application. When the environment turns increasingly cluttered by objects, forcing the USM to exploit its articulated structure, this will fail. Consequently, 3D curved path following exploiting the kinematics of the USM seems to be the more practical approach.

Generally, 3D guidance algorithms have been developed to ensure convergence for general continuously differentiable curves, see e.g. [202], [203]. In the specific context of USMs however, no such algorithms exist to ensure *full-body convergence* for such curves — i.e. that multiple links follow the desired curve in addition to the base link[1]. A solution for 3D straight line path following is presented in [204], where the dynamic control and thruster allocation algorithms by no means are restricted to the particular application. The guidance and kinematic controllers, on the other hand, are restrictive in the sense that they do not guarantee full-body convergence.

Finally, maximizing perceptual performance because of cluttered, unstructured, underwater environments is necessary to ensure autonomous operation. With a limited FoV and a lack of environmental features, high speed and accelerating maneuver degrade — and possibly hinder — the USMs ability to navigate. Therefore, attitude and velocity control of links with exteroceptive sensors such as cameras are desirable.

The control system presented in this chapter is designed to solve the above-mentioned problems while providing convergence and stability guarantees from a kinematic perspective. Firstly, the interface between the control system and the rest of the autonomous system is presented in section 15.1. A guidance controller providing the basis for kinematic control is presented in section 15.2. A kinematic control methodology ensuring *full-body convergence* is presented in section 15.3, with a constructive cascaded stability proof. Finally, experimental results are provided and practical tuning of the controllers are discussed to

[1]A formal definition will be introduced in section 15.3

ensure robust operation.

15.1 Interface

To capture the top-level state of the autonomous system, communication lines with the state machine have to be established. The control system implements an internal state machine, see fig. 15.1, that handle events according to the Liskov substitution principle, see e.g. [24]. That is, the state of the top-level state machine determines the state of the control system state machine. State transitions in the SLAM and path planning state machines are treated as events in the top-level state machine. If these events do not trigger a state transition, they are handled explicitly in the lower level state machines. An example of such a low level state machine is shown in fig. 15.1, where events are the named state transition arrows.

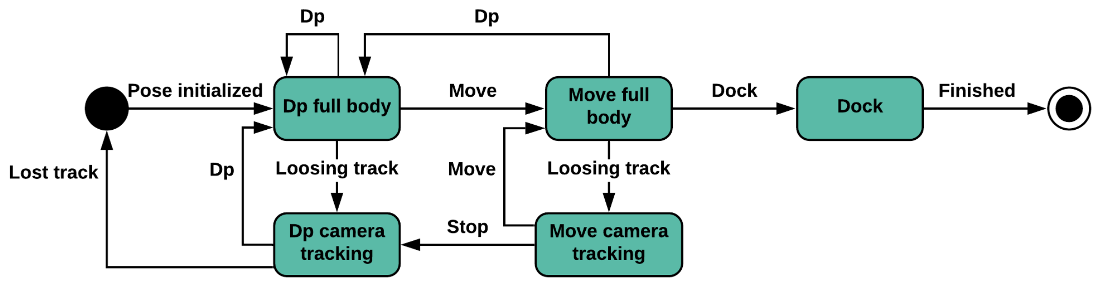


Figure 15.1: Missions specification control system originating from top-level system events, i.e the internal state of the SLAM and planning modules mapped to global events. Specified *camera tracking* missions imply that the USM module with an attached camera is oriented towards some setpoint while some other task is performed. *Stopping* is requested if the current objective is infeasible.

The top-level states *DP* and *move* are of main interest for the current control system design. These are reflected by the control system states *DP full body* and *move full body*. To demonstrate how the control system can aid the navigation module, a *loosing track* events are introduced. Path following could then be turned off for the link with an exteroceptive sensor, a camera in this case, such that the FoV could be directed towards previously seen environmental features. This could be done with the framework presented in [205], for example. However, this is not considered any further in this chapter, as 3D full body path following is the main objective.

Further, it is assumed that the SLAM module provide pose and velocity estimates, η_{Ib}, $\dot{\eta}_{Ib}$ respectively, for the base link. The path planning module provides a path, $\pi(\cdot)$, that is assumed to be kinematically feasible, section 12.6, for the USM. Joint angles are measured from magnetic angular joint encoders [2].

Finally, the control system is designed to compute desired body and joint velocities for a lower level control system to interpret. This was chosen to limit the scope of the thesis to allow for a rigorous kinematic control design. An overview of the above-mentioned input and output interfaces is displayed in fig. 15.2.

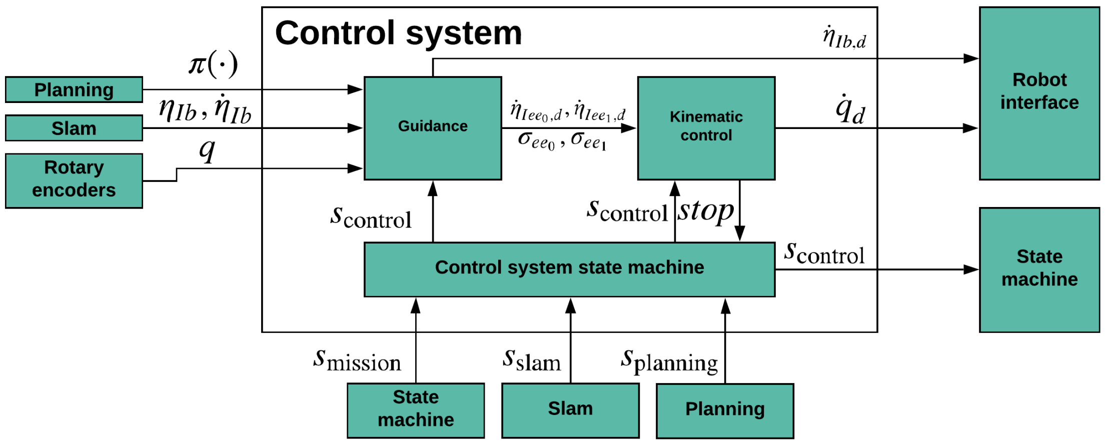


Figure 15.2: Control system interface towards SLAM, planning, state machine and the USM. All communication externally and internally are displayed.

15.2 Guidance

Path following along 3D curves firstly require a guidance law that computes a desired homogeneous transform from the robots current pose towards the path, thus guiding the robot in an appropriate direction. In the context of an articulated USM, it is required that multiple links follow the path to avoid collisions, see section 12.6, making it a set of homogeneous transforms. Finally, computation of initial linear and angular velocity references that can be used for any of these links is also considered in this section.

15.2.1 Performance Specifications

Standard performance measures for guidance algorithms, from a practical perspective, are:

- *Convergence*: Deviation between the desired path centric target pose and the robot pose.
- *Overshoot*: Measure of deviation after the robot pose cross the path after a transient approach period.

[2]Considering the Eelume robot

- *Velocity reference tracking*: Feasibility of the velocity reference.

Convergence and overshoot are prioritized in this project since the path has an attached *safe-distance*, see e.g. section 14.3, which constrain the allowed deviation from the path. The maximum allowable path deviation Δ_π, subsequently define the primary performance measure along with convergence. It is assumed that the USM manages to track the velocity requested by the guidance law, making the velocity reference tracking a given[3].

15.2.2 Existing Solutions

A kinematic path following scheme for wheeled robots was treated in [206], where the authors consider a strategy for projecting the position of the wheeled robot onto a desired geometric path. A path tangential Frenet frame is computed at this point, effectively representing the deviation between the robot and the path. The downside of this approach is the singularity that arise at the center of every osculating circle along the path [4].

The guidance framework from [206] was extended in [203] and [207] with a singularity-free path following scheme, in 2D and 3D respectively. This is done by introducing a propagation law for the path parametrization variable, i.e. a way to *push* the path tangential Frenet frame along the path. The paper [207] also demonstrate closed-loop stability properties of the guidance framework in cascade with an ISS backstepping controller for an under-actuated 3-DoF unmanned surface vessel (USV). Note that this framework works both for under- and fully-actuated robots.

The approach presented in [202] also propagate a Frenet frame along a desired 3D path, and shows that the base of an under-actuated underwater vehicle converge towards it with uniformly global asymptotic stability (UGAS) [5]. The control law is implemented in cascade with a feedback linearizing surge controller and a backstepping attitude controller to achieve closed-loop stability results. This approach also guarantee path convergence in the presence of unknown ocean currents.

In [208], it is shown that straight line path following can be achieved in 3D with the UGAS and uniformly local exponential stability (ULES) properties, using an integral line of sight (ILOS) approach. This approach can also ensure convergence in the presence of constant irrotational ocean currents.

These approaches suffice in the context of non-articulated robots, as the practical aspects relating to the above-mentioned performance measures are demonstrated in conjunction

[3]This is a superfluous assumption that does not hold in practice. However, robust dynamic control is not considered in this thesis, making the measure difficult to test.

[4][206] shows that an asymptotically stable (AS) tube can be constructed around the path which depends on the curvature.

[5]Given certain assumptions. The most important one affecting this work is that the algorithm is limited by the Euler angle singularity.

with theoretical stability proofs. However, an USM require multiple links to follow the desired path. The following section will describe the guidance algorithm that will be used in this project and how it extends to guide multiple links.

15.2.3 Solution

Guidance law

In this work, multiple path-tangential Frenet frames will be defined to guide a set of links towards a path. To develop such a framework the body frame, virtually any desired point on the USM, is defined.

Definition 15.1. The body frame, F_b is the root of the kinematic tree which branch out to both sides of the USM and connects to the inertial frame, T_I^b.

Remark 15.2.1. Even though the body frame can be located at any position, it is chosen as the the longest link on the USM. This point defines a moving base with two attached manipulators.

In the guidance context, the body frame is *virtually connected* to a path-tangential Frenet frame, F_{f_b}, such that the deviation between the frames can be represented effectively[6],

$$\eta^I_{f_b/b} = \begin{bmatrix} x_{f_b/b} & y_{f_b/b} & z_{f_b/b} & \phi_{f_b/b} & \theta_{f_b/b} & \psi_{f_b/b} \end{bmatrix} \tag{15.1}$$

At this point, the 3D guidance algorithms from e.g. [202] or [203] from section 15.2.2 could be used to control the body frame. The algorithm from [202] is chosen, implying that the maneuvering task [209] is solved for the base frame, with the possibility to add current compensation. For completeness the guidance law is presented next. The guidance angles are

$$\begin{aligned} \psi_{b/f_b,d} &= \arctan\left(\frac{y_{f_b/b}}{\sqrt{\Delta^2 + x^2_{f_b/b} + z^2_{f_b/b}}}\right) \\ \theta_{b/f_b,d} &= -\arctan\left(\frac{z_{f_b/b}}{\sqrt{\Delta^2 + y^2_{f_b/b}}}\right) \end{aligned} \tag{15.2}$$

where Δ is the lookahead distance. The propagation law is

[6] $y_{f_b/b}$ and $z_{f_b/b}$ are known as the base frame cross-track and vertical errors respectively

$$
\begin{aligned}
\dot{\varpi} = U \cdot &\frac{\sqrt{\Delta^2 + y_{f_b/b}^2}}{\sqrt{\Delta^2 + y_{f_b/b}^2 + z_{f_b/b}^2}} \cdot \frac{\sqrt{\Delta^2 + x_{f_b/b}^2 + z_{f_b/b}^2}}{\sqrt{\Delta^2 + x_{f_b/b}^2 + y_{f_b/b}^2 + z_{f_b/b}^2}} \\
&- U \cdot \frac{x_{f_b/b}}{\sqrt{\Delta^2 + x_{f_b/b}^2 + y_{f_b/b}^2 + z_{f_b/b}^2}}
\end{aligned}
\tag{15.3}
$$

where U is the desired body frame speed, not accounting for currents as in [202].

To further ensure that the maneuvering task is solved for multiple links, the set of guided links and their corresponding path tangential Frenet frames are defined, see fig. 15.3.

Definition 15.2. The full set of guided links constitutes the two end-effectors and the link attached to the base frame, $\{F_b, F_{ee_0}, F_{ee_1}\}$[7]. Their path tangential Frenet frames are $\{F_{f_b}, F_{f_{ee_0}}, F_{f_{ee_1}}\}$, see fig. 15.3. Their progress along the path are represented by $\{\pi(\varpi_b), \pi(\varpi_{ee_0}), \pi(\varpi_{ee_1})\}$, respectively.

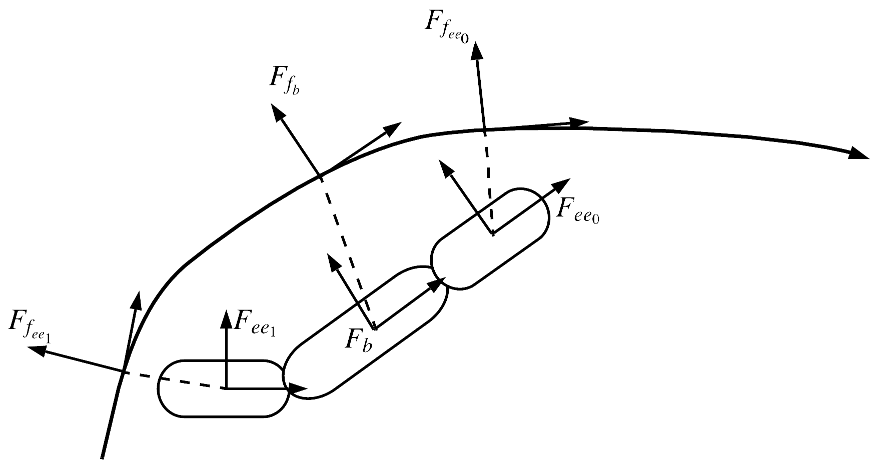


Figure 15.3: The robot centric coordinate systems $\{F_b, F_{ee_0}, F_{ee_1}\}$ and their associated path tangential Frenet frames. The dashed line represent the projection from the end-effector frames onto the curve.

Three instances are chosen as this is sufficient to ensure that the USM adheres to the path given that the assumptions from section 12.6 holds (more on this in section 15.3). To reduce complexity, ϖ_{ee_0} and ϖ_{ee_1} are found by projection, see appendix E. It is consequently prone to ambiguities, see e.g. [206], as illustrated in fig. 15.4. This is resolved by saturating $|\varpi_b - \varpi_{ee_0}|$ and $|\varpi_b - \varpi_{ee_1}|$, such that F_{ee_0} and F_{ee_1} are constrained to F_b.

Finally, the output from the guidance algorithm are represented as a set of homogeneous transformations from the USM towards the path, $\{\mathbf{T}_b^{f_b}, \mathbf{T}_{ee_0}^{f_{ee_0}}, \mathbf{T}_{ee_1}^{f_{ee_1}}\}$. Next, the velocity

[7]The end-effector denoted the 0 link in the link enumeration has the same name here

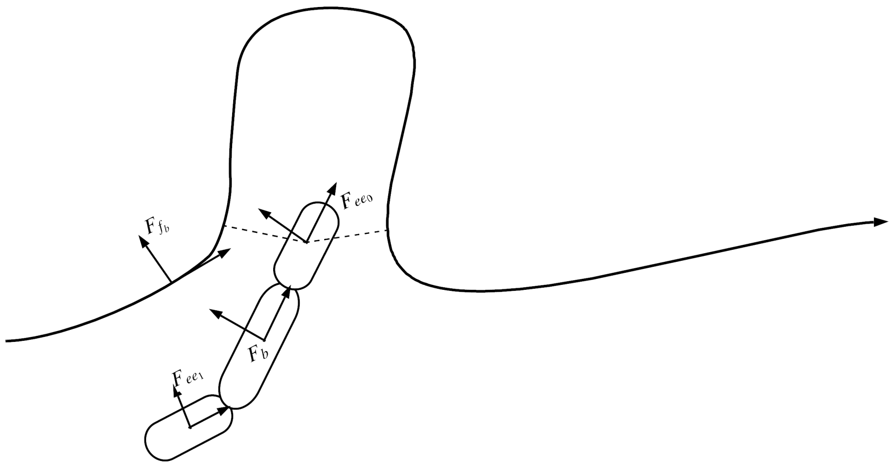


Figure 15.4: A USM encapsulated by a path. The dashed lines represent the projection ambiguity that may arise in such situations.

reference generation will be presented before cascaded stability results are presented in the kinematic control section.

Velocity reference generation

Velocity generation includes computing a desired linear and angular velocity. In general, the linear velocity reference is the desired velocity in the guidance direction

$$\dot{\eta}_{f_i/i,1,d} = \mathbf{R}_{i,d}^{f_i} \begin{bmatrix} u_{b,d} \\ 0 \\ 0 \end{bmatrix} \tag{15.4}$$

where $i \in \{ee_0, b, ee_1\}$ and $u_{b/b,d}$ is the desired body frame surge velocity. The angular velocity reference, on the other hand, is a function of the path curvature, the guidance angles and the angular velocity. In the two-dimensional case the yaw rate reference is defined as

$$r_{i,d} = \kappa v_{f_i} + k_{p,\psi}\psi_{f_i/i,d} + \dot{\psi}_{f_i/i,d} \tag{15.5}$$

see e.g. [15], ensuring that $\psi_{i/I} \to \psi_{f_i/I,d}$. However, a general three-dimensional curve with a path-tangential Frenet formulas inherently promote *pitch-free* angular motion

$$\vec{\omega} = \kappa\vec{b} + \tau\vec{t} \tag{15.6}$$

8

[8]This vector of angular velocity is commonly referred to as the Darboux vector [210]

with $\vec{b}, \vec{t}$ being the binormal and tangent vectors at a certain point along the path. This can be seen from the Frenet-Serret formulas which includes both curvature and torsion

$$\begin{bmatrix} \vec{t}' & \vec{n}' & \vec{b}' \end{bmatrix} = \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \cdot \begin{bmatrix} \vec{t} & \vec{n} & \vec{b} \end{bmatrix} \tag{15.7}$$

where κ is the path curvature and τ is the relative torsion of the path — for simplicity named torsion in this thesis. From eq. (15.6) and eq. (15.7) it is evident that the path tangential Frenet frame only rotates about its tangential axis due to torsion, τ, and about its binormal axis due to curvature, κ.

To make full-body convergence possible, the set of Frenet frames need to be relatively oriented such that they are reachable given the joint configuration of the USM. Consequently, hardware related assumptions are made to tailor the angular velocity references to a particular configuration.

Assumption 15.2.1. *The USM has only revolute joints providing yaw and pitch motion.*

Assumption 15.2.2. *Roll is actively stabilized by low-level controllers.*

Assumption 15.2.2 implies that a *rotation-minimizing adapted frame* (RMAF) have to be constructed, see e.g. [210], to provide a reference frame constrained by no rolling motion. This can be achieved by applying a rotation to the Frenet frames about their tangential axes, $\vec{t}$[9].

$$\begin{bmatrix} \vec{t} \\ \vec{u} \\ \vec{v} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi_{f_i}) & \sin(\phi_{f_i}) \\ 0 & -\sin(\phi_{f_i}) & \cos(\phi_{f_i})\cdot \end{bmatrix} \cdot \begin{bmatrix} \vec{t} \\ \vec{n} \\ \vec{b} \end{bmatrix} \tag{15.8}$$

where ϕ_{f_i} is the rotation of the Frenet frame of interest around its own tangential axis. Differentiating eq. (15.8) is necessary to obtain an expression for a new coordinate frame that promotes *roll-free* angular motion.

$$\begin{aligned} \begin{bmatrix} \vec{t}' \\ \vec{u}' \\ \vec{v}' \end{bmatrix} &= \frac{d}{d\varpi}(\mathbf{R}_{x,\phi_{f_i}})^T \cdot \begin{bmatrix} \vec{t} \\ \vec{n} \\ \vec{b} \end{bmatrix} + \mathbf{R}_{x,\phi_{f_i}}^T \cdot \begin{bmatrix} \vec{t}' \\ \vec{n}' \\ \vec{b}' \end{bmatrix} \\ &= -\mathrm{S}(\vec{\omega}_{\phi_{f_i}}) \cdot \mathbf{R}_{x,\phi_{f_i}}^T \cdot \begin{bmatrix} \vec{t} \\ \vec{n} \\ \vec{b} \end{bmatrix} + \mathbf{R}_{x,\phi_{f_i}}^T \cdot \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \cdot \begin{bmatrix} \vec{t} \\ \vec{n} \\ \vec{b} \end{bmatrix} \end{aligned} \tag{15.9}$$

[9]See e.g. [211]

where eq. (15.8) is written in matrix form to simplify calculations, $\vec{\omega}_{\phi_{f_i}} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T \omega_{\phi_{f_i}}$, the Frenet formulas are inserted for $\begin{bmatrix} \vec{t}' & \vec{n}' & \vec{b}' \end{bmatrix}^T$ and ϖ is the curvilinear progress along the path. Further, substituting eq. (15.8) into eq. (15.9) gives

$$\begin{aligned}
\begin{bmatrix} \vec{t}' \\ \vec{u}' \\ \vec{v}' \end{bmatrix} &= -\mathbf{S}(\vec{\omega}_{\phi_{f_i}}) \cdot \mathbf{R}_{x,\phi_{f_i}}^T \cdot \mathbf{R}_{x,\phi_{f_i}} \cdot \begin{bmatrix} \vec{t} \\ \vec{u} \\ \vec{v} \end{bmatrix} + \mathbf{R}_{x,\phi_{f_i}}^T \cdot \begin{bmatrix} 0 & \kappa & 0 \\ -\kappa & 0 & \tau \\ 0 & -\tau & 0 \end{bmatrix} \cdot \mathbf{R}_{x,\phi_{f_i}} \cdot \begin{bmatrix} \vec{t} \\ \vec{u} \\ \vec{v} \end{bmatrix} \\
&= \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -\omega_{\phi_{f_i}} \\ 0 & \omega_{\phi_{f_i}} & 0 \end{bmatrix} + \begin{bmatrix} 0 & \kappa\cos(\phi_{f_i}) & -\kappa\sin(\phi_{f_i}) \\ -\kappa\cos(\phi_{f_i}) & 0 & \tau \\ \kappa\sin(\phi_{f_i}) & -\tau & 0 \end{bmatrix} \right) \cdot \begin{bmatrix} \vec{t} \\ \vec{u} \\ \vec{v} \end{bmatrix} \\
&= \begin{bmatrix} 0 & \kappa\cos(\phi_{f_i}) & -\kappa\sin(\phi_{f_i}) \\ -\kappa\cos(\phi_{f_i}) & 0 & \tau - \omega_{\phi_{f_i}} \\ \kappa\sin(\phi_{f_i}) & -\tau + \omega_{\phi_{f_i}} & 0 \end{bmatrix} \cdot \begin{bmatrix} \vec{t} \\ \vec{u} \\ \vec{v} \end{bmatrix}
\end{aligned} \tag{15.10}$$

The final expression in eq. (15.10) shows that an appropriate choice of $\omega_{\phi_{f_i}}$ cancels the torsion of the curve, ee e.g. [210] for a possible choice. Consequently, rolling motion is cancelled out in the coordinate frame $F_{f'} = (\vec{t}, \vec{u}, \vec{v})$

$$\vec{\omega} = \kappa \cdot (\sin(\phi_{f_i})\vec{u} + \cos(\phi_{f_i})\vec{v}) \tag{15.11}$$

In 2D, the rotation rate between the angle $\psi_{i/I} - \psi_{f_i'/I}$ is $\psi_{i/I} - \psi_{f_i'/I} = \kappa\dot{\varpi}$, see e.g. [15]. Extending this observation two 3D with respect to $F_{f'}$ then gives the dynamical system

$$\begin{aligned}
\dot{\psi}_{i/I} - \dot{\psi}_{f_i'/I} &= \kappa\cos(\phi_{f_i})v_{f_i} \\
\dot{\theta}_{i/I} - \dot{\theta}_{f_i'/I} &= \kappa\sin(\phi_{f_i})v_{f_i}
\end{aligned} \tag{15.12}$$

Yaw and pitch rate references can now be defined with respect to $F_{f'}$ to ensure convergence and stability of eq. (15.12)

$$\begin{aligned}
r_{i,d} &= \kappa\cos(\phi_{f_i})v_{f_i} + k_{p,\psi}\psi_{f_i'/i,d} + \dot{\psi}_{i/I,d} \\
q_{i,d} &= \kappa\sin(\phi_{f_i})v_{f_i} + k_{p,\theta}\theta_{f_i'/i,d} + \dot{\theta}_{i/I,d}
\end{aligned} \tag{15.13}$$

Note that for practical purposes $k_{p,\psi} = k_{p,\theta}$ and $k_{d,\psi} = k_{d,\theta}$ based on rotary joint and thruster symmetry on a typical USM. The following lemma extends the results in [15] to 3D.

Lemma 15.2.1. *Consider the dynamical system in eq.* (15.12) *where* $\dot{\psi}_{f_i'/I}, \dot{\theta}_{f_i'/I}$ *are the angular velocities of the Frenet frame. Then the reference angles (15.13) make the dynamic system*

(15.12) converge.

Proof. Stability and convergence will be shown for the yaw rate reference. Because of the symmetry, the proof for pitch angle convergence is equivalent. Insertion of (15.13) into (15.12) gives

$$\begin{aligned} \kappa\cos(\phi_{f_i})v_{f_i} + k_{p,\psi}\psi_{f_i'/i,d} + \dot{\psi}_{f_i'/i,d} - \dot{\psi}_{f_i'/I} &= \kappa\cos(\phi_{f_i})v_{f_i} \\ \dot{\tilde{\psi}} + k_{p,\psi}\psi_{f_i'/i,d} &= 0 \end{aligned} \tag{15.14}$$

Consequently, $\psi_{i/I} \rightarrow \psi_{f_i'/I}$ ensuring convergence. As mentioned above, the same result holds for the pitch angles through the same procedure. □

For clarification, the control laws in eq. (15.2), eq. (15.3) and eq. (15.13) is applied to the base link. This ensures convergence for curved paths provided sufficiently large proportional velocity reference gains, $k_{p\psi}$, $k_{p,\theta}$, and a sufficiently small lookahead distance, Δ_b. Results exploring how this tuning parameters are chosen to obtain the performance specifications will be presented in section 15.4. The end-effectors path-tangential Frenet frames are found by projecting the end-effector link position onto path. This point of projection is then constrained to an interval a fixed distance from the origin of the Frenet frame associated with the base link. How these projected points are used to ensure full-body path convergence is explored in the subsequent section.

15.3 Kinematic Control

15.3.1 Performance Specifications

The kinematic controller can be used for a variety of tasks including collision avoidance, joint constraints, ensuring FoV and path following. However, only the latter will be a focus area in this section as collision avoidance is covered by the path planning module and the FoV are implicitly guaranteed during path following along a pipe. The goal for the kinematic controller is therefore to ensure that the end-effectors converge to the path given the path errors specified from the guidance output in subsection 15.2.3. Formally, it is expected that the cross-track and vertical errors for both end-effectors vanish

$$\begin{aligned}
&\lim_{t\to\infty} y_{ee_0/f_{ee_0}}(t) \to 0\\
&\lim_{t\to\infty} z_{ee_0/f_{ee_0}}(t) \to 0\\
&\lim_{t\to\infty} y_{ee_1/f_{ee_1}}(t) \to 0\\
&\lim_{t\to\infty} z_{ee_1/f_{ee_1}}(t) \to 0
\end{aligned} \tag{15.15}$$

The values in 15.15 will be used as performance measures as they specify the relative closeness between the end-effectors and the path.

15.3.2 Existing Solutions

In [212], an *operational space* formulation for robot manipulator control was developed. This framework makes it possible to specify desired end-effector setpoints in 3D space that a robot manipulator can reach. This end-effector setpoint is formally denoted as a *task*. The methodology was further developed with a framework called *closed loop inverse kinematics* (CLIK), see e.g. [19]. The CLIK algorithm that is presented is exponentially stable, assuming independent tasks. Since only two, kinematically independent tasks are used in this work, this framework can be used.

Kinematic control have been used extensively both for tracking control of USMs and more generally for tracking control for robot manipulators. In particular, kinematic control was used in [16] to generate velocity references for straight line path following. However, only 2D straight line path following was considered. Straight line path following in 3D was considered in [204] with references generate in a similar manner as in [16]. The next section describes how the above-mentioned CLIK algorithm can be modified to fit the position tracking problem for curved 3D paths.

15.3.3 Solution

In this work, position tracking tasks will be defined for the end-effectors to ensure full body path following. As these tasks are computed with the projective method described in section 15.2.3, they are not *reachable* by default. Figure 15.5 demonstrate how consecutive projection steps bends the USM towards the path, never reaching it. Note that, in general, a curved path either bends towards or away from the USM (or a combination), and that the USM end-effectors will move towards them and saturate[10].

To ensure that the end-effectors track the path, convergence and stability will be proven

[10]An observation at this point is that the USM is able to extend further towards the path if the number of links between base and end-effector frames increase.

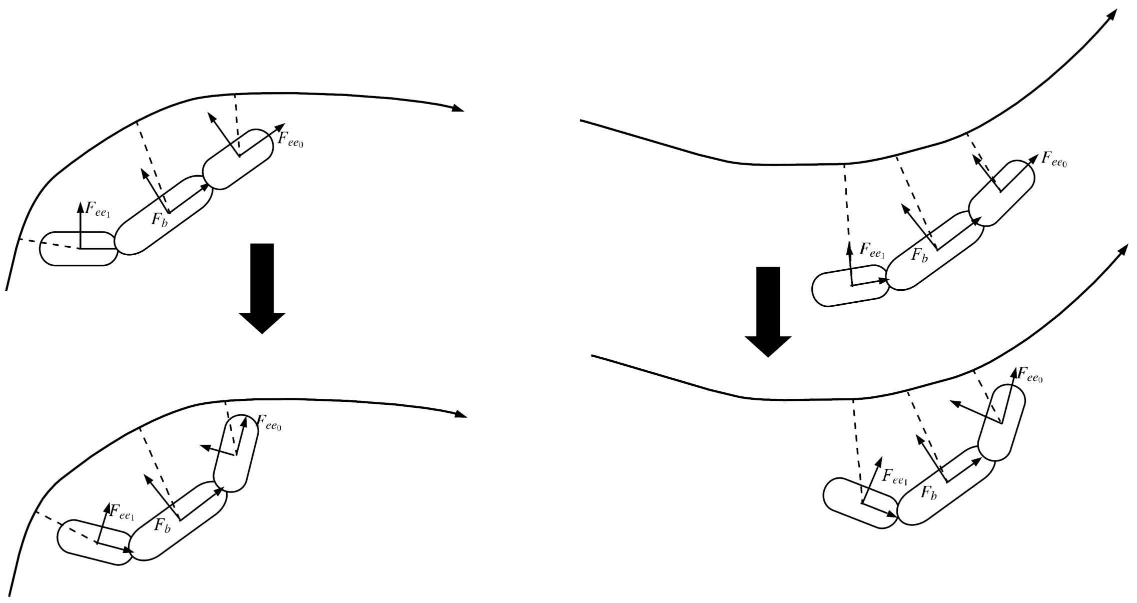


Figure 15.5: The convergence of unconstrained consecutive projection steps for paths bending towards and away from the USM. The base frame are assumed at rest and the large black errors denote how the system naturally would bend in the given cases. The dashed line represent the error between the respective path-tangential Frenet frames and their guided links.

given end-effector position tasks. The tasks are defined as an objective for the end-effector to achieve, namely touch the path at some point. Consequently,

$$\begin{aligned} \sigma_{ee_0} &= \eta_{ee_0/I,1} \\ \sigma_{ee_1} &= \eta_{ee_1/I,1} \end{aligned} \tag{15.16}$$

is chosen for the front and rear end-effectors. By differentiating the tasks with respect to time, the mapping between the joint and task velocities appear, see e.g. [19], $\frac{d\sigma}{dt} = \frac{d\sigma}{dq}\frac{dq}{dt}$. For the tasks in eq. (15.16), this mapping is denoted as position Jacobian[11], $\mathbf{J}_{i,\text{pos}},\ i \in \{ee_0, ee_1\}$

$$\begin{aligned} \dot{\sigma}_{ee_0} &= \mathbf{J}_{ee0,\text{pos}}\dot{q} \\ \dot{\sigma}_{ee_1} &= \mathbf{J}_{ee1,\text{pos}}\dot{q} \end{aligned} \tag{15.17}$$

The *operational space*[12] position errors $\eta_{f_{ee_0}/ee_0,1}, \eta_{f_{ee_1}/ee_1,1}$, define the task errors — displayed by the dashed lines in fig. 15.5. These are given by

[11]How to compute this Jacobian is intentionally omitted as it does not contribute to the understanding of the developed approach. See e.g. [14] for a comprehensive description of derivation, use cases and implementation.

[12]Meaning that the task is specified in the space in which the manipulator task is specified, see e.g. [19]

$$\tilde{\sigma}_i = \eta_{f_i/i} = \mathbf{R}_I^{i\top}(\pi(\varpi_i) - \eta_{i/I,1}) \tag{15.18}$$

where $\pi(\cdot)$ is the path and ϖ the curvilinear progress along it. From [19], the CLIK algorithm is presented as

$$\dot{q}_{i,\text{des}} = \mathbf{J}_{i,\text{pos}}^{\dagger}(\dot{\sigma}_{i,d} + \mathbf{\Lambda}_i\tilde{\sigma}_i) \tag{15.19}$$

where $\mathbf{\Lambda}_i \succ 0$ is a gain matrix, $\dot{\sigma}_{i,d}$ is the desired task velocity and $J_{i,\text{pos}}^{\dagger}$ is the Moore-Penrose inverse of $J_{i,\text{pos}}$ — commonly called the *pseudoinverse.* The task error gain and the desired task velocity are determined to ensure stability of the task error dynamics, $\dot{\tilde{\sigma}}_i$.

$$\begin{aligned}\dot{\tilde{\sigma}}_i &= \mathbf{R}_I^{i\top}(\dot{\pi}(\varpi_i) - \dot{\eta}_{i/I,1}) - \mathbf{S}(\vec{\omega}_{i/I})\mathbf{R}_I^{i\top}(\pi(\varpi_i) - \eta_{i/I,1}) \\ &= \mathbf{R}_I^{i\top}(\dot{\pi}(\varpi_i) - \mathbf{J}_{i,\text{pos}}\dot{q}_i) - \mathbf{S}(\vec{\omega}_{i/I})\mathbf{R}_I^{i\top}(\pi(\varpi_i) - \eta_{i/I,1}) \\ &= \mathbf{R}_I^{i\top}(\dot{\pi}(\varpi_i) - \mathbf{J}_{i,\text{pos}}\dot{q}_{i,\text{des}}) - \mathbf{S}(\vec{\omega}_{i/I})\mathbf{R}_I^{i\top}(\pi(\varpi_i) - \eta_{i/I,1})\end{aligned} \tag{15.20}$$

Note that the final expression in eq. (15.20) assumes that the joint velocities is a control variable and that $\dot{q}_i \approx \dot{q}_{i,\text{des}}$. This assumption holds given a sufficiently fast dynamic control loop[13] — or sufficiently slow kinematic control loop — which will be discussed further from an experimental standpoint, section 15.4. A control system design that assumes the opposite is out of the scope of this thesis and will therefore not be considered.

The next sections proves boundedness of the task errors, eq. (15.18), for a particular choice of $\dot{\sigma}_{i,d}$ in eq. (15.19). Stability and convergence of the cascade in fig. 15.2 is also shown.

Boundedness

Proving boundedness of the task errors is necessary to ensure that the end-effector joint angles do not diverge in the presence of *disturbances* introduced by a moving base and a curved path. Meaning, the velocity of the base link relative to the path will not necessarily equal the end-effector velocity relative to the path. Thus, the base frame moving introduce a path offset felt by the end-effectors. The following lemma establish the boundedness of the task errors when a LOS vector — found from the angles in eq. (15.2) — is used to determine the desired task velocities

Lemma 15.3.1. *Let the task error dynamics from eq.* (15.20) *be controlled by the joint desired joint velocities from eq.* (15.19). *A choice of the desired task velocities is*

[13]See e.g. [195] for a description of this concept in cascaded control loops

$$\begin{aligned}\dot{\sigma}_{i,d} &= \mathbf{R}_I^i \vec{v}_{i,1,d} \\ &= \begin{bmatrix} \vec{t}_{i,d} & \vec{n}_{i,d} & \vec{b}_{i,d/i} \end{bmatrix} \begin{bmatrix} u_{i,d} \\ 0 \\ 0 \end{bmatrix} \\ &= \vec{t}^I_{i,d} u_{i,d}\end{aligned} \tag{15.21}$$

where $\vec{t}^I_{i,d}$ is the velocity direction with respect to F_i in F_I and $u_{i/i,d}$ its surge magnitude. With this choice of control, and the Lyapunov function candidate $\mathbf{V}_{\tilde{\sigma}_i} = \frac{1}{2}\tilde{\sigma}_i^T\tilde{\sigma}_i$, the tracking task errors are uniformly ultimately bounded (UUB) with $||\tilde{\sigma}_i|| \geq \frac{2U_{max}}{\theta} > 0$, where $0 < \theta < \lambda_{min}(\mathbf{\Lambda}_i)$.

Proof. Differentiating the Lyapunov function candidate, $\mathbf{V}_{\tilde{\sigma}_i} = \frac{1}{2}\tilde{\sigma}_i^T\tilde{\sigma}_i$, and inserting eq. (15.20), and rotating the task error in eq. (15.19) to the inertial frame, gives

$$\begin{aligned}\dot{\mathbf{V}}_{\tilde{\sigma}_i} &= \tilde{\sigma}_i^T \dot{\tilde{\sigma}}_i \\ &= \tilde{\sigma}_i^T \Big(\mathbf{R}_I^{i\top} (\dot{\pi}(\varpi_i) - \mathbf{J}_{i,\text{pos}} \dot{q}_{i,\text{des}}) - \mathbf{S}(\vec{\omega}_{i/I}) \mathbf{R}_I^{i\top} (\pi(\varpi_i) - \eta_{i/I,1}) \Big) \\ &= \tilde{\sigma}_i^T \Big(\mathbf{R}_I^{i\top} (\dot{\pi}(\varpi_i) - \mathbf{J}_{i,\text{pos}} \mathbf{J}^{\dagger}_{i,\text{pos}} (\dot{\sigma}_{i,d} + \mathbf{\Lambda}_i \mathbf{R}_I^i \tilde{\sigma}_i)) - \mathbf{S}(\vec{\omega}_{i/I}) \tilde{\sigma}_i \Big) \\ &= \tilde{\sigma}_i^T \mathbf{R}_I^{i\top} \Big(\dot{\pi}(\varpi_i) - (\mathbf{R}_I^i \vec{v}_{i,d} + \mathbf{\Lambda}_i \mathbf{R}_I^i \tilde{\sigma}_i) \Big) \\ &= -\tilde{\sigma}_i^T \mathbf{\Lambda}_i \tilde{\sigma}_i + \tilde{\sigma}_i^T \mathbf{R}_I^{i\top} \Big(\dot{\pi}(\varpi_i) - \mathbf{R}_I^i \vec{v}_{i,d} \Big)\end{aligned} \tag{15.22}$$

where the fourth inequality use that $x^T\mathrm{S}(\cdot)x \equiv 0$ for any skew-symmetric matrix $\mathrm{S}(\cdot)$. from the chain rule, it follows that $\dot{\pi}(\varpi_i) = \frac{d\pi(\varpi_i)}{d\varpi_i}\dot{\varpi}_i = \vec{t}^I_{f_i/I}(\varpi_i)\dot{\varpi}_i$ where $\vec{t}^I_{f_i/I}(\varpi_i)$ is the unit tangent vector and $\dot{\varpi}_i$ is the desired base frame surge speed. Consequently, the derivative of the Lyapunov function can be bounded from above

$$\begin{aligned}\dot{\mathbf{V}}_{\tilde{\sigma}_i} &\leq -\lambda_{\min}(\mathbf{\Lambda}_i)\, ||\tilde{\sigma}_i||_2^2 + ||\tilde{\sigma}_i||_2 \left\| \vec{t}^I_{f_i/I}(\varpi_i) u_{b,d} - \vec{t}^I_{i,d} u_{i,d} \right\|_2 \\ &\leq -\lambda_{\min}(\mathbf{\Lambda}_i)\, ||\tilde{\sigma}_i||_2^2 + \theta\, ||\tilde{\sigma}_i||_2^2 - \theta\, ||\tilde{\sigma}_i||_2^2 + ||\tilde{\sigma}_i||_2 \left\| \vec{t}^I_{f_i/I}(\varpi_i) - \vec{t}^I_{i,d} \right\|_2 \max(u_{b,d}, u_{i,d}) \\ &\leq -(\lambda_{\min}(\mathbf{\Lambda}_i) - \theta)\, ||\tilde{\sigma}_i||_2^2, \ \forall\, ||\tilde{\sigma}_i||_2 \geq \frac{\left\| \vec{t}^I_{f_i/I}(\varpi_i) - \vec{t}^I_{i,d} \right\|_2 \max(u_{b,d}, u_{i,d})}{\theta}\end{aligned} \tag{15.23}$$

where $\lambda_{\min}(\mathbf{\Lambda}_i)$ is the smallest eigenvalue of $\mathbf{\Lambda}_i$, and $0 < \theta < \lambda_{\min}(\mathbf{\Lambda}_i)$. Since $\vec{t}^I_{f_i/I}(\varpi_i), \vec{t}^I_{i,d}$ are unit vectors, their relative norm is bounded from above by the case were they stand parallel in opposite directions, $\left\| \vec{t}^I_{f_i/I}(\varpi_i) - \vec{t}^I_{i,d} \right\|_2 \leq 2$. Finally, the desired surge velocity can be chosen implying that it is naturally bounded from above $\max(u_{b,d}, u_{i,d}) \leq U_{\max}$. It follows that the result holds for all $\|\tilde{\sigma}_i\|_2 \geq \frac{2U_{\max}}{\theta} > 0$. According to theorem 4.18 in [83], the tracking tasks are uniformly ultimately bounded.

□

By lemma 15.3.1, the task errors remain ultimately bounded within some ball which can be found from theorem 4.18 in [83]. Note that this bound is overly conservative, as the USM typically moves either towards the path or along it. To ensure this, however, convergence of the task towards this path have to be proven. This requires the desired velocity tangent to converge to the path tangent, $\left\|\vec{t}^{I}_{f_i/I} - \vec{t}^{I}_{i,d}\right\|_2 \longrightarrow 0$, as the bound in lemma 15.3.1 then shrinks to 0.

An observation from lemma 15.3.1, is that the kinematic control law introduce two tuning parameters to control the convergence rate. The velocity direction is scaled by a lookahead distance, henceforth denoted Δ_i, and the magnitude of the velocity can be chosen freely. Consequently, angular rate can be limited, by tuning of the above-mentioned parameters along with the gain $\mathbf{\Lambda}_i$. This is a desirable feature considering the effect of increased yaw rate on the SLAM module, see part III. Note, however, that the angular velocity of the end-effector is a sum of the velocity imposed by joint velocities and the velocity of the base link. The two therefore have to be tuned with the same goal in mind. A thorough discussion on this topic is covered in section 15.4.

Cascade stability

In section 15.2 it was established that the the base link tracks a path-tangential Frenet frame which propagates along a path. Since the end-effectors move when the base link moves, the projection from the end-effectors onto the curve will depend on the base link velocity. Additionally, as the curve has a curvature profile the projection will also vary irrespective of the base frame velocity, see fig. 15.5 for an illustration. To ensure convergence of the task errors, the cascade of the controllers in fig. 15.2 have to satisfy the criteria of lemma 2.1 in [213]. To prove this, firstly stability of the end-effector error dynamics have to be shown given no influence from the base link.

Lemma 15.3.2. *Consider the position task error dynamics from eq.* (15.20)*. Under influence of the control law eq.* (15.21)*, The unperturbed system without any base link velocity is UGAS.*

Proof. Firstly, eq. (15.20) can be re-written as

$$\begin{aligned}\dot{\tilde{\sigma}}_i &= \mathbf{R}_I^{i\top}(\dot{\pi}(\varpi_i) - \mathbf{J}_{i,\text{pos}}\dot{q}_{i,\text{des}}) - \mathbf{S}(\vec{\omega}_{i/I})\mathbf{R}_I^{i\top}(\pi(\varpi_i) - \eta_{i/I,1}) \\ &= \mathbf{R}_I^{i\top}(\dot{\pi}(\varpi_i) - (\dot{\sigma}_{i,d} + \Lambda_i \mathbf{R}_I^i \tilde{\sigma}_i)) - \mathbf{S}(\vec{\omega}_{i/I})\tilde{\sigma}_i \\ &= -(\Lambda_i + \mathbf{S}(\vec{\omega}_{i/I}))\tilde{\sigma}_i - \dot{\sigma}_{i,d} + \mathbf{R}_I^{i^T}(\dot{\pi}(\varpi_i)) \\ &= -(\Lambda_i + \mathbf{S}(\vec{\omega}_{i/I}))\tilde{\sigma}_i - \vec{t}_{i,d}^{I} u_{i,d} + \mathbf{R}_I^{i^T}(\vec{t}_{f_i/I,d} u_{b,d})\end{aligned} \tag{15.24}$$

where the notation follows that of the derivation in lemma 15.3.1. The assumption of a base link with zero velocities translates to $u_{b,d} = 0$ and $\vec{\omega}_{b/I} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. The error dynamics in eq. (15.24) unperturbed can be written as

$$\dot{\tilde{\sigma}}_i = -(\Lambda_i + \mathbf{S}(\vec{\omega}_{i/b}))\tilde{\sigma}_i - \vec{t}_{i,d}^{I} u_{i,d} \tag{15.25}$$

By introducing same Lyapunov function candidate, $\mathbf{V}_{\tilde{\sigma}_i} = \frac{1}{2}\tilde{\sigma}_i^T \tilde{\sigma}_i$ in lemma 15.3.1 the following expression is obtained

$$\begin{aligned}\dot{\mathbf{V}}_{\tilde{\sigma}_i} &= -\tilde{\sigma}_i^T \Lambda_i \tilde{\sigma}_i - \tilde{\sigma}^T \vec{t}_{i,d}^{I} u_{i,d} \\ &\le -\tilde{\sigma}_i^T \Lambda_i \tilde{\sigma}_i - \left\|\tilde{\sigma}^T\right\|_2 u_{i,\min}\end{aligned} \tag{15.26}$$

where $u_{i,\min}$. Under the assumption that the task is feasible — that the end-effector actually can reach the path — the criteria of theorem 4.9 in [83] is fulfilled such that the unperturbed system, eq. (15.25), is UGAS. □

Note that lemma 15.3.2 assumes a non-zero velocity. The direction of the velocity is uniquely determined by $\vec{t}_{i,d}^{I}$ implying that the result holds for end-effectors moving in any direction satisfying the hardware assumptions mentioned in 15.2.3. At this point, lemma 2.1 in [213] can be applied to obtain a cascaded stability result.

Consider the cascaded guidance and kinematic control system presented in fig. 15.2, which is intended to solve the maneuvering problem from [209] and ensure end-effector convergence in the sense of eq. (15.15). Further assume that the criteria of lemmas 15.2.1, 15.3.1, 15.3.2, and that of the guidance algorithm presented in [202], are satisfied. Consequently, by application of lemma 2.1 from [213], the cascaded system is UGAS:

$$\text{UGAS} + \text{UGB} + \text{UGAS} = \text{UGAS} \tag{15.27}$$

An implication of the above-mentioned result is that the guidance and kinematic controllers can be tuned separately. This fact is exploited in section 15.4.

15.4 Simulation

To ensure re-usability, readability and portability (in case a conversion from ROS to another messaging system is desirable at a later point), the control system implementation is grounded in an object-oriented methodology. The control system itself is implemented as a C++ library included in a ROS wrapper to allow communication with the rest of the autonomous system. The simulator avoids using any external libraries except for Eigen, [214], an open source linear algebra library.

15.4.1 Kinematics

The kinematics in the simulations are modelled using differential kinematics and evaluated using Euler integration, see e.g. [17],

$$
\begin{aligned}
p_{b/I}[k+1] &= p_{b/I}[k] + h \cdot \vec{v}_{b/I} \\
\mathbf{R}_I^b[k+1] &= \mathbf{R}_I^b[k] + h \cdot (\mathbf{R}_I^b[k] \cdot \mathrm{S}(\vec{\omega}_{b/I})) \\
q[k+1] &= q[k] + h \cdot dq
\end{aligned}
\tag{15.28}
$$

where h is the step length which is chosen to be $h = 0.001\,\mathrm{ms}$ and the square brackets $[\cdot]$ denote a discrete time state space formulation, see e.g. [215]. To ensure some attachment to reality, the joint velocities are saturated according to the specifications of the joint motors on the Eelume robot, $dq < 1.5\,\mathrm{rad\,s^{-1}}$.

Simulations are carried out to verify the correctness of the implementation and evaluate algorithmic choices specified in sections 15.2 and 15.3. Simulation results regarding the most significant design choices will be outlined subsequently. The factors determining the significance of the choices are whether they have a considerable effect on the performance goals specified in section 15.2.1 and section 15.3.1.

15.4.2 Tests

Simulation experiments are carried out to provide an ideal base-line for practical experiments. Consequently, a set of benchmark tests have to be performed with appropriate performance measures. The desired task specifications provide a scaffolding for a sufficient set of such tests and the performance specifications for each control system an associated performance measure.

Task	**Test**	**Test specs**	**Measure**
Move full body	Straight line	distance $= 10.0\,\mathrm{m}$	Full body error
Move full body	Hairpin	$\kappa = 0.5\,\mathrm{m}^{-1}$	Full body error
Move full body	Hairpin	$\kappa = 0.33\,\mathrm{m}^{-1}$	Full body error

For simplicity in visualizing results, the USM used for simulations contains two links, the rear joint being the the base link. The control objective is therefore to ensure that both links adheres to the path, see fig. 15.6.

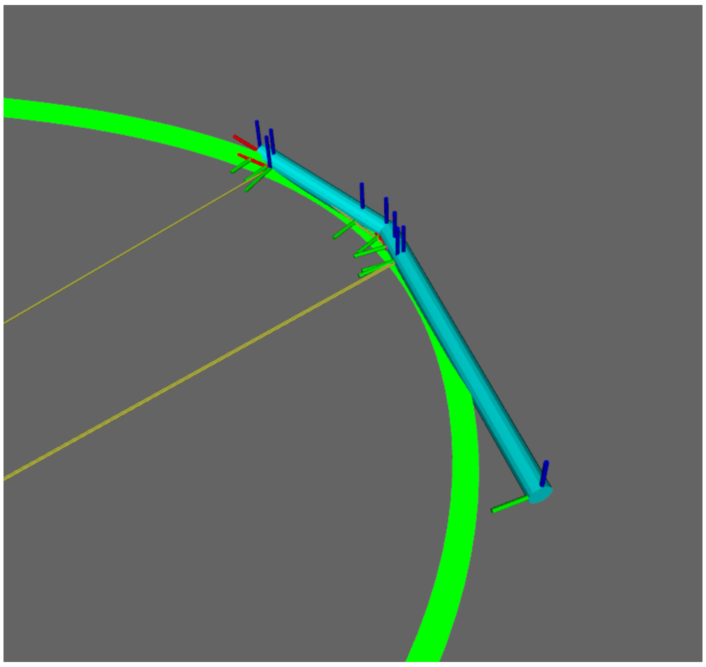


Figure 15.6: .

15.4.2.1 Straight line

Straight line path following mainly demonstrate the performance of the guidance algorithm and verify the existence, or lack thereof, of transient behaviour due to the kinematic control methodology. The goals of this test is therefore to document both base frame and end-effector convergence.

Firstly, base link convergence is evaluated for varying speed and lookahead distance. As specified in section 15.2.1, the goal is to minimize overshoot while not compromising convergence time considerably. In fig. 15.7 base link convergence for varying lookahead distance is demonstrated at $u_{b,d} = 0.5\,\mathrm{m\,s^{-1}}$ and $u_{b,d} = 1.0\,\mathrm{m\,s^{-1}}$. Consequently, $\Delta_b = 0.5\,\mathrm{m}$ results in the best performance according to the specifications. Note, however, that lowering Δ_b results in a larger yaw rate peak, implying a negative impact on navigation.
Another important performance aspect is the effect of the velocity reference proportional gain, k_{p,ψ_b} from eq. (15.13), concerning overshoot and convergence time. fig. 15.8 display

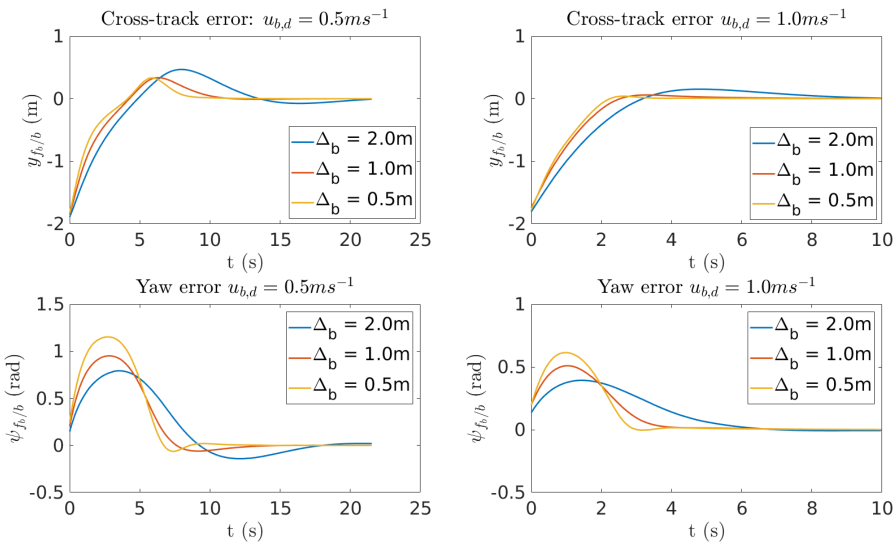


Figure 15.7: Base link cross-track and yaw error displayed using $k_{p,\psi_b} = 0.8$. From the leftmost figures, it is evident that lowering the surge speed results in a second order response due to the coupling between relative attitude and positioning with respect to the path.

how lowering the gain increase yaw error, but decrease overshooting at $u_{b,d} = 0.5$ m and $u_{b,d} = 1.0$ m. As the proportional gain decrease, the yaw rate peak will increase proportionally. Increased yawing error, on the other hand, imply an increased end-effector error. Therefore, choosing the proportional gain requires trading off base link convergence for transient end-effector error.

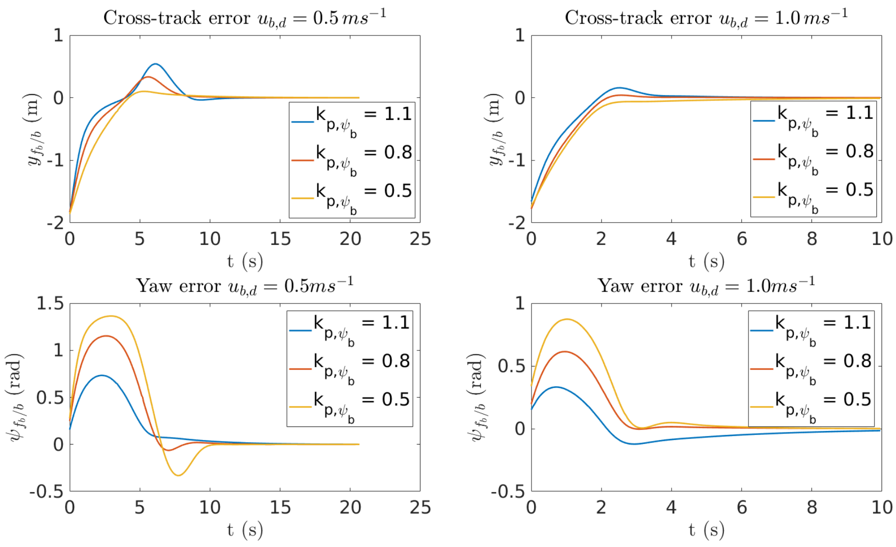


Figure 15.8: Base link cross-track and yaw error displayed using $\Delta_b = 0.5$ m.

The second objective from this test is verification of end-effector convergence. As the end-effector position error is coupled with the base link velocity, convergence is tested during medium conditions; $\Delta_b = 1.0$ m, $k_{p,\psi_b} = 0.8$. The end-effector gain, velocity direction and magnitude are the main contributors to convergence and are therefore varied. Intuitively, the desired task velocity magnitude $u_{ee_0,d}$ increase convergence speed and reduce overshoot. It will particularly cause quicker convergence when the yaw error is large, independent of the cross-track error, when increased. Scaling the lookahead distance Δ_{ee_0} smooth out the desired task velocity direction, $\vec{t}^{I}_{i,d}$, when increased. Whence a large desired task velocity magnitude and a small lookahead distance is desirable. Scaling the task error with $\mathbf{\Lambda}_i$ will have an increased effect with an increased cross-track error. In the case of yaw errors larger than a quarter circle, $\psi_{f_{ee_0/ee_0}} > \frac{\pi}{2}$, the task error will switch signs and pull in the opposite direction. The task error should therefore be tuned accordingly, such that the desired task velocity dominates.

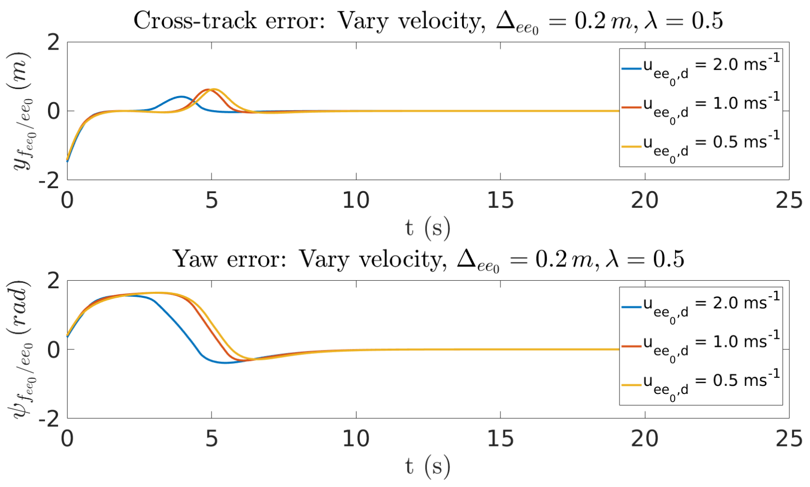
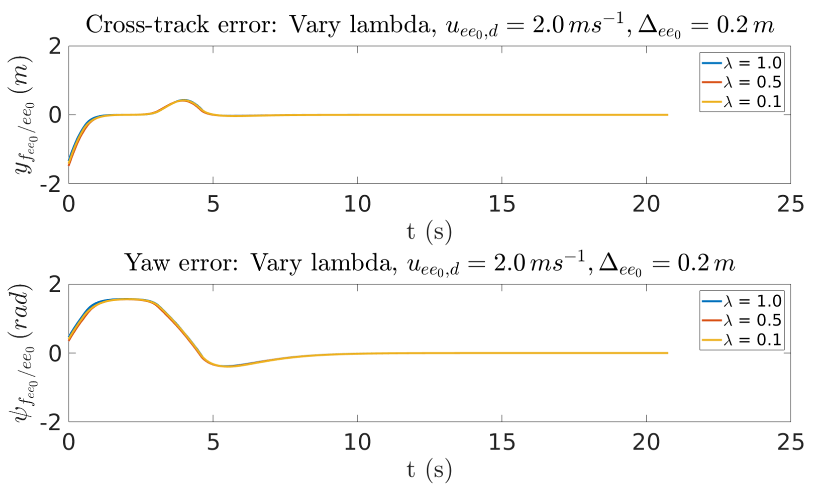
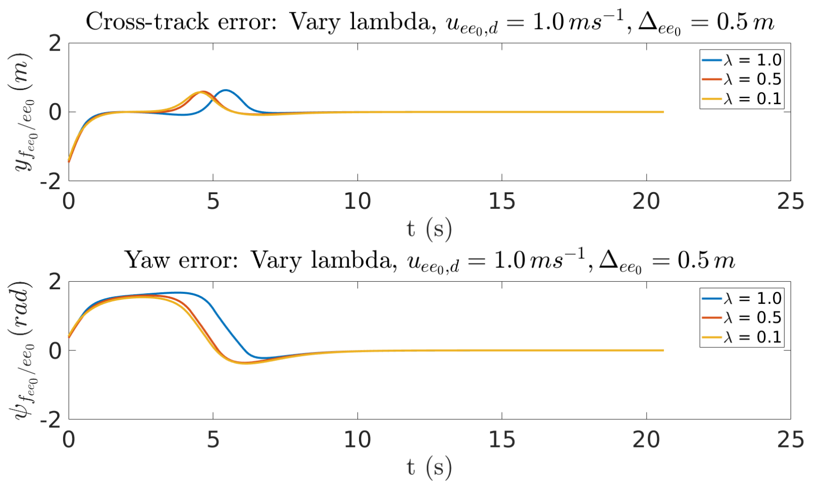


Figure 15.9: End-effector link cross-track and yaw error displayed using $\Delta_{ee_0} = 0.2\,\mathrm{m}$, $\lambda = 0.5$ with varying desired task velocity magnitude.

The effect of increasing the desired task velocity magnitude for constant task error gain and lookahead is demonstrated in fig. 15.9. Clearly, both cross-track and yaw errors are subject to fast convergence for increased velocities. The bump in the cross-track error arise due to the base link overshooting — the magnitude equals the orange line in fig. 15.8. The effect of varying the task error gain holding the desired task velocity magnitude constant is displayed in fig. 15.10a for $u_{ee_0,d} = 2.0\,\mathrm{m\,s^{-1}}$ and in fig. 15.10b for $u_{ee_0,d} = 1.0\,\mathrm{m\,s^{-1}}$. In fig. 15.10b, increased gain relative to velocity cause a delayed peak in cross-track error. This is an effect of the of the yaw-error increasing above a quarter circle, as mentioned earlier. In fig. 15.10a, a sufficiently large velocity seems to alleviate the effect of an increasing task error gain — this only holds up to a certain point, of course. The desired task velocity magnitude will therefore be chosen considerably larger than the task error gain to ensure convergence in the sense displayed above.



(a) End-effector link cross-track and yaw error displayed using $\Delta_{ee_0} = 0.2\,\mathrm{m}$, $u_{ee_0,d} = 2.0\,\mathrm{m\,s^{-1}}$ with varying task error gain.



(b) End-effector link cross-track and yaw error displayed using $\Delta_{ee_0} = 0.2\,\mathrm{m}$, $u_{ee_0,d} = 1.0\,\mathrm{m\,s^{-1}}$ with varying task error gain.

Hairpin

Curved path following is demonstrated using hairpins of varying curvature. Hairpin curves provide a test both for both a steady state and transient phase — i.e. constant and varying curvature respectively — such that both base and end-effector convergence can be evaluated under such conditions. Figure 15.11 display the curvature profiles of two hairpin curves with curvatures $\kappa = 0.5\,\mathrm{m}^{-1}$ and $\kappa = 0.33\,\mathrm{m}^{-1}$, where transient phases can be recognised from a linearly increasing curvature.

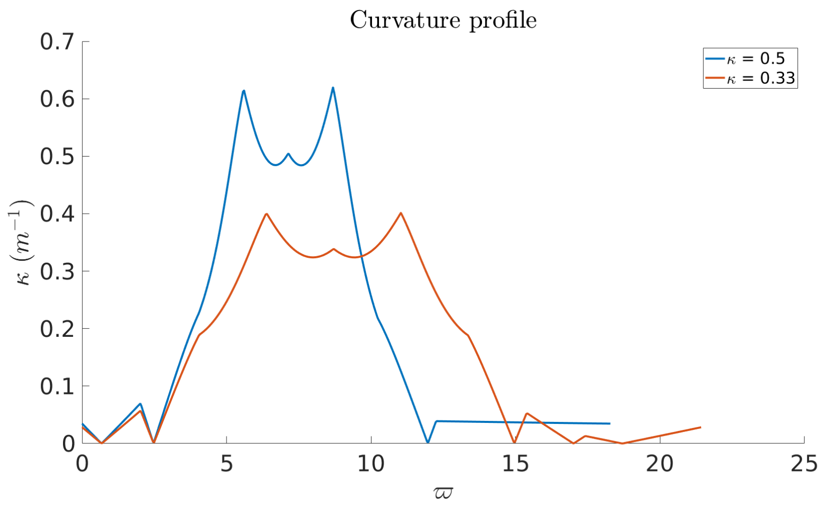
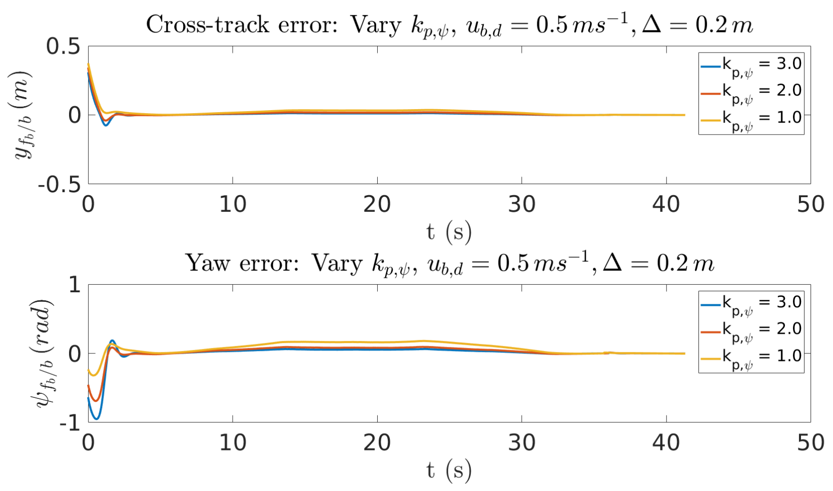


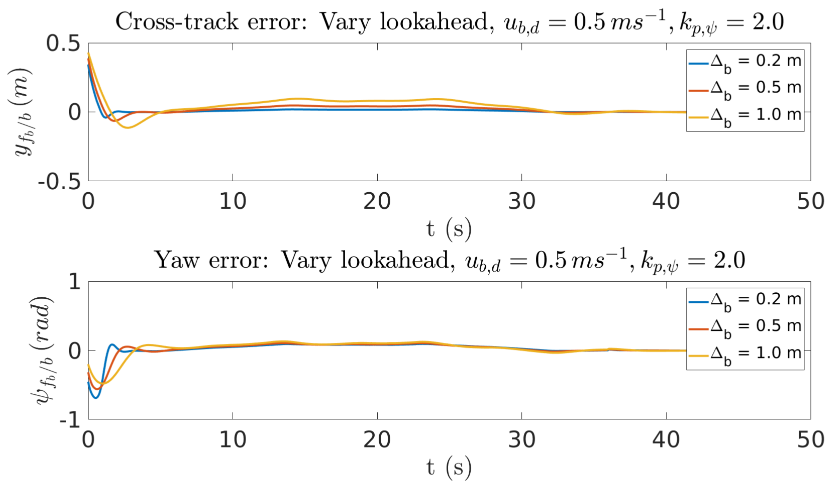
Figure 15.11: Curvature profiles of two hairpin curves with $\kappa = 0.5\,\mathrm{m}^{-1}$ and $\kappa = 0.33\,\mathrm{m}^{-1}$. As the hairpins are computed with natural cubic splines, they do not represent perfect half circles. Consequently, transient phases are recurring throughout both profiles.

Firstly, steady state behaviour of the base link is demonstrated for varying lookahead distances and the proportional angular deviation gain as in section 15.4.2.1, see fig. 15.12b and fig. 15.12a. From fig. 15.12b, it is evident that an increasing lookahead cause a significant increase in the cross-track error. The lookahead distance similarly cause the cross-track error to gradually resemble the curvature profile. This observation implies that increasing the lookahead distance makes the robot less reactive to changes in curvature, which makes sense as it can be thought of as a *damping term* on the desired angular setpoint, eq. (15.2). Increasing the velocity reference proportional gain have a similar effect as the lookahead distance, applied to the yaw error instead of the cross-track error. Consequently, to alleviate the effect of a changing curvature, the above-mentioned tuning parameters should be chosen $k_{p,\psi} \geq 2.0$ and $\Delta_b \leq 0.2$. Note however, that such a choice of parameters will cause large overshoots, as discussed in section 15.4.2.1. Thus, either a trade-off between these two cases have to be found, or a gain-scheduling scheme have to be implemented. This will, however, not be considered any further in this thesis.

Secondly, steady state behaviour of the end-effector is demonstrated following the same procedure as in section 15.4.2.1. The effect of having a constantly small lookahead

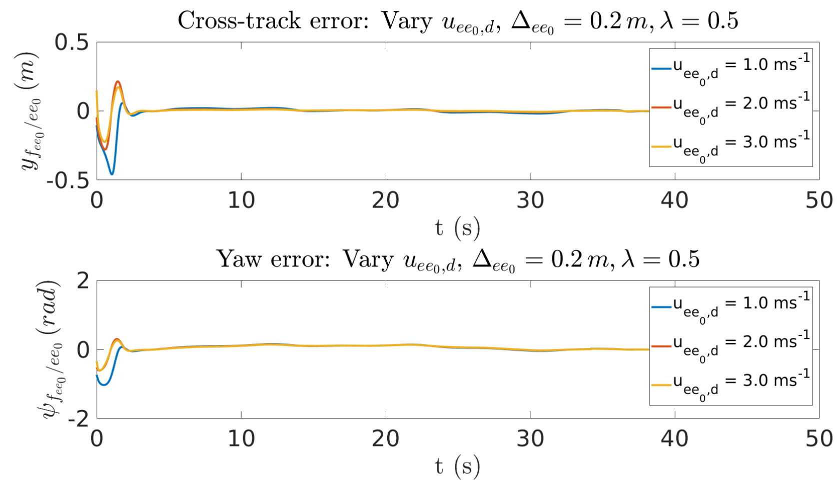


(a) Steady state hairpin base link cross-track and yaw error with constant speed and lookahead distance, while varying proportional velocity reference gain.

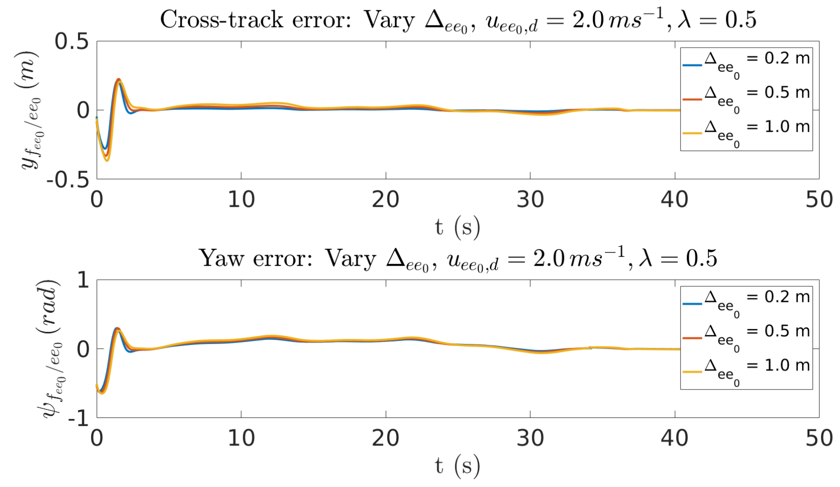


(b) Steady state hairpin base link cross-track and yaw error with constant speed and proportional velocity reference gain, while varying lookahead distance.

distance and a constant task error gain is demonstrated in fig. 15.13a. Looking away from the transient convergence in the beginning, the end-effector sensitivity towards changing curvature is reduced by an increased desired task velocity. In fig. 15.13b the effect of holding the desired task velocity constant and varying the lookahead distance is demonstrated. The result is similar to that for the base link, the yaw error follow the curvature changes with an increasing extent with increasing lookahead. Again it is therefore desirable to reduce the lookahead distance and increase the desired task velocity to ensure robustness.

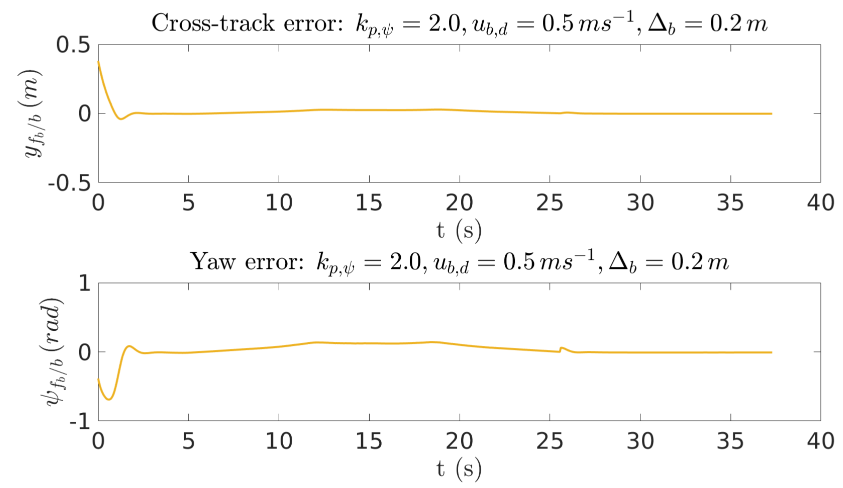


(a) Steady state hairpin end-effector cross-track and yaw error with constant lookahead distance and task error proportional gain, while varying desired task velocity.

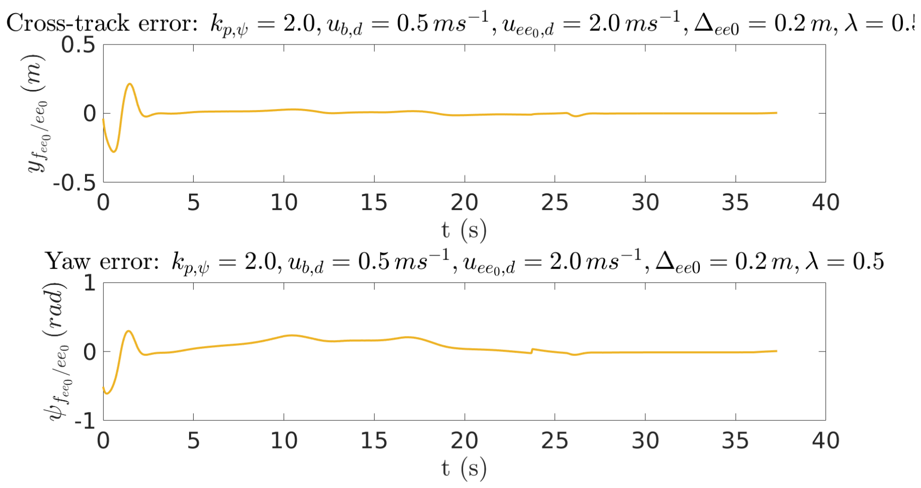


(b) Steady state hairpin end-effector cross-track and yaw error with constant lookahead distance and desired task velocity, while varying task error proportional gain.

Finally, the performance of both base link and end-effector is tested for a curve with $\kappa = 0.5\,\mathrm{m}^{-1}$. This is to validate how the performance of the control system scales with varying track conditions. The results are shown in fig. 15.14a and fig. 15.14b, with base link parameters $u_{b,d} = 0.5\,\mathrm{m\,s}^{-1}$, $k_{p,\psi} = 2.0$, $\Delta_b = 0.2\,\mathrm{m}$ and end-effector parameters $u_{ee_0,d} = 2.0\,\mathrm{m\,s}^{-1}$, $\Delta_{ee_0} = 0.2\,\mathrm{m}$, $\lambda = 0.5$. It is evident that the errors are amplified during transient phases, but that they remain bounded. Depending on the precision desired in actual operation, the bound can be tightened by parameter tuning as explained in this and the previous section.



(a) Transient hairpin base link cross-track and yaw error.



(b) Transient hairpin end-effector cross-track and yaw error.

Part VII

Closing the Loop

16 | Connecting the Modules

SYSTEM integration is a big part of this thesis since multiple modules are implemented and combined to create a complete autonomy pipeline. Seeing as these components have been discussed in detail in parts II and IV to VI, this chapter will look more into how these modules are interconnected and demonstrate how the more complete system operates. These closed-loop connections will be tested on a selected set of simulated cases, from which the overall system performance will be evaluated.

16.1 Combining Object Detection and Planning

One of the underlying objectives of this thesis was to use semantic, or contextual, information to help drive the inspection operation. This was intended achieved through the implementation of the object detection module in part IV. The idea behind this was to use object detection techniques to segment the incoming images from which more context-aware path planning could be performed. In the complete system, this information would arrive through the semantic map provided by the SLAM module. For this test, however, the semantic information was shared directly to the exploration module from the classification system, as illustrated in fig. 16.1. This is mostly done to simplify the test, partly because the current SLAM method provides sparse maps (see section 16.2 for examples of the more complete semantic maps). Of course, if the pipe location is known exactly, the robot can

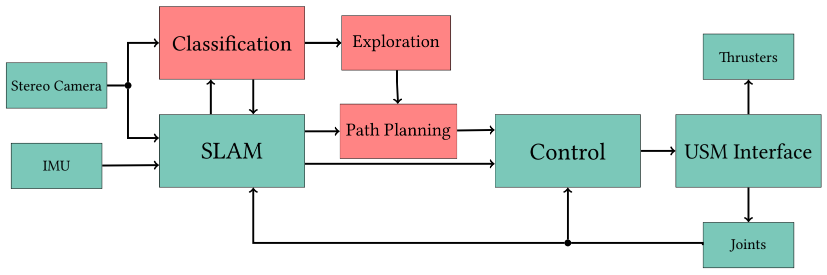


Figure 16.1: Altered block diagram structure for easier testing of the object detection, exploration, and planning modules.

be given a set of precalculated waypoints. Therefore, an assumption for these test cases is that the pipe location is not entirely known.[1] Instead, the positions of a docking station and selected subsea structures, to which the pipes are connected, are assumed known. The USM is then intended to start its inspection by approaching one of these subsea structures,

[1]This is not unrealistic, as the pipe could have shifted or been covered by sand, biomass, or sediment.

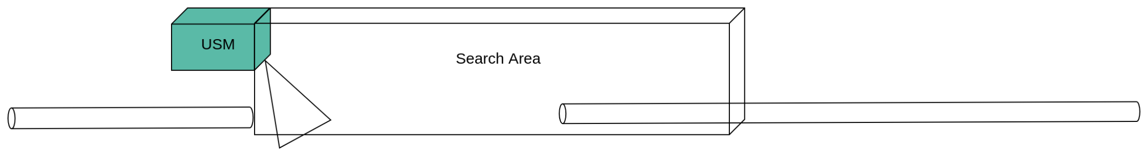


Figure 16.2: Pipe tracking is lost due to occluded segments.

identify the pipe based on its camera feed, then continuously follow the pipe until the second subsea structure is reached.

To follow the pipe, new waypoints needs to be set along the pipe based on incoming images. Based on the implementation specifics from section 10.1, the output consists of a bounding box estimating the pipe position in the image, with a center point close to, or on, the pipe itself — in the case of relatively straight pipes, this should be a reasonable assumption. This point is then re-projected from the image plane to the world frame, along the pipe at a fixed distance (using the intrinsic camera matrix, briefly discussed in section 4.4.1), and passed as a goal to the path planning module. As the purpose of this is to test the modules, the goal points were also placed a set distance above the pipe, to emulate a proximity/depth sensor, or depth information from the SLAM system, and simplify the problem slightly. Furthermore, instead of just sending a point directly, a point cloud containing all points labelled as pipe could be passed to the exploration module. From this, the next-best goal for following the pipe could be extrapolated. An interesting problem arises if the USM loses track of the pipe, however. This could happen due to the pipe being occluded by particles thrown around by the thrusters or completely hidden by sand or biomass. In this case, the USM is required to re-locate the pipe. To this end, this thesis proposes to utilize the exploration module.

Upon losing the pipe, a search area, or geofence (see chapter 14), will be established. This area originates from the USM, extending a distance along the estimated pipe location, as exemplified in fig. 16.2. If the pipe is located within this region, the operation resumes. If not, however, an expanded search area will be defined, as shown in fig. 16.3. By performing this extended search, the USM will be able to re-locate the pipe in the case of a hidden pipe bend. This process will continue a set number of times, expanding the search area each time. If the pipe cannot be rediscovered, the operation terminates by sending the USM back to the docking station and alerting the possible need for manual remote inspection. To test this, a simple simulated scenario was set up.

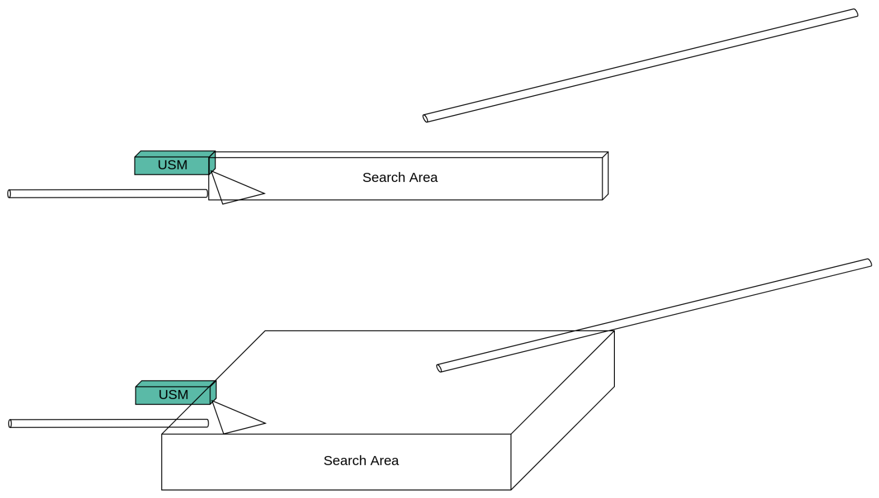


Figure 16.3: Example scenario where the pipe is lost due to a hidden bend.

16.1.1 Case: Context-driven Subsea Pipe Inspection

In order to test the system with the path planning and classification module working as a closed looped system, a simulated scenario was constructed. This scenario consisted of an operation where the USM was supposed to deploy from a docking station at the sea floor, before it travelled to a nearby subsea structure and initiated an inspection. The pipe would then be followed until the second subsea structure was reached. Upon reaching the second subsea structure, the USM would then return to the docking station. Once the USM reached the docking station, the operation was complete. An overview of the case can be seen in fig. 16.4.

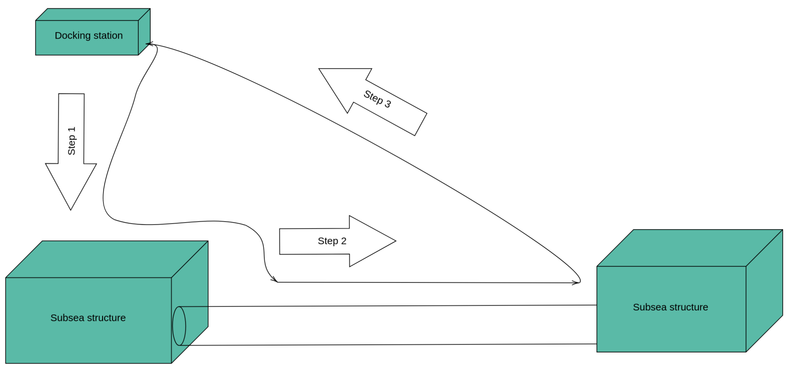


Figure 16.4: Illustration of the simulated pipe inspection operation.

The goal of the experiment was to see how the path planner could utilize the classification module to explore along a pipe section. Since the test was conducted in a simulated environment, where the pipe is continuous and visible, the test was performed without the exploration module for finding the pipe upon lost tracking. The commands for when to start the operation, as well as returning to the docking station, were given manually. In the simulated environment, the subsea structures are represented as large red boxes and the docking station as a large blue box. These colors were primarily chosen to be easily distinguishable from the background for human observers, and is not exploited by the classification algorithm.

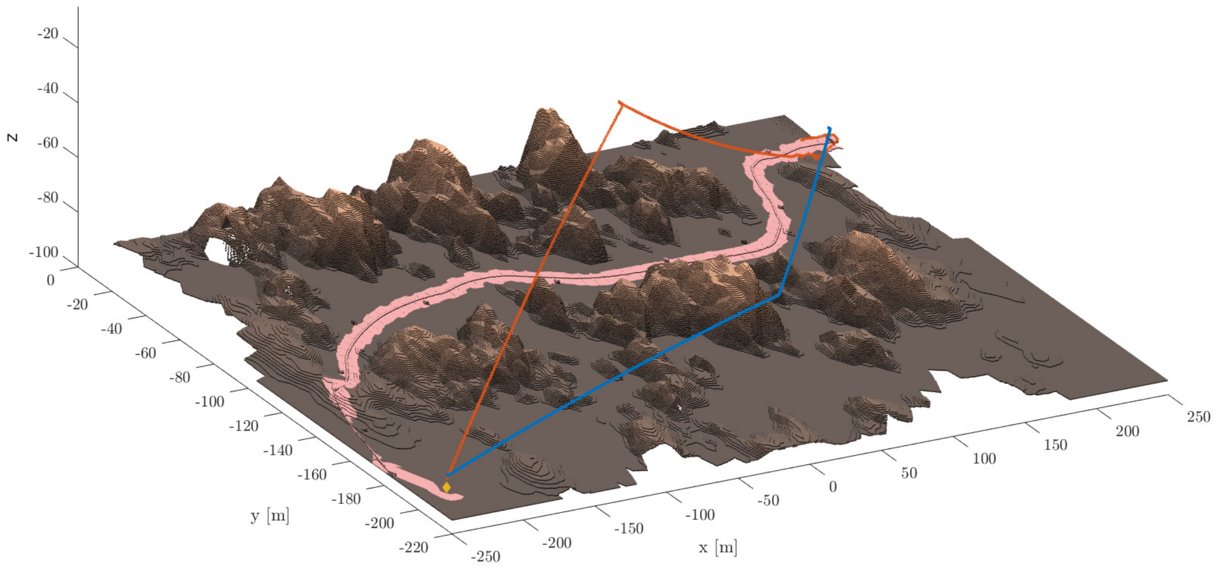
From the experiment, it was observed that the USM sometimes oscillated between the sides of the pipe rather than staying exactly above it (this is present in the video linked in fig. 16.5). This is due to the goal position only being reliant on the center pixel of the bounding box, and is, thus, sensitive to bends in the pipe. This results in the center pixel being re-projected away from the pipe. It was also observed that, when this occurred, the next goal would be closer to the pipe and, after some time, the USM was steered back on track. If instead operating on the final semantic map

Figure 16.5: Video of pipe inspection section of the experiment designed to test the combination of the object detection and planning.

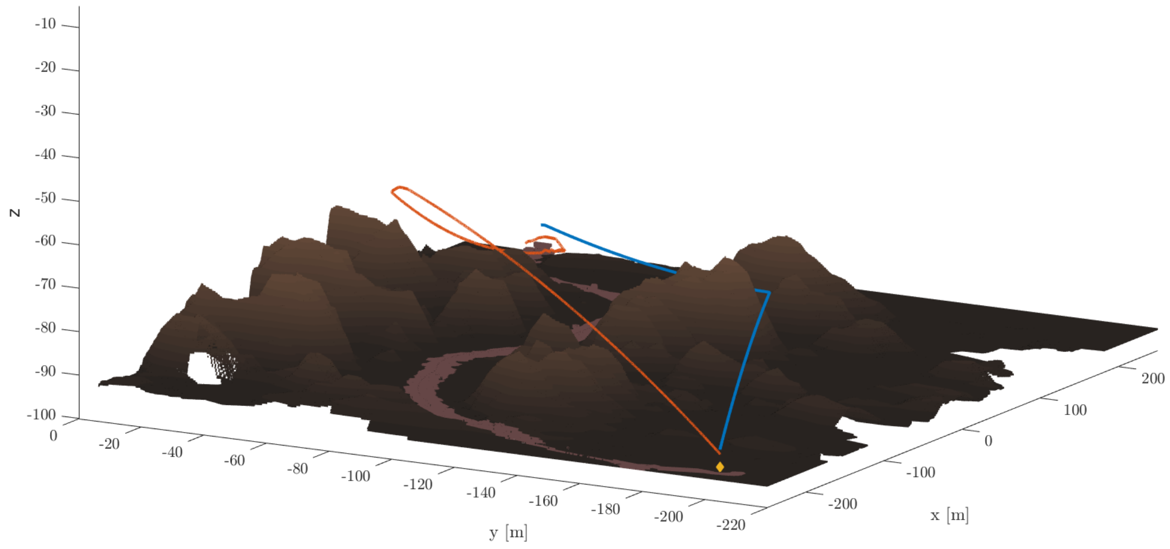
The resulting map produced by following the pipe is shown in fig. 16.6 as a lighter shade, most easily visible in fig. 16.6a and section 16.1.1. To further test the collision avoidance, the USM was given two return trips starting at the final subsea structure. In the first, no a priori knowledge of the map was present, which lead to several replanning stages and a final path with increased clearance. This is shown in fig. 16.6 as the red path. In the second scenario, the map of the surrounding environment was assumed at least partially known. In this case, the return path were much more direct while still providing sufficient clearance to guarantee a safe path. This is shown in fig. 16.6 as the blue path.

To see if the system could handle a lost pipe, the same setup as described above was used, with one simple alteration. A section of the pipe was *hidden* by covering it with a rectangle such that the object detection fails to locate any pipe segments. This initiates an exploration action, starting with a relatively narrow search area. This setup is shown in fig. 16.7. In fig. 16.7a, the robot loses track roughly around (x, y) = (7,-80). A narrow exploration area is then assigned, shown as a yellow box in fig. 16.7, and the exploration module drives the relocation process. As mentioned in previous chapters, the pipe is assumed straight over short distances, which is not exactly the case in this simulation. It is evident from fig. 16.7a that the search area is offset slightly from the actual pipe location. Due to the relative small curvature of the pipeline, however, the relocation procedure succeeds in rediscovering the pipe, allowing the USM to continue in normal operating conditions.

The placement of the exploration area is not perfect, and could be improved upon through extrapolating the pipe itself given a labeled point cloud. All in all, however, the operation was a success and the USM was able to complete the entire operation in a satisfactory manner. The entirety of the first part of the experiment, without the exploration module, can be viewed through the QR code found in fig. 16.5.

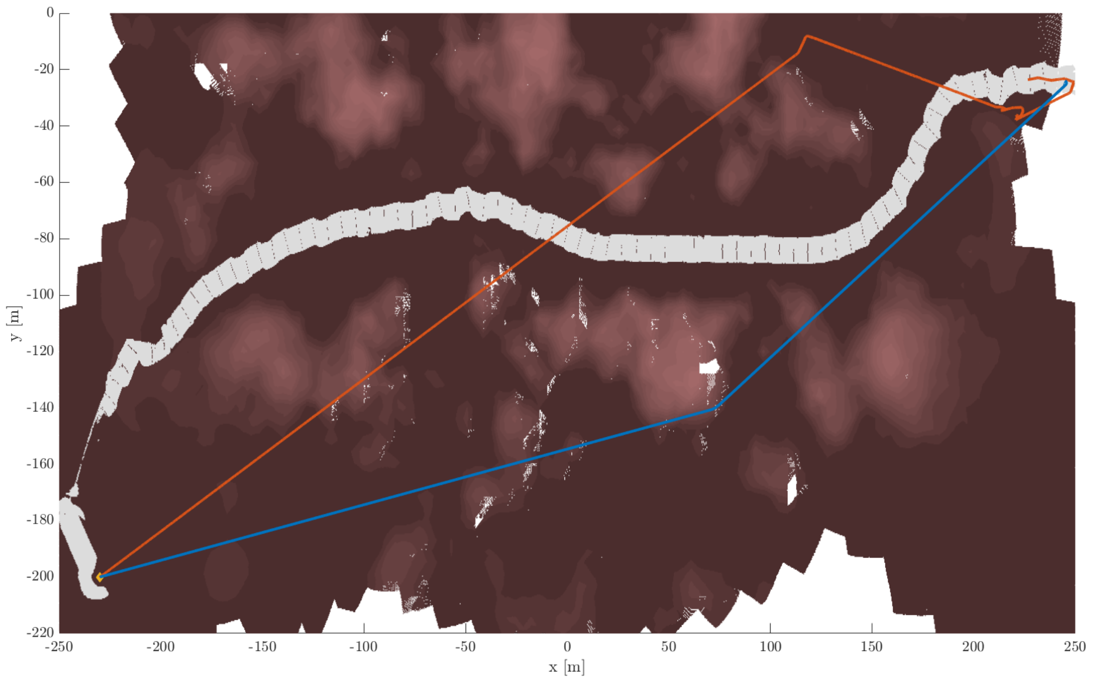


(a) Overview of the entire inspection and return operation.



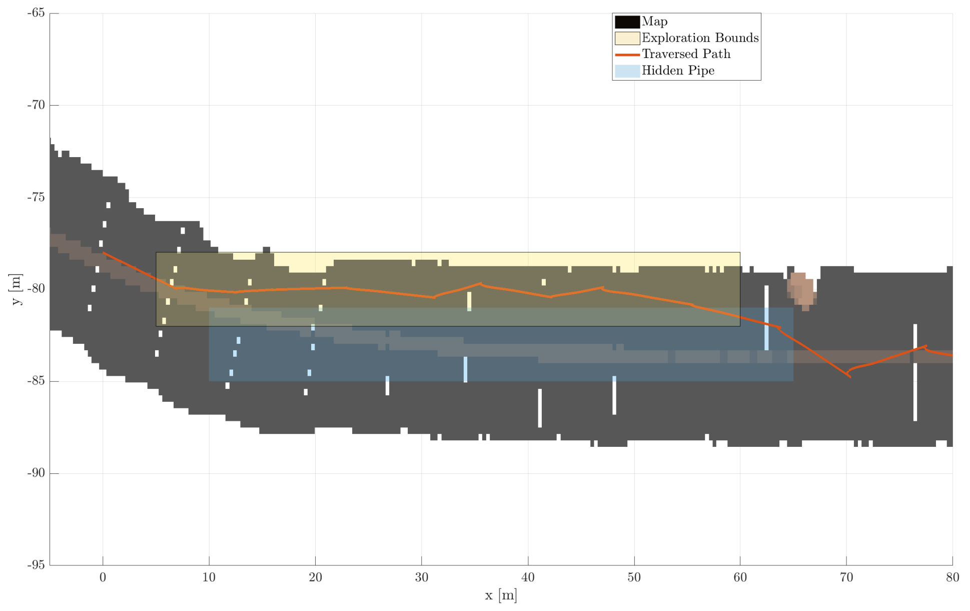
(b) Slightly more angled view, showcasing the different depths of the return paths.

Figure 16.6: Example paths calculated from a subsea structure back to the docking station (yellow) after a completed inspection case. The red curve shows the path calculated without knowledge about the environment surrounding the pipe, resulting in a couple of replanning situations and a much larger clearance due to repeated collision avoidance scenarios. The blue curve indicate the path calculated with knowledge about the surrounding area. The lighter patches of the map show the inspected area during execution.

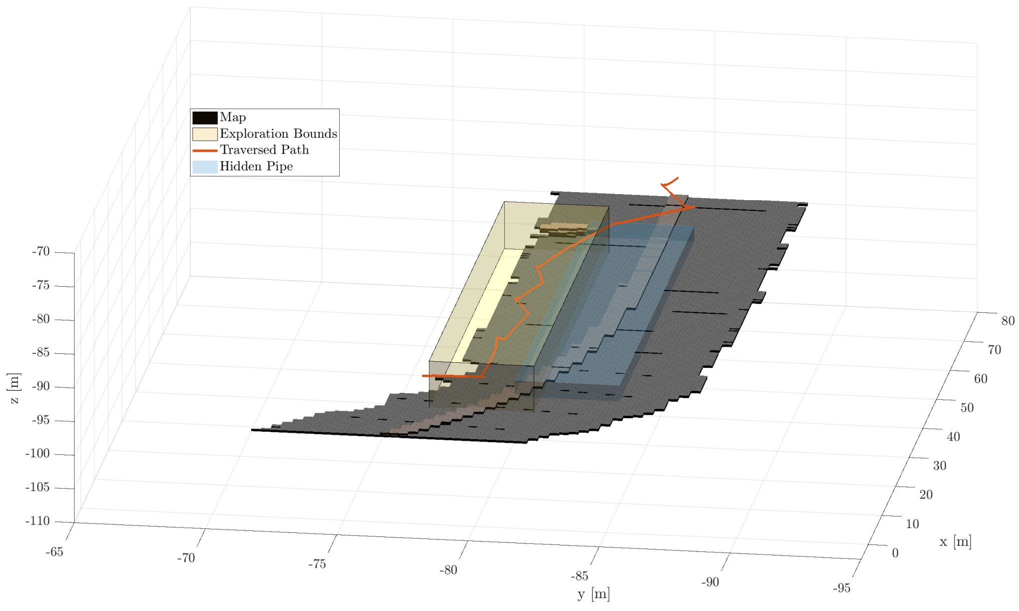


(c) Top-down view of the entire inspection operation and the return paths.

Figure 16.6: Continued figure showing the same case as the previous page, but displayed in a top-down view.



(a) Top-down view of the pipe re-location procedure.



(b) Side view of the pipe re-location procedure.

Figure 16.7: Results from a small pipe re-location procedure. After the USM encounters the hidden pipe (blue), the exploration module is initiated with the search bounds shown in yellow. The exploration then continues until the pipe is re-located.

16.2 Combining Object Detection and SLAM

The output from the SLAM system is, generally, a point cloud constituting the map, which is used to locate the USM. However, for any contextual planning to occur, it is important to have a semantic understanding of what is in the map. This context can be provided from the classification system. The desired output will be a map augmented with semantic information. In order for this augmentation to occur, it is necessary to have some sort of connection point between the classification and SLAM module. The output of the classification module will be bounding boxes containing the information about the detected classes, as well as a label of which frame the bounding boxes are classified from. Whenever the SLAM system decides that a image I_k contains enough new information for it to be accepted as a key-frame and thus be used to generate new map points it simultaneously transfer I_k to the classification module and the mapping module. While the mapping module uses I_k to generate new map points, performs local bundle adjustment and performs map maintenance, the object detection detects bounding boxes in the image and returns them to the SLAM system. Once the mapping module is finished, it projects all the visible map points into the image I_k and gives them a labeling score if they are within one of the bounding boxes. The bounding boxes are stored together with the key-frame I_k, so that when a new map point p_i which are visible by I_k are generated later on by a new key-frame $I_k + n$, the new map point p_i is matched against the stored bounding boxes of I_k and the labeling score is incremented accordingly. To ensure each map point only is labeled by the same key-frame once each map point keep a list of key-frames they have already been labeled by. Figure 16.8 show the schematics of the connection between the Classification and SLAM module.

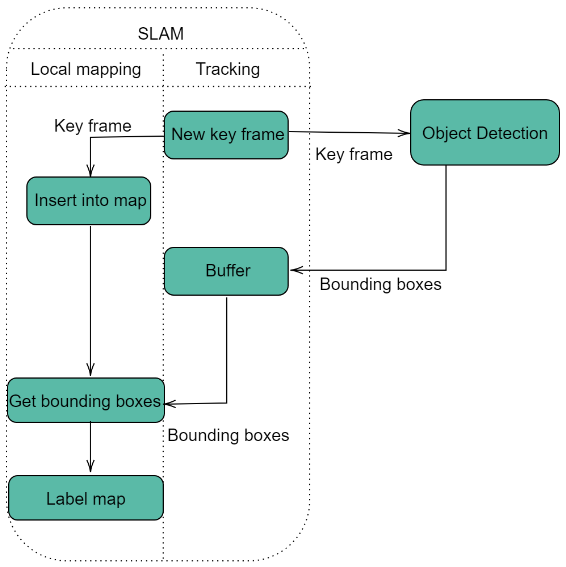
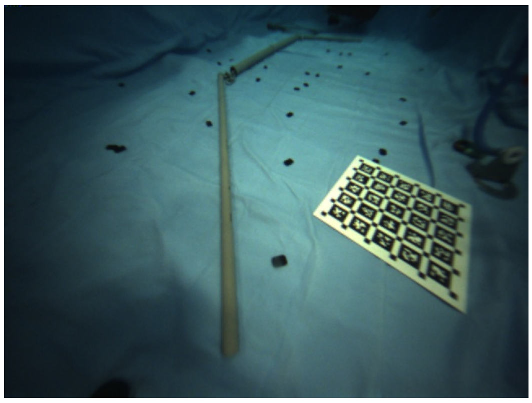
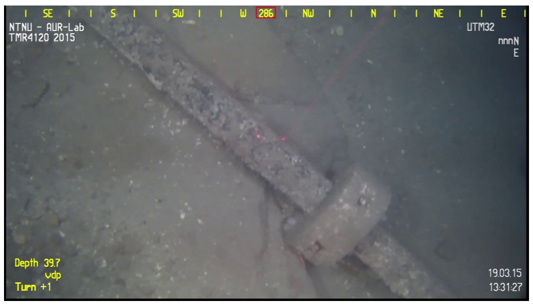


Figure 16.8: Schematic of connection between Classification and SLAM module

Map points can, of course, be classified multiple times, as they will be visible from multiple key frames. This means that there can be different class assignments for the same map points. In order to account for this, a voting scheme was necessary. The voting scheme works by having a counter for each possible class, including a no class-label, associated with the map point. On each positive hit from the classification, this counter will be iterated. Upon evaluating the semantic map, this voting scheme will provide a representation on the certainty of which class the map point belong to. The class with the most classifications will then be selected and the certainty will be represented as a percentage of the positive classifications of said class in relation to the complete number of classifications for the given map point.

To see how the combination between the classification and SLAM modules worked, the following test was conducted. Three pipe segments were placed in the test basin. Additionally, various small weights where scattered around the test basin to ensure that there was a sufficient amount of features present for the SLAM algorithm to operate well. Figure 16.9a shows an example of these conditions. The hardware platform was then moved around in the test basin while recording the data as ROS Bags to process the data at a later point.


(a)



(b)

Figure 16.9: Comparison of the experiment setup in the test basin and a real operational case.

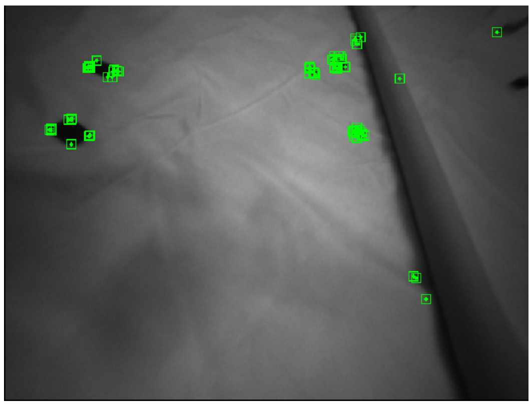
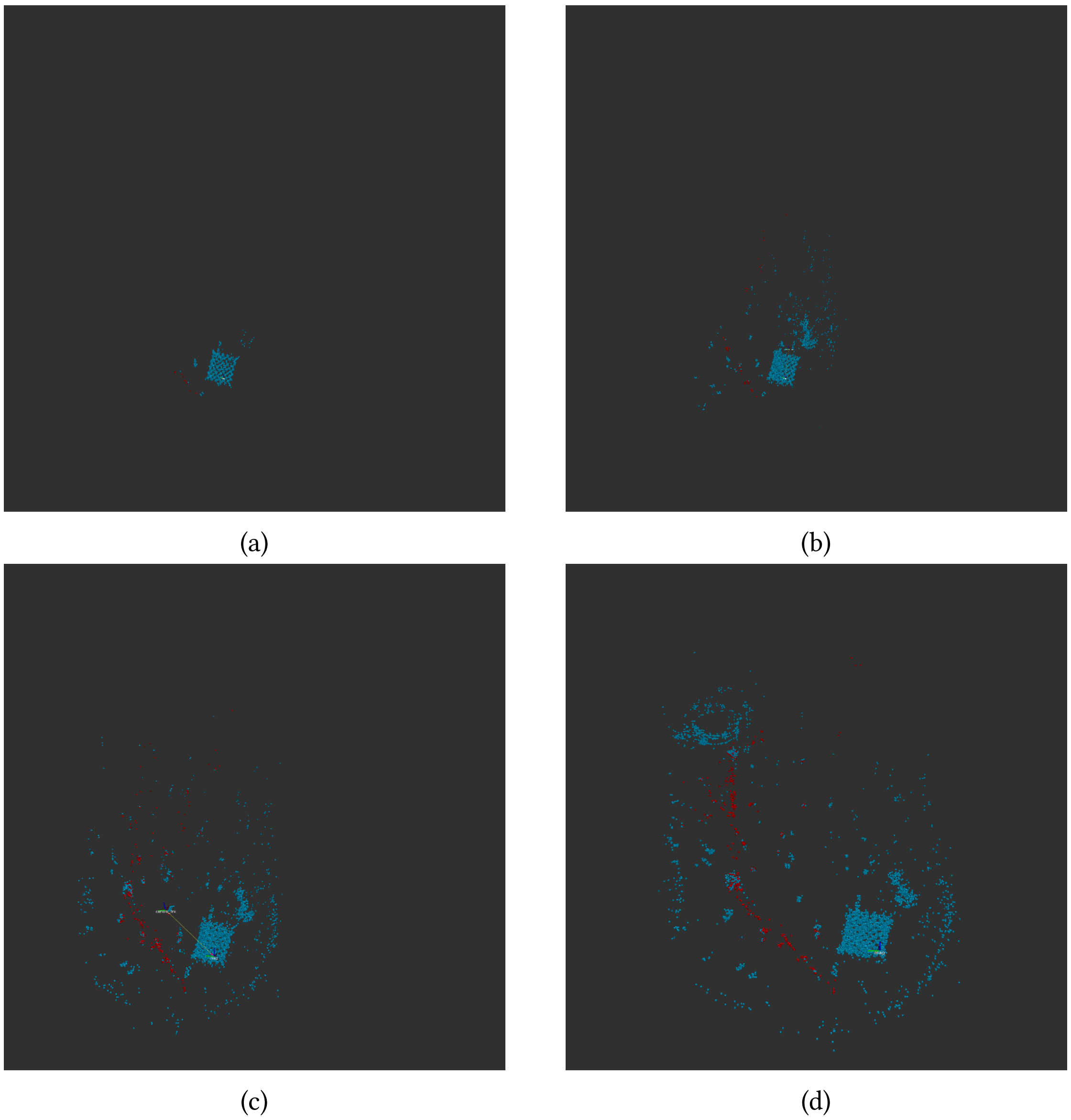


Figure 16.12: Example of smooth pipe resulting in few map points on the pipe itself.

Figure 16.11 shows the final SLAM map from the experiment where the SLAM module was combined with the Classification module to produce a labeled map. The pipe is clearly visible as a series of red map points. However, there are several outlying false positives present This was due to the bounding boxes encompassing features that are relatively far from the pipe, especially when the pipe located close to a 45° angle in the image. A way to reduce these outliers could be to do semantic segmentation from the classification module, make the voting scheme stricter, incorporating the confidence parameter of the bounding boxes in the voting scheme, or employ an algorithm to reduce the outliers. This thesis will not investigate these im-



(a) (b) (c) (d)

Figure 16.10: Developing SLAM map with object detection labeling map points.

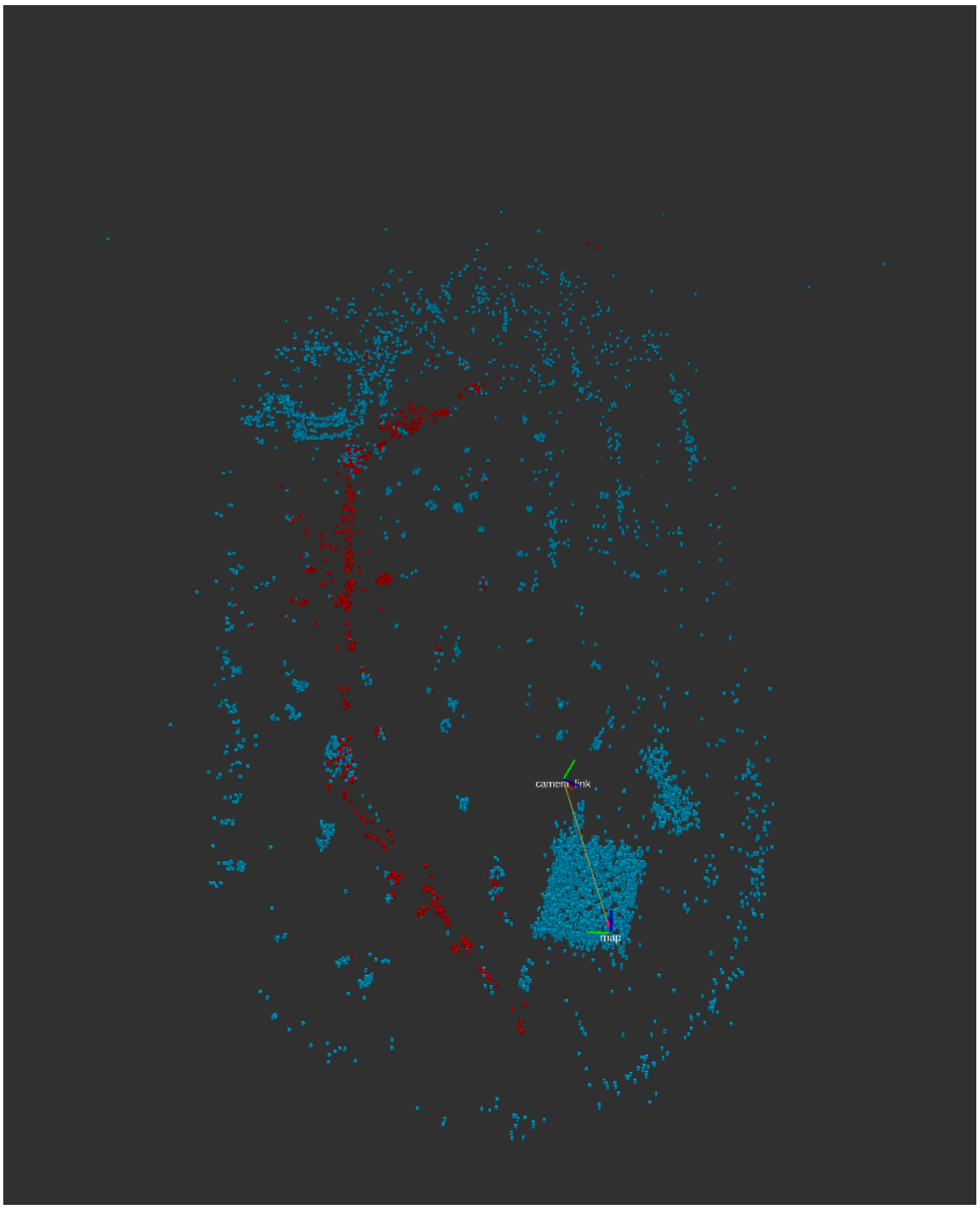


Figure 16.11: Final SLAM map with object detection labeling map points.

provements further, but the most straightforward method would probably be to incorporate a simple RANSAC outlier removal procedure.

In the test basin the pipe was very smooth, which resulted in there being very few map points directly located on the pipe, see fig. 16.12. Most of the labeled points were, therefore, located close to the pipe, but not directly on it. For a real operation, one would expect pipes closer to the example from fig. 16.9b. In these conditions, the pipes are not smooth, due to biofouling, and one would expect the algorithm to find a significantly higher number of map points located on the pipe itself.

16.3 Combining SLAM and Control

The SLAM and control modules are designed to be compatible, but are not explicitly tested together. The reason for this is twofold: a simulation framework compatible with both modules was not prioritized, and the kinematic control system does not have a notion of robustness. The former is an effect of the group firstly prioritizing testing and verification of the individual autonomous modules. The latter is predicated on the the control system being solely kinematic, part VI. The robustness of the full control system largely depends on a properly implemented dynamic controller, see e.g. [204], and is not prioritized, as explained in part VI.

The control system module testing, however, discusses the effect of tuning the controller to enhance SLAM performance, section 15.4. The SLAM performance limitations, section 9.2.2, explicitly limit the allowable linear and angular velocity of the visual sensors. Although the control system enables control of these variables, proper simulation and practical experiments should be conducted to ensure reliable performance.

16.4 Control and Planning

The planning module was able to compute a path, satisfying the criteria specified in section 12.6, for the guidance and control system to follow. Such a path is computed either when a new goal state is received or when a collision is detected along the current path, initiating a replanning procedure. In either way, the control system will stop and wait for a path to be provided. Consequently, the interface between the modules does not require any elaborate testing other than performance testing at the module level and ensuring that the communication between the modules is compatible. This has been verified through simple simulation experiments. An example path following case is shown in fig. 16.13.

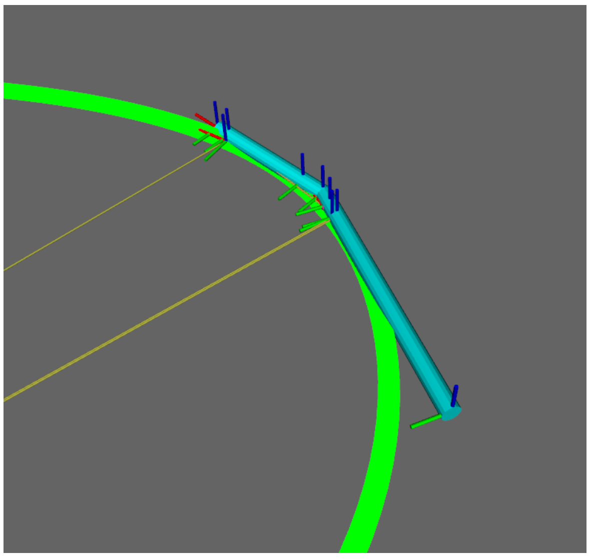


Figure 16.13: USM with two links adheres to a path satisfying kinematic path criteria.

17 | Conclusion

The goal of this thesis was to propose and implement a system architecture capable of performing context-based reasoning and mapping while autonomously executing inspection tasks. This was intended accomplished through the implementation of several modules, each responsible for each of the sub-tasks which needed to be solved.

To accomplish this, a satisfactory sensor rig was needed. A high quality stereo-visual-inertial sensor consisting of a stereo camera, an IMU and a pressure sensor was constructed. The rig synchronize the measurements with the synchronization circuit. The hardware platform worked well for the requirements of this thesis. Due to the goal of constructing a hardware platform that can be utilized by other projects as well, there modular design enable mounting of additional functionality such as an acoustic underwater positioning system or a light source. In short, the hardware platform provided as required and are ready to be developed further to meet new challenges.

Localization and mapping was achieved by implementing a visual-intertial SLAM system. The system works well in the ideal environments of the test basin, the monocular inertial SLAM system estimates scale accurately and are thus able to provide accurate, metric pose estimates at 20.83 Hz with 45 ms latency. Further the robustness of the monocular SLAM system is shown to increase when fusing in a Inertial sensor, particularly with respect to pure rotations and scale drift. In the more realistic environment in the Dora dock, the visual SLAM system struggles. If the cameras are carefully directed at scenes with a lot of contrast and structure while maintained at a close distance, the system manages to keep tracking the scene. Thus one applicable function could be to maintain a given position close to another objects. However except for station keeping, it is not robust and where not able to complete any of the sequences from the Dora Dock dataset. To be able to work robustly in real environments the state estimation can not solely be based on a visual inertial SLAM system. In the long term extending the state estimation to include additional sensors such as underwater acoustic positioning systems, Doppler velocity logger or SONAR is essential. Then the visual SLAM system would only be activated while the scenery is close enough and has sufficient structure and contrast, similarly to the work in [216]. Thus having multiple maps like in [217], but instead of initializing a new map when tracking is lost, rely on the other sensors to estimate the trajectory. Resulting in multiple visual maps, connected by trajectories estimated from other sensors.

With the combined insight of the qualitative and quantitative results, this report concludes that the added challenges present when detecting pipes underwater seem to be solvable using deep learning techniques. The network performed well on similar data to the data it had trained on, but struggled once the conditions changed substantially. The difference in lighting conditions, and proximity to the pipe, had a drastic impact on the performance of the network. However, by augmenting the dataset with a relative small amount of data, it experienced a significant improvement on these conditions. In order to get a system operational, it is suggested to have a surveillance period to see which type of images the network struggle to detect and add some of them to the dataset until the network start performing to a satisfactory level. This work can be done by observing the USM in operation when controlled manually over time. By collecting data in this phase it seems likely that most conditions can be added to the dataset. It will, of course, require a labour intensive process of looking through hours of video footage, but the potential rewards of a fully autonomous systems seem to be worth it. This process will make the network increasingly robust over time.

Through experiments, the implemented Voronoi-based planner was deemed inadequate for an exploration scenario — or a scenario in which only partial map knowledge is present — due to the nature of the operational environment. Instead, an informed sampling-based planning module based on BIT* was used, which, combined with a kinematically constrained state space formulation and natural cubic spline interpolation, provided flyable paths coinciding with the kinematic constraints imposed by the control system. This module was then coupled with an actuator fixated frontier exploration strategy based on point cloud clustering to provide fairly efficient map coverage.

For guidance and control, a cascaded control system consisting of a guidance and a kinematic control algorithm with the UGAS property was designed. Consequently, full body convergence can theoretically be achieved for 3D path following on general curves. This result is validated through simulations on a straight line and two hairpin turns with differing curvature with deviations resulting from base and joint velocity constraints. This experimental setup also demonstrate how the system can be tuned to minimize base link and end-effector overshoot and convergence time under these constraints. The cascade is solely kinematic and should be extended with a dynamic controller to ensure robustness in the presence of noisy sensor data.

17.1 Further Research

In any project, there is always the constraint of time. The following is a collection of topics and improvements the team would have continued with given more time:

- Test MPNet [106] on submarine data through transfer learning. Training data for this could be generated using the presented planner, or any other for that matter, and could thus be specialized to different applications.

- It could be interesting to further develop on AEP [172] to better suit underwater environments/dynamics, e.g. through the use of alternative planning strategies or fusing camera and sonar measurement to be used for optimal viewpoint estimation.
- To better handle energy- or time constrained cases, these aspects could be included in the planning task. This could possibly be accomplished by including current measurements, or by performing more task specific planning by including more of the kinematics and making use of velocity measurements.
- More closely incorporate SLAM information into planning. Although this makes for a less general system, it could allow for the system to better utilize information about the map, landmarks, and uncertainties for improved goal selection and handling of the loss of track.
- An extension of the cascaded stability proof including dynamic control. The development of such a cascaded control system makes it possible to account for software and hardware delays, measurement noise and modelling errors in a robust manner.
- Realize the control system implementation in conjunction with the SLAM module on an actual USM. Exploration of different yaw- and pitch-rates the SLAM module can handle in various conditions, and how the resulting performance affect the control system would be particularly interesting.
- Other networks should be tested and compared to YOLOv3-tiny to see if any solutions would be a more suitable choice.
- Alternative approaches for improving the dataset, such as image augmentation, should be tried in addition to the proposed strategy for improving robustness.
- Traditional computer vision techniques, such as sliding window or RANSAC, should be tried to improve the robustness of the solution as well as narrow in on the spatial location of the pipes within the bounding box.
- The dataset should be labeled on the form required by segmentation networks to see if such networks can be better suited than object detection networks.
- A light source should be added to the hardware platform to enable the system to perform without the heavy dependency upon external light sources.
- Improve the SLAM system to incorporate the stereo information provided by the hardware platform.

- Adding additional sensors to see how they impact the robustness of the SLAM system, such as Doppler velocity log and acoustic underwater positioning.

Bibliography

[1] K. S. Gjerden, *Dynamic Path Planning for Autonomous Exploration.* 2018, Unpublished, specialization project.

[2] O. B. Utbjoe, *Autonomous navigation and mapping for underwater snake robots*, Unpublished, specialization project, 2018.

[3] V. A., *Autonomous navigation and mapping for underwater snake robots.* 2018, Unpublished, specialization project.

[4] J. A. Dowdeswell, J. Evans, R. Mugford, G. Griffiths, S. Mcphail, N. Millard, P. Stevenson, M. A. Brandon, C. Banks, K. J. Heywood, M. R. Price, P. A. Dodd, A. Jenkins, K. W. Nicholls, D. Hayes, E. P. Abrahamsen, P. Tyler, B. Bett, D. Jones, P. Wadhams, J. P. Wilkinson, K. Stansfield, and S. Ackley, "Instruments and Methods Autonomous underwater vehicles (AUVs) and investigations of the ice-ocean interface in Antarctic and Arctic waters," Tech. Rep. [Online]. Available: `www.bodc.ac.uk/`.

[5] G. Marani, S. K. Choi, and J. Yuh, "Underwater autonomous manipulation for intervention missions AUVs," *Ocean Engineering*, vol. 36, no. 1, pp. 15–23, 2009.

[6] R. B. Wynn, V. A. Huvenne, T. P. Le Bas, B. J. Murton, D. P. Connelly, B. J. Bett, H. A. Ruhl, K. J. Morris, J. Peakall, D. R. Parsons, E. J. Sumner, S. E. Darby, R. M. Dorrell, and J. E. Hunt, "Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience," *Marine Geology*, vol. 352, pp. 451–468, 2014. [Online]. Available: `http://dx.doi.org/10.1016/j.margeo.2014.03.012`.

[7] A. Lavin, "Optimized Mission Planning for Planetary Exploration Rovers," *CoRR*, vol. abs/1511.0, 2015. [Online]. Available: `http://arxiv.org/abs/1511.00195`.

[8] P. Tompkins, "Mission-directed path planning for planetary rover exploration," PhD thesis, The Robotics Institute, Carnegie Mellon University, 2005, pp. 1–192. [Online]. Available: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.9399&amp;rep=rep1&amp;type=pdf`.

[9] M. Faria, I. Maza, and A. Viguria, "Applying Frontier Cells Based Exploration and Lazy Theta* Path Planning over Single Grid-Based World Representation for Autonomous Inspection of Large 3D Structures with an UAS," *Journal of Intelligent and Robotic Systems: Theory and Applications*, pp. 1–21, 2018.

[10] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age," *IEEE Trans. Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. [Online]. Available: `http://arxiv.org/abs/1606.05830%0Ahttp://dx.doi.org/10.1109/TRO.2016.2624754`.

[11] L.-Y. Weng, M. Li, Z. Gong, and S. Ma, "Underwater object detection and localization based on multi-beam sonar image processing," *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, no. December, pp. 514–519, 2012. [Online]. Available: `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6491018`.

[12] M. Leonardi, A. Stahl, M. Gazzea, M. Ludvigsen, I. Rist-Christensen, and S. M. Nornes, "Vision based obstacle avoidance and motion tracking for autonomous behaviors in underwater vehicles," *OCEANS 2017 - Aberdeen*, vol. 2017-Octob, no. June, pp. 1–10, 2017.

[13] J. Sverdrup-Thygeson, E. Kelasidi, K. Y. Pettersen, and J. T. Gravdahl, "The Underwater Swimming Manipulator   A Bioinspired Solution for Subsea Operations," *IEEE Journal of Oceanic Engineering*, vol. 43, no. 2, pp. 402–417, Apr. 2018. [Online]. Available: `https://ieeexplore.ieee.org/document/8121980/`.

[14] P. J. From, *Vehicle-manipulator systems : Modeling for simulation, analysis, and control*, eng, London, 2014.

[15] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control.* 2011, pp. 15–41.

[16] J. Sverdrup-Thygeson, E. Kelasidi, K. Y. Pettersen, and J. T. Gravdahl, "The underwater swimming manipulator;a bioinspired solution for subsea operations," eng, *Oceanic Engineering, IEEE Journal of*, vol. 43, no. 2, pp. 402–417, 2018.

[17] O. Egeland and T. Gravdahl, *Modeling and Simulation for Automatic Control.* 2002, vol. 53, pp. 1689–1699.

[18] P. Liljeb ck, K. Y. Pettersen, Stavdahl, and J. T. Gravdahl, *A 3d motion planning framework for snake robots*, eng, 2014. [Online]. Available: `http://hdl.handle.net/11250/286585`.

[19] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*, 1st. Springer Publishing Company, Incorporated, 2008.

[20] M. M. M. Manhães, S. A. Scherer, M. Voss, L. R. Douat, and T. Rauschenbach, "UUV Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation," in *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*, 2016.

[21] L. F. Melo and T. R. Bott, "Biofouling in water systems," *Experimental Thermal and Fluid Science*, vol. 14, no. 4, pp. 375–381, 1997.

[22] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *2011 International Conference on Computer Vision*, IEEE, Nov. 2011.

[23] M. Samek and P. C. Montgomery, "State-oriented programming," 2000.

[24] B. H. Liskov and J. M. Wing, "A behavioral notion of subtyping," *ACM Trans. Program. Lang. Syst.*, vol. 16, no. 6, pp. 1811–1841, Nov. 1994. [Online]. Available: `http://doi.acm.org/10.1145/197320.197383`.

[25] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, no. 1, pp. 1–21, Feb. 2016.

[26] *IEEE standard specification format guide and test procedure for single-axis interferometric fiber optic gyros*, Feb. 1998.

[27] O. J. Woodman, "An introduction to inertial navigation," University of Cambridge, Computer Laboratory, Tech. Rep., 2007.

[28] R. Mur-Artal and J. D. Tardos, "Visual-inertial monocular SLAM with map reuse," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 796–803, Apr. 2017.

[29] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "SVO: Semidirect visual odometry for monocular and multicamera systems," *IEEE Transactions on Robotics*, vol. 33, no. 2, pp. 249–265, Apr. 2017.

[30] J. Engel, V. Koltun, and D. Cremers, “Direct sparse odometry,” Jul. 9, 2016. arXiv: `http://arxiv.org/abs/1607.02565v2` `[cs.CV]`.

[31] D. Scaramuzza, A. Martinelli, and R. Siegwart, “A flexible technique for accurate omnidirectional camera calibration and structure from motion,” in *Fourth IEEE International Conference on Computer Vision Systems (ICVS’06)*, IEEE, 2006.

[32] T. Łuczyński, M. Pfingsthorn, and A. Birk, “The pinax-model for accurate and efficient refraction correction of underwater cameras in flat-pane housings,” *Ocean Engineering*, vol. 133, pp. 9–22, Mar. 2017.

[33] D. Li, L. Xu, X. S. Tang, S. Sun, X. Cai, and P. Zhang, “3D imaging of greenhouse plants with an inexpensive binocular stereo vision system,” *Remote Sensing*, vol. 9, no. 5, 2017.

[34] J. Weng, P. Cohen, and M. Herniou, “Camera calibration with distortion models and accuracy evaluation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 965–980, 1992.

[35] N. Kaempchen and K. Dietmayer, “Data synchronization strategies for multi-sensor fusion,” in *Proceedings of the IEEE Conference on Intelligent Transportation Systems*, vol. 85, 2003, pp. 1–9. [Online]. Available: `https://www.researchgate.net/profile/Nico_Kaempchen/publication/228537991_Data_synchronization_strategies_for_multi-sensor_fusion/links/564050da08ae45b5d28d3ab6.pdf`.

[36] J. Nikolic, J. Rehder, M. Burri, P. Gohl, S. Leutenegger, P. T. Furgale, and R. Siegwart, “A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2014.

[37] P. Furgale, T. D. Barfoot, and G. Sibley, “Continuous-time batch estimation using temporal basis functions,” in *2012 IEEE International Conference on Robotics and Automation*, IEEE, May 2012.

[38] P. Furgale, J. Rehder, and R. Siegwart, “Unified temporal and spatial calibration for multi-sensor systems,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Nov. 2013.

[39] D. Galvez-López and J. D. Tardos, “Bags of binary words for fast place recognition in image sequences,” *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, Oct. 2012.

[40] M. Cummins and P. Newman, “FAB-MAP: Probabilistic localization and mapping in the space of appearance,” *The International Journal of Robotics Research*, vol. 27, no. 6, pp. 647–665, Jun. 2008.

[41] H. Strasdat, J. Montiel, and A. J. Davison, “Visual SLAM: Why filter?” *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, Feb. 2012.

[42] F. Kschischang, B. Frey, and H.-A. Loeliger, “Factor graphs and the sum-product algorithm,” *IEEE Transactions on Information Theory*, vol. 47, no. 2, pp. 498–519, 2001.

[43] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “G2o: A general framework for graph optimization,” in *2011 IEEE International Conference on Robotics and Automation*, IEEE, May 2011.

[44] S. Agarwal, K. Mierle, *et al.*, *Ceres solver*, `http://ceres-solver.org`.

[45] G. Klein and D. Murray, “Parallel tracking and mapping for small AR workspaces,” in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, IEEE, Nov. 2007.

[46] C. Forster, M. Pizzoli, and D. Scaramuzza, “SVO: Fast semi-direct monocular visual odometry,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2014.

[47] R. Mur-Artal and J. D. Tardos, “ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, Oct. 2016.

[48] T. Qin, P. Li, and S. Shen, “VINS-mono: A robust and versatile monocular visual-inertial state estimator,” *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 1004–1020, Aug. 2018.

[49] H. Liu, M. Chen, G. Zhang, H. Bao, and Y. Bao, “Ice-ba: Incremental, consistent and efficient bundle adjustment for visual-inertial slam,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1974–1982.

[50] J. Engel, T. Schöps, and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 834–849.

[51] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, “ORB-SLAM: A versatile and accurate monocular SLAM system,” *IEEE Transactions on Robotics*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[52] M. Kaess, A. Ranganathan, and F. Dellaert, “iSAM: Incremental smoothing and mapping,” *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, Dec. 2008.

[53] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, “iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering,” in *2011 IEEE International Conference on Robotics and Automation*, IEEE, May 2011.

[54] V. Ila, L. Polok, M. Solony, and P. Svoboda, “SLAM++ -a highly efficient and temporally scalable incremental SLAM framework,” *The International Journal of Robotics Research*, vol. 36, no. 2, pp. 210–230, Feb. 2017.

[55] A. P. Bustos, T.-J. Chin, A. Eriksson, and I. Reid, “Visual slam: Why bundle adjust?,” Feb. 11, 2019. arXiv: `http://arxiv.org/abs/1902.03747v1 [cs.CV]`.

[56] J. G. Mangelson, J. Liu, R. M. Eustice, and R. Vasudevan, “Guaranteed globally optimal planar pose graph and landmark slam via sparse-bounded sums-of-squares programming,” Sep. 20, 2018. arXiv: `http://arxiv.org/abs/1809.07744v1 [cs.RO]`.

[57] A. Allevato, *Converting between ros images and opencv images (c++)*, [Online; last edit 2017-04-20], Apr. 2017. [Online]. Available: `http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImages`

[58] G. Bradski and A. Kaehler, *Learning OpenCV*. O Reilly Media, Inc., 2008.

[59] M. Bjelonic, *YOLO ROS: Real-time object detection for ROS*, `https://github.com/leggedrobotics/darknet_ros`, 2016–2018.

[60] SharkD, *File:hsv color solid cylinder.png*, [Online; accessed June 13, 2019], 2018. [Online]. Available: `https://commons.wikimedia.org/wiki/File:HSV_color_solid_cylinder.png`.

[61] Glosser.ca, *File:colored neural network.svg*, [Online; accessed June 13, 2019], 2013. [Online]. Available: `https://commons.wikimedia.org/wiki/File:Colored_neural_network.svg`.

[62] S. J. Lee, T. Chen, L. Yu, and C. H. Lai, “Image Classification Based on the Boost Convolutional Neural Network,” *IEEE Access*, vol. 6, pp. 12 755–12 768, 2018.

[63] M. W. Gardner and S. R. Dorling, “Artificial neural networks (the multilayer perceptron) - a review of applications in the atmospheric sciences,” *Atmospheric Environment*, vol. 32, no. 14-15, pp. 2627–2636, 1998.

[64] Zadeh, Reza Bosagh and Ramsundar, Bharath, *Chapter 4. fully connected deep networks*, [Online; accessed June 13, 2019], [Online]. Available: `https://www.oreilly.com/library/view/tensorflow-for-deep/9781491980446/ch04.html`.

[65] M. R. M. Talabis, R. McPherson, I. Miyamoto, J. L. Martin, and D. Kaye, "Analytics Defined," *Information Security Analytics*, pp. 1–12, 2014.

[66] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *ImageNet Classification with Deep Convolutional Neural Networks*, pp. 1097–1105, 2012. [Online]. Available: `http://papers.nips.cc/paper/4824-imagenet-classification-w%5Cnpapers3://publication/uuid/1ECF396A-CEDA-45CD-9A9F-03344449DA2A`.

[67] Shaikh, Faizan, *Deep learning vs. machine learning the essential differences you need to know!* [Online; accessed June 13, 2019], 2017. [Online]. Available: `https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/`.

[68] TUM Wiki, *Image Semantic Segmentation*, [Online; accessed June 13, 2019], 2017. [Online]. Available: `https://wiki.tum.de/display/lfdv/Image+Semantic+Segmentation`.

[69] Tzutalin, *LabelImg*, Online; last accessed 03-May-2019, 2015. [Online]. Available: `https://github.com/tzutalin/labelImg`.

[70] S.-i. AMARI, *Neural Information Processing*, 823. Springer International Publishing, 2017, vol. 90, pp. 758–759. [Online]. Available: `http://dx.doi.org/10.1007/978-3-030-04212-7_32`.

[71] R. Girshick, J. Donahue, T. Darrell, U. C. Berkeley, and J. Malik, "(r-cnn) Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2–9, 2012. [Online]. Available: `http://ieeexplore.ieee.org/document/6909475/`.

[72] R. Girshick, "R-CNN_ICCV_2015_paper," *International Conference on Computer Vision*, pp. 1440–1448, 2015.

[73] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks Shaoqing," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017.

[74] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem, pp. 779–788, 2016.

[75] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017.

[76] ——, "YOLOv3: An Incremental Improvement," 2018. [Online]. Available: `http://arxiv.org/abs/1804.02767`.

[77] J. Redmon, *Darknet: Open source neural networks in c*, `http://pjreddie.com/darknet/`, 2013–2016.

[78] G. P. Zhang, "Neural networks for classification: a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 30, no. 4, pp. 451–462, 2000.

[79] J.-c. Latombe, *Robot motion plannig*, 9. Springer Science + Business Media, LLC, 1991, vol. 53, pp. 58–105.

[80] S. M. LaValle, "Planning algorithms," *Planning Algorithms*, vol. 9780521862, pp. 1–826, 2006.

[81] S. M. Lavalle, “Motion planning: Part I: The essentials,” *IEEE Robotics and Automation Magazine*, vol. 18, no. 1, pp. 79–89, 2011.

[82] A. Tsourdos, B. White, and M. Shanmugavel, *Cooperative Path Planning of Unmanned Aerial Vehicles Cooperative Path Planning of Unmanned Aerial Vehicles Aerospace Series List Design and Analysis of Composite Structures: With Applications to Aerospace Structures.* 2011.

[83] H. K. Khalil, *Nonlinear Systems*, 3rd. Prentice Hall, 2002.

[84] F. A. Leve, B. J. Hamilton, and M. A. Peck, *Spacecraft momentum control systems.* 2015, pp. 1–247.

[85] T. W. Gamelin and R. E. Greene, *Introduction to Topology*, 2nd. Mineola, N.Y.: Drover Publications, 1999, p. 234.

[86] Shonk, “Special Groups and Projective Planes,” Tech. Rep. [Online]. Available: `http://www.sellingwaves.com/projplane.pdf`.

[87] M. do Carmo, M. Ritoré, and A. Ros, *Compact minimal hypersurfaces with index one in the real projective space.* Springer, 2012, pp. 407–414.

[88] ProofWiki.com, *Projective Space*, 2017. [Online]. Available: `https://proofwiki.org/wiki/Definition:Projective_Space`.

[89] A. Lekkas and T. I. Fossen, *Introduction to Path Planning, Properties of Curves, Dubins Paths and Clothoids - TTK8190 Lecture Notes (NTNU).*

[90] D. Devaurs, T. Simeon, and J. Cortés, “Optimal Path Planning in Complex Cost Spaces With Sampling-Based Algorithms,” *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 415–424, 2016. [Online]. Available: `https://hal.archives-ouvertes.fr/hal-01231482`.

[91] J. T. Schwartz and M. Sharir, “On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds,” *Advances in Applied Mathematics*, vol. 4, no. 3, pp. 298–351, 1983.

[92] T. Lozano-Pérez and M. A. Wesley, “An algorithm for planning collision-free paths among polyhedral obstacles,” *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979. [Online]. Available: `http://portal.acm.org/citation.cfm?doid=359156.359164`.

[93] T. Lozano-Perez, “Spatial Planning: A Configuration Space Approach,” *Computers, IEEE Transactions on*, vol. c, no. 2, pp. 108–120, 1983. [Online]. Available: `http://lis.csail.mit.edu/pubs/tlp/spatial-planning.pdf`.

[94] B. Chazelle, *Approximation and Decomposition of Shapes*, 1987. [Online]. Available: `https://www.cs.princeton.edu/~chazelle/pubs/ApproxDecompShapes.pdf`.

[95] J. H. Reif, “Complexity of the mover’s problem and generalizations,” *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pp. 421–427, 1979. [Online]. Available: `http://ieeexplore.ieee.org/document/4568037/`.

[96] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, “On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE-Hardness of the "Warehouseman’s Problem",” Tech. Rep. [Online]. Available: `http://journals.sagepub.com/doi/pdf/10.1177/027836498400300405`.

[97] J. F. Canny, *The Complexity of Robot Motion Planning.* 1988, vol. Doctoral D, p. 195. [Online]. Available: `https://ia801604.us.archive.org/21/items/TheComplexityOfRobotMotionPlanning/The%20Complexity%20of%20Robot%20Motion%20Planning.pdf%20http://books.google.com/books?hl=en&lr=&id=_VRM_sczrKgC&pgis=1`.

[98] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *The International Journal of Robotics Research*, vol. 5, no. 1, 1986.

[99] S. Campbell, W. Naeem, and G. W. Irwin, "A review on improving the autonomy of unmanned surface vehicles through intelligent collision avoidance manoeuvres," *Annual Reviews in Control*, vol. 36, no. 2, pp. 267–283, 2012. [Online]. Available: `http://dx.doi.org/10.1016/j.arcontrol.2012.09.008`.

[100] M. Imran and F. Kunwar, "A hybrid path planning technique developed by integrating global and local path planner," *2016 International Conference on Intelligent Systems Engineering, ICISE 2016*, pp. 118–122, 2016.

[101] A. M. Lekkas, A. L. Roald, and M. Breivik, "Online Path Planning for Surface Vehicles Exposed to Unknown Ocean Currents Using Pseudospectral Optimal Control," *IFAC-PapersOnLine*, vol. 49, no. 23, pp. 1–7, 2016. [Online]. Available: `https://linkinghub.elsevier.com/retrieve/pii/S2405896316319000`.

[102] P. Tompkins, A. Stentz, and D. Wettergreen, "Global path planning for Mars rover exploration," *IEEE Aerospace Conference Proceedings*, vol. 2, pp. 801–814, 2004.

[103] D. S. Rao and S. B. Williams, *Large-scale path planning for Underwater Gliders in ocean currents*, 2009. [Online]. Available: `https://www.semanticscholar.org/paper/Large-scale-path-planning-for-Underwater-Gliders-in-Rao-Williams/2d5c2dae59cee6a2664148026c5d1b768fa7075c`.

[104] Z. Zeng, K. Sammut, L. Lian, F. He, A. Lammas, and Y. Tang, "A comparison of optimization techniques for AUV path planning in environments with ocean currents," *Robotics and Autonomous Systems*, vol. 82, pp. 61–72, 2016. [Online]. Available: `http://dx.doi.org/10.1016/j.robot.2016.03.011`.

[105] F. Sadeghi and S. Levine, "CAD2RL: Real Single-Image Flight without a Single Real Image," Nov. 2016. [Online]. Available: `http://arxiv.org/abs/1611.04201`.

[106] A. H. Qureshi, M. J. Bency, and M. C. Yip, "Motion Planning Networks," Tech. Rep. [Online]. Available: `https://sites.google.com/view/mpnet/home.`.

[107] F. Engelmann, T. Kontogianni, A. Hermans, and B. Leibe, "Exploring Spatial Context for 3D Semantic Segmentation of Point Clouds," Feb. 2018. [Online]. Available: `http://arxiv.org/abs/1802.01500%20http://dx.doi.org/10.1109/ICCVW.2017.90`.

[108] L. E. Kavraki, "RANDOM NETWORKS IN CONFIGURATION SPACE FOR FAST PATH PLANNING," Tech. Rep., 1994. [Online]. Available: `http://i.stanford.edu/pub/cstr/reports/cs/tr/95/1535/CS-TR-95-1535.pdf`.

[109] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[110] A.-a. Agha-mohammadi, S. Chakravorty, and N. M. Amato, "Sampling-based nonholonomic motion planning in belief space via Dynamic Feedback Linearization-based FIRM," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Oct. 2012, pp. 4433–4440. [Online]. Available: `http://ieeexplore.ieee.org/document/6385970/`.

[111] P. Corke, *Robotics, Vision and Control*, ser. Springer Tracts in Advanced Robotics. Cham: Springer International Publishing, 2017, vol. 118. [Online]. Available: `http://link.springer.com/10.1007/978-3-319-54413-7`.

[112] K. Solovey and M. Kleinbort, "The Critical Radius in Sampling-based Motion Planning *," Tech. Rep., 2018. [Online]. Available: `https://arxiv.org/pdf/1709.06290.pdf`.

[113] A. Elfes, “Using Occupancy Grids for Mobile Robot Perception and Navigation,” *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

[114] R. R. Murphy, *AI Robotics*. 2000, p. 466. [Online]. Available: `http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Ai+robotics#5%5Cnhttp://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:AI+Robotics%235`.

[115] D. Meagher, “Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer,” *Rensselaer Polytechnic Institute*, no. Technical Report IPL-TR-80-111, 1980.

[116] M. Seda, “Roadmap Methods vs. Cell Decomposition in Robot Motion Planning,” *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation*, pp. 127–132, 2007. [Online]. Available: `https://pdfs.semanticscholar.org/d4b6/2b89acbf9eb50782f04c4b38c47cac515dbe.pdf`.

[117] M. De Berg, O. Cheong, M. Van Kreveld, and M. Overmars, *Computational geometry: Algorithms and applications*. 2008, pp. 1–386.

[118] M. Candeloro, A. M. Lekkas, A. J. Sørensen, and T. I. Fossen, “Continuous Curvature Path Planning using Voronoi diagrams and Fermat’s spirals,” *IFAC Proceedings Volumes*, vol. 46, pp. 132–137, 2013. [Online]. Available: `https://ac.els-cdn.com/S147466701646146X/1-s2.0-S147466701646146X-main.pdf?_tid=13204da2-b21c-4d05-a4b9-325f8b84955e&acdnat=1543911094_a861ac33fcc8a82a74cdb71f03c3bd58`.

[119] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls,” *ACM Transactions on Mathematical Software*, vol. 22, no. 4, pp. 469–483, Dec. 1996. [Online]. Available: `http://dpd.cs.princeton.edu/Papers/BarberDobkinHuhdanpaa.pdf`.

[120] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, “Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning,” no. Section III, 2018. [Online]. Available: `http://arxiv.org/abs/1803.04345`.

[121] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees,” *Autonomous Robots*, 2013. [Online]. Available: `http://octomap.github.com.`.

[122] M. Candeloro, A. M. Lekkas, and A. J. Sørensen, “A Voronoi-diagram-based dynamic path-planning system for underactuated marine vessels,” *Control Engineering Practice*, vol. 61, no. February, pp. 41–54, 2017. [Online]. Available: `http://dx.doi.org/10.1016/j.conengprac.2017.01.007`.

[123] M. Candeloro, A. M. Lekkas, J. Hegde, and A. J. Sorensen, “A 3D dynamic Voronoi diagram-based path-planning system for UUVs,” *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*, 2016.

[124] J. Y. Yen, “An algorithm for finding shortest routes from all source nodes to a given destination in general networks,” *Quarterly of Applied Mathematics*, vol. 27, no. 4, pp. 526–530, 1970. [Online]. Available: `http://www.ams.org/qam/1970-27-04/S0033-569X-1970-0253822-7/`.

[125] M. Ester, K. Hans-Peter, S. Jorg, and X. Xiaowei, “Density-Based Clustering Algorithms for Discovering Clusters,” *Comprehensive Chemometrics*, vol. 2, pp. 635–654, 2010.

[126] D. R. Canelhas, “Truncated Signed Distance Fields Applied To Robotics,” PhD thesis, Örebro University, 2017. [Online]. Available: `www.publications.oru.se`.

[127] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, “Safe Local Exploration for Replanning in Cluttered Unknown Environments for Micro-Aerial Vehicles,” 2017. [Online]. Available: `http://arxiv.org/abs/1710.00604%0Ahttp://dx.doi.org/10.1109/LRA.2018.2800109`.

[128] S. M. LaValle, “Rapidly-Exploring Random Trees: A New Tool for Path Planning,” *In*, vol. 129, pp. 98–11, 1998. [Online]. Available: `http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Rapidly-exploring+random+trees:+A+new+tool+for+path+planning#0`.

[129] S. Karaman and E. Frazzoli, “Sampling-based Algorithms for Optimal Motion Planning,” pp. 1–76, 2011. [Online]. Available: `http://arxiv.org/abs/1105.1186`.

[130] S. M. LaValle and J. J. Kuffner, “Rapidly-exploring random trees: Progress and prospects,” *4th Workshop on Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2000. [Online]. Available: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.1387`.

[131] I. Noreen, A. Khan, and Z. Habib, “Optimal Path Planning using RRT * based Approaches : A Survey and Future Directions,” *(IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 7, no. 11, pp. 97–107, 2016. [Online]. Available: `www.ijacsa.thesai.org`.

[132] L. Yang, J. Qi, D. Song, J. Xiao, J. Han, and Y. Xia, “Survey of Robot 3D Path Planning Algorithms,” *Journal of Control Science and Engineering*, vol. 2016, pp. 1–22, Jul. 2016. [Online]. Available: `http://www.hindawi.com/journals/jcse/2016/7426913/`.

[133] M. Elbanhawi and M. Simic, “Sampling-Based Robot Motion Planning: A Review,” *IEEE Access*, vol. 2, pp. 56–77, 2014. [Online]. Available: `http://ieeexplore.ieee.org/document/6722915/`.

[134] S. Karaman and E. Frazzoli, “Optimal kinodynamic motion planning using incremental sampling-based methods,” in *49th IEEE Conference on Decision and Control (CDC)*, IEEE, Dec. 2010, pp. 7681–7687. [Online]. Available: `http://ieeexplore.ieee.org/document/5717430/`.

[135] O. Adiyatov and H. A. Varol, “Rapidly-exploring random tree based memory efficient motion planning,” in *2013 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2013*, 2013, pp. 354–359.

[136] ——, “A novel RRTstar-based algorithm for motion planning in Dynamic environments,” in *2017 IEEE International Conference on Mechatronics and Automation, ICMA 2017*, IEEE, 2017, pp. 1416–1421.

[137] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic,” Tech. Rep. [Online]. Available: `https://arxiv.org/pdf/1404.2334.pdf`.

[138] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer, “Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning,” in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2016, pp. 4207–4214. [Online]. Available: `http://ieeexplore.ieee.org/document/7487615/`.

[139] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, “CHOMP: Covariant Hamiltonian optimization for motion planning,” *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, Aug. 2013. [Online]. Available: `http://journals.sagepub.com/doi/10.1177/0278364913488805`.

[140] R. B. Cattell, “The description of personality: basic traits resolved into clusters.,” *The Journal of Abnormal and Social Psychology*, vol. 38, no. 4, pp. 476–506, 1943. [Online]. Available: `http://doi.apa.org/getdoi.cfm?doi=10.1037/h0054116`.

[141] J. MacQueen, “Some Methods for Classification and Analysis of Multivariate Observations,” in *5th Berkeley Symposium on Mathematical Statistics and Probability, Vol. 1*, University of California Press, 1967, pp. 281–297.

[142] S. Lloyd, "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, no. 2, pp. 129–137, Mar. 1982. [Online]. Available: `http://ieeexplore.ieee.org/document/1056489/`.

[143] H. Steinhaus, "Sur la division des corps materiels en parties," *Bulletin of the Polish Academy of Sciences*, vol. 4, no. 3, pp. 801–804, 1956. [Online]. Available: `http://www.laurent-duval.eu/Documents/Steinhaus_H_1956_j-bull-acad-polon-sci_division_cmp-k-means.pdf`.

[144] R. M. Esteves, T. Hacker, and C. Rong, "Competitive K-Means, a New Accurate and Distributed K-Means Algorithm for Large Datasets," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, IEEE, Dec. 2013, pp. 17–24. [Online]. Available: `http://ieeexplore.ieee.org/document/6753773/`.

[145] H. Ng, S. Ong, K. Foong, P. Goh, and W. Nowinski, "Medical Image Segmentation Using K-Means Clustering and Improved Watershed Algorithm," in *2006 IEEE Southwest Symposium on Image Analysis and Interpretation*, IEEE, pp. 61–65. [Online]. Available: `http://ieeexplore.ieee.org/document/1633722/`.

[146] D. Xu and Y. Tian, "A Comprehensive Survey of Clustering Algorithms," *Annals of Data Science*, vol. 2, no. 2, pp. 165–193, Jun. 2015. [Online]. Available: `http://link.springer.com/10.1007/s40745-015-0040-1`.

[147] F. Imeson and S. L. Smith, "Clustering in discrete path planning for approximating minimum length paths," in *2017 American Control Conference (ACC)*, IEEE, May 2017, pp. 2968–2973. [Online]. Available: `http://ieeexplore.ieee.org/document/7963402/`.

[148] O. Arslan, D. P. Gulranik, and D. E. Koditschek, "Clustering-Based Robot Navigation and Control," no. May, 2016. [Online]. Available: `http://kodlab.seas.upenn.edu/uploads/Main/arslan_guralnik_kod_ICRA2016TopologyWorkshop.pdf`.

[149] L. E. Dubins, "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, vol. 79, no. 3, p. 497, Jul. 1957. [Online]. Available: `https://www.jstor.org/stable/2372560?origin=crossref`.

[150] F. Lamiraux and J.-P. Lammond, "Smooth motion planning for car-like vehicles," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 498–501, 2001. [Online]. Available: `http://ieeexplore.ieee.org/document/954762/`.

[151] A. M. Lekkas and T. I. Fossen, "Integral LOS Path Following for Curved Paths Based on a Monotone Cubic Hermite Spline Parametrization," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2287–2301, Nov. 2014. [Online]. Available: `http://ieeexplore.ieee.org/document/6767080/`.

[152] P. Costantini, T. N. T. Goodman, and C. Manni, "Constructing C 3 shape preserving interpolating space curves," Tech. Rep., 2001, pp. 103–127. [Online]. Available: `https://link.springer.com/content/pdf/10.1023%2FA%3A1016664630563.pdf`.

[153] A. M. Lekkas, A. R. Dahl, M. Breivik, and T. I. Fossen, "Continuous-Curvature Path Generation Using Fermat's Spiral," vol. 34, no. 4, pp. 183–198, 2013. [Online]. Available: `http://www.mic-journal.no/PDF/2013/MIC-2013-4-3.pdf%20http://www.mic-journal.no/ABS/MIC-2013-4-3.asp`.

[154] A. Cosgun and H. I. Christensen, "Context-aware robot navigation using interactively built semantic maps," Tech. Rep., 2018. [Online]. Available: `https://arxiv.org/pdf/1710.08682.pdf`.

[155] L. Carlone, J. Du, M. Kaouk Ng, B. Bona, and M. Indri, "Active SLAM and exploration with particle filters using Kullback-Leibler divergence," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 75, no. 2, pp. 291–311, Aug. 2014. [Online]. Available: `http://link.springer.com/10.1007/s10846-013-9981-9`.

[156] M. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 304–312, Jun. 1992. [Online]. Available: `http://ieeexplore.ieee.org/document/143349/`.

[157] C. H., *Robot Motion Planning: Bug Algorithms*, Online; last accessed 29-April-2019, 2007. [Online]. Available: `https://www.cs.cmu.edu/~motionplanning/lecture/Chap2-Bug-Alg_howie.pdf`.

[158] B. Yamauchi, "A frontier-based exploration for autonomous exploration," *IEEE International Symposium on Computational Intelligence in Robotics and Automation, Monterey, CA*, pp. 146–151, 1997.

[159] S. D., F. T., F. M., and W. M., *turtlebot ROS package*, `http://wiki.ros.org/turtlebot`, Online; last accessed 29 April 2019, 2018.

[160] L. Murphy and P. Newman, "Using incomplete online metric maps for topological exploration with the Gap Navigation Tree," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, May 2008, pp. 2792–2797. [Online]. Available: `http://ieeexplore.ieee.org/document/4543633/`.

[161] A. Makarenko, S. Williams, F. Bourgault, and H. Durrant-Whyte, "An experiment in integrated exploration," in *IEEE/RSJ International Conference on Intelligent Robots and System*, vol. 1, IEEE, pp. 534–539. [Online]. Available: `http://ieeexplore.ieee.org/document/1041445/`.

[162] A. Zelinsky, "Using Path Transforms to Guide the Search for Findpath in 2D," *The International Journal of Robotics Research*, vol. 13, no. 4, pp. 315–325, Aug. 1994. [Online]. Available: `http://journals.sagepub.com/doi/10.1177/027836499401300403`.

[163] S. Wirth and J. Pellenz, "Exploration transform: A stable exploring algorithm for robots in rescue environments," *SSRR2007 - IEEE International Workshop on Safety, Security and Rescue Robotics Proceedings*, 2007.

[164] P. Norvig and S. J. Russel, *Artificial intelligence A modern approach*, 3rd. Harlow, UK: Pearson Education Ltd., Mar. 2010. arXiv: `9809069v1 [gr-qc]`. [Online]. Available: `http://www.journals.cambridge.org/abstract_S0269888900007724`.

[165] R. Valencia and J. Andrade-cetto, *Springer Tracts in Advanced Robotics 119 Mapping , Planning and Exploration with Pose SLAM*, B. Siciliano and O. Khatib, Eds. Springer International Publishing, 2018, pp. 53–98.

[166] T. Tao, Y. Huang, F. Sun, and T. Wang, "Motion planning for SLAM based on frontier exploration," *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation, ICMA 2007*, no. 60605021, pp. 2120–2125, 2007.

[167] C. Zhu, R. Ding, M. Lin, and Y. Wu, "A 3D Frontier-Based Exploration Tool for MAVs," in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, Nov. 2015, pp. 348–352. [Online]. Available: `http://ieeexplore.ieee.org/document/7372156/`.

[168] D. Priyasad, Y. Jayasanka, H. Udayanath, D. Jayawardhana, S. Sooriyaarachchi, C. Gamage, and N. Kottege, "Point Cloud Based Autonomous Area Exploration Algorithm," in *2018 Moratuwa Engineering Research Conference (MERCon)*, IEEE, May 2018, pp. 318–323. [Online]. Available: `https://ieeexplore.ieee.org/document/8421954/`.

[169] S. S. Belavadi, R. Beri, and V. Malik, "Frontier Exploration Technique for 3D Autonomous SLAM Using K-Means Based Divisive Clustering," in *2017 Asia Modelling Symposium (AMS)*, IEEE, Dec. 2017, pp. 95–100. [Online]. Available: `https://ieeexplore.ieee.org/document/8424313/`.

[170] D. Holz, S. Behnke, N. Basilico, and F. Amigoni, "Evaluating the Efficiency of Frontier-based Exploration Strategies," *ISR/Robotik*, no. 2, p. 8, 2010. [Online]. Available: `https://www.vde-verlag.de/proceedings-en/453273006.html`.

[171] A. Bircher, M. Kamel, K. Alexis, H. Oleynikova, and R. Siegwart, "Receding Horizon &quot;Next-Best-View&quot; Planner for 3D Exploration," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2016, pp. 1462–1468. [Online]. Available: `http://ieeexplore.ieee.org/document/7487281/`.

[172] M. Selin, M. Tiger, D. Duberg, F. Heintz, and P. Jensfelt, "Efficient Autonomous Exploration Planning of Large-Scale 3-D Environments," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1699–1706, Apr. 2019. [Online]. Available: `https://ieeexplore.ieee.org/document/8633925/`.

[173] E. Vidal, J. D. Hernandez, K. Istenic, and M. Carreras, "Optimized Environment Exploration for Autonomous Underwater Vehicles," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2018, pp. 6409–6416. [Online]. Available: `https://ieeexplore.ieee.org/document/8460919/`.

[174] R. Nuredini, B. Fetaji, and I. Chorbev, "Bio-inspired Obstacle Avoidance: From Animals to Intelligent Agents," vol. 13, no. 2, pp. 146–153, 2017.

[175] B. Sun, D. Zhu, and S. X. Yang, "An Optimized Fuzzy Control Algorithm for Three-Dimensional AUV Path Planning," *International Journal of Fuzzy Systems*, vol. 20, no. 2, pp. 597–610, 2018.

[176] J. Witt and M. Dunbabin, "Go with the Flow: Optimal Path Planning in Coastal Environments," *Proceedings of the Australasian Conference on Robotics & Automation (ACRA)*, pp. 1–9, 2008.

[177] A. Hussain Qureshi and Y. Ayaz, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments *," Tech. Rep., 2017. [Online]. Available: `https://arxiv.org/pdf/1703.08944.pdf`.

[178] M. Candeloro, A. M. Lekkas, A. J. Sørensen, and T. I. Fossen, *Continuous curvature path planning using voronoi diagrams and Fermat's spirals*, PART 1. IFAC, 2013, vol. 9, pp. 132–137. [Online]. Available: `http://dx.doi.org/10.3182/20130918-4-JP-3022.00064`.

[179] K. Takaya, T. Asai, V. Kroumov, and F. Smarandache, "Simulation environment for mobile robots testing using ROS and Gazebo," in *2016 20th International Conference on System Theory, Control and Computing, ICSTCC 2016 - Joint Conference of SINTES 20, SACCS 16, SIMSIS 20 - Proceedings*, 2016.

[180] D. Arthur and S. Vassilvitskii, "k-means++: The Advantages of Careful Seeding," Tech. Rep. [Online]. Available: `http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf`.

[181] D. Radu and B. Rusu, "Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments," Tech. Rep. [Online]. Available: `http://mediatum.ub.tum.de/doc/800632/941254.pdf`.

[182] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[183] S. M. Omohundro, “Five Balltree Construction Algorithms,” *Bulletin of Mathematical Biology*, vol. 51, no. 1, pp. 39–54, 1989. [Online]. Available: `http://www.springerlink.com/index/10.1016/S0092-8240(89)80047-3`.

[184] K. Klasing, D. Althoff, D. Wollherr, and M. Buss, “Comparison of surface normal estimation methods for range sensing applications,” in *2009 IEEE International Conference on Robotics and Automation*, IEEE, May 2009, pp. 3206–3211. [Online]. Available: `http://ieeexplore.ieee.org/document/5152493/`.

[185] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, “Surface Reconstruction from Unorganized Points,” in *SIGGRAPH ’92 Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, New York: ACM, 1992, pp. 71–78.

[186] C. Shakarji, “Least-squares fitting algorithms of the NIST algorithm testing system,” *Journal of Research of the National Institute of Standards and Technology*, vol. 103, no. 6, p. 633, 2012.

[187] M. Danielczuk, M. Matl, S. Gupta, A. Li, A. Lee, J. Mahler, and K. Goldberg, “Segmenting Unknown 3D Objects from Real Depth Images using Mask R-CNN Trained on Synthetic Data,” Sep. 2018. [Online]. Available: `http://arxiv.org/abs/1809.05825`.

[188] T. M. Chan, “A Minimalist’s Implementation of the 3-d Divide-and-Conquer Convex Hull Algorithm,” Tech. Rep., 2003. [Online]. Available: `http://www.cs.uwaterloo.ca/~tmchan/..`

[189] R. Seidel, “A convex hull algorithm optimal for point sets in even dimensions,” 1981. [Online]. Available: `https://open.library.ubc.ca/cIRcle/collections/ubctheses/831/items/1.0051821`.

[190] S. Hert and S. Schirra, “3D convex hulls,” in *CGAL User and Reference Manual*, 4.13, CGAL Editorial Board, 2018. [Online]. Available: `https://doc.cgal.org/4.13/Manual/packages.html#PkgConvexHull3Summary`.

[191] S. Suri, “Lifting Transform, Voronoi, Delaunay, Convex Hulls,” pp. 1–5,

[192] I. A. Şucan, M. Moll, and L. E. Kavraki, “The Open Motion Planning Library,” *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, Dec. 2012, `http://ompl.kavrakilab.org`.

[193] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, “Batch Informed Trees (BIT*): Sampling-based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs,” Tech. Rep. [Online]. Available: `https://www.ri.cmu.edu/pub_files/2015/5/Gammell15-bitstar.pdf`.

[194] A. M. Shkel and V. Lumelsky, “Classification of the Dubins set,” *Robotics and Autonomous Systems*, vol. 34, no. 4, pp. 179–202, Mar. 2001. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0921889000001275`.

[195] R. W. Beard and T. W. Mclain, “Implementing Dubins Airplane Paths on Fixed-wing UAVs,” in *Handbook for Unmanned Aerial Vehicles*, Springer, 2013, pp. 5–15.

[196] D. Schneider, “Master Thesis Path Planning for Fixed-Wing Unmanned Aerial Vehicles,” 2016.

[197] Y. Lin and S. Saripalli, “Path planning using 3D Dubins Curve for Unmanned Aerial Vehicles,” in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, May 2014, pp. 296–304. [Online]. Available: `http://ieeexplore.ieee.org/document/6842268/`.

[198] J. Pan, S. Chitta, and D. Manocha, “FCL: A general purpose library for collision and proximity queries,” in *2012 IEEE International Conference on Robotics and Automation*, IEEE, May 2012, pp. 3859–3866. [Online]. Available: `http://ieeexplore.ieee.org/document/6225337/`.

[199] E. Weiszfeld and F. Plastria, “On the point for which the sum of the distances to n given points is minimum,” *Annals of Operations Research*, vol. 167, no. 1, pp. 7–41, Mar. 2009. [Online]. Available: `http://link.springer.com/10.1007/s10479-008-0352-z`.

[200] M. B. Cohen, Y. T. Lee, G. Miller, J. Pachocki, and A. Sidford, “Geometric Median in Nearly Linear Time,” Jun. 2016. [Online]. Available: `http://arxiv.org/abs/1606.05225`.

[201] N. Koenig and A. Howard, “Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator,” Tech. Rep. [Online]. Available: `http://playerstage.sourceforge.net/gazebo/`.

[202] S. Moe, W. Caharija, K. Y. Pettersen, and I. Schjolberg, “Path following of underactuated marine surface vessels in the presence of unknown ocean currents,” eng, American Automatic Control Council, 2014, pp. 3856–3861.

[203] M. Breivik and T. I. Fossen, “Principles of guidance-based path following in 2d and 3d,” in *Proceedings of the 44th IEEE Conference on Decision and Control*, Dec. 2005, pp. 627–634.

[204] I.-L. Borlaug, K. Pettersen, and J. Gravdahl, “Trajectory tracking for an articulated intervention auv using a super-twisting algorithm in 6 dof,” eng, *IFAC PapersOnLine*, vol. 51, no. 29, pp. 311–316, 2018.

[205] S. Moe, G. Antonelli, A. R. Teel, K. Y. Pettersen, and J. Schrimpf, “Set-based tasks within the singularity-robust multiple task-priority inverse kinematics framework: General formulation, stability analysis, and experimental results,” *Frontiers in Robotics and AI*, vol. 3, p. 16, 2016. [Online]. Available: `https://www.frontiersin.org/article/10.3389/frobt.2016.00016`.

[206] C. Samson, “Path following and time-varying feedback stabilization of a wheeled mobile robot,” 1992.

[207] M. Breivik and T. I. Fossen, “Path following for marine surface vessels,” in *Oceans ’04 MTS/IEEE Techno-Ocean ’04 (IEEE Cat. No.04CH37600)*, vol. 4, Nov. 2004, 2282–2289 Vol.4.

[208] W. Caharija, K. Y. Pettersen, M. Bibuli, P. Calado, E. Zereik, J. Braga, J. T. Gravdahl, A. J. S rensen, M. Milovanovi, and G. Bruzzone, “Integral line-of-sight guidance and control of underactuated marine vehicles: Theory, simulations, and experiments,” *IEEE Transactions on Control Systems Technology*, vol. 24, no. 5, pp. 1623–1642, Sep. 2016.

[209] R. Skjetne, T. I. Fossen, and P. Kokotovi, “Output maneuvering for a class of nonlinear systems,” *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 501–506, 2002, 15th IFAC World Congress. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S1474667015386663`.

[210] R. T. Farouki, C. Giannelli, M. L. Sampoli, and A. Sestini, “Rotation-minimizing osculating frames,” *Computer Aided Geometric Design*, vol. 31, no. 1, pp. 27–42, 2014. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0167839613001003`.

[211] H. Guggenheimer, “Computing frames along a trajectory,” *Computer Aided Geometric Design*, vol. 6, no. 1, pp. 77–78, 1989. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/0167839689900083`.

[212] O. Khatib, “A unified approach for motion and force control of robot manipulators: The operational space formulation,” *IEEE Journal on Robotics and Automation*, vol. 3, no. 1, pp. 43–53, Feb. 1987.

[213] A. Loría and E. Panteley, “2 cascaded nonlinear time-varying systems: Analysis and design,” in *Advanced Topics in Control Systems Theory: Lecture Notes from FAP 2004*, F. Lamnabhi-Lagarrigue, A. Loría, and E. Panteley, Eds. London: Springer London, 2005, pp. 23–64. [Online]. Available: `https://doi.org/10.1007/11334774_2`.

[214] G. Guennebaud, B. Jacob, *et al.*, *Eigen v3*, http://eigen.tuxfamily.org, 2010.

[215] J. G. Balchen, *Reguleringsteknikk*, nor, Trondheim, 2016.

[216] A. Kim and R. M. Eustice, “Real-time visual SLAM for autonomous underwater hull inspection using visual saliency,” *IEEE Transactions on Robotics*, vol. 29, no. 3, pp. 719–733, Jun. 2013.

[217] H. A. Daoud, A. Q. M. Sabri, C. K. Loo, and A. M. Mansoor, “SLAMM: Visual monocular SLAM with continuous mapping using multiple maps,” *PLOS ONE*, vol. 13, no. 4, A. Agudo, Ed., e0195878, Apr. 2018.

[218] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” Tech. Rep. [Online]. Available: `http://lmb.informatik.uni-freiburg.de/`.

[219] D. Maturana and S. Scherer, “VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition,” Tech. Rep. [Online]. Available: `https://www.ri.cmu.edu/pub_files/2015/9/voxnet_maturana_scherer_iros15.pdf`.

[220] Z. Wu, S. Song, A. Khosla, Y. Fisher, L. Zhang, X. Tang, and J. Xiao, “3D ShapeNets: A Deep Representation for Volumetric Shapes,” Tech. Rep. [Online]. Available: `http://3DShapeNets.cs.princeton.edu`.

[221] H. Su, S. Maji, E. Kalogerakis, and E. Learned-Miller, “Multi-view Convolutional Neural Networks for 3D Shape Recognition,” Tech. Rep. [Online]. Available: `http://vis-www.cs.umass.edu/mvcnn.`.

[222] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation,” Tech. Rep. [Online]. Available: `https://arxiv.org/pdf/1612.00593.pdf`.

[223] L. Landrieu and M. Simonovsky, “Large-scale Point Cloud Semantic Segmentation with Superpoint Graphs,” Tech. Rep. [Online]. Available: `https://arxiv.org/pdf/1711.09869.pdf`.

Appendices

A Simulator Environment

To test planning and exploration implementations, as well as to run more interconnected system experiments on selected operational cases, a simulator environment was required. The use of a simulator lessens the need for tedious practical tests and allows for expeditious evaluation of modules and bug fixing. To this end, the *UUV simulator* [20] was used. This simulator includes the implementation of the equations of motion for underwater vehicles [15], thruster allocation modules, several controllers, and incorporates lift, drag, and current simulations. The simulator is made publicly available as a research prototype, and is under continuous development.

The UUV simulator is implemented as a set of Gazebo plugins and ROS packages. Gazebo is a robotics simulator capable of simulating complex 3D environments and can perform extensive dynamic interaction between objects. These environments are defined using *world* files which describes the objects, robots, and global parameters (such as physics properties) present in the simulation. To allow for model reuse, and open source distribution, simulated objects are stored as *model* files, using the *simulated description format* (.sdf). These files include the information of the specific model and its 3D rendering data.

Being based on Gazebo, the UUV simulator makes use of the supported physics engines, such as ODE and Bullet, and the environment and robot model rendering capabilities. This allows for the use of other open source simulator resources, such as pregenerated 3D environments and more specific sensor plugins. One such sensor plugin that was used in this thesis to emulate an idealized SLAM scenario, was the image sonar plugin created by the Swedish maritime robotics centre (SMaRC). This plugin is available on their Git repo.[1], but have since been merged with the UUV simulator [2].

Due to this simulator, essentially, being a ROS package, it can be seamlessly integrated into a ROS project. This allows for running a complete simulation scenario using a single ROS *launch* file written in the XML format[3]. A simple example launch file, showing the essentials, is presented in listing 1. The default robot model setup can be customized by

[1]SMaRC Gazebo plugins: `https://github.com/smarc-project/smarc_simulations/tree/master/smarc_gazebo_plugins`

[2]UUV Simulator: `https://github.com/uuvsimulator/uuv_simulator`

[3]See `http://wiki.ros.org/ROS/Tutorials` for a more complete introduction to ROS and its components.

modifying a xacro file[4] defining its sensors, links, and 3D model. An example xacro file describing a robot with a center-mounted IMU and a tilted camera and image sonar is presented in listing 2. This is an example of a file loaded by the "upload_default.launch" call on line 6 in listing 1. Thus, specialized robot models can be included and set up, allowing for the simulation of different operations and scenarios.

[4]Xacro (XML Macro) files contains XML macro code, constructing shorter and more readable XML files through the use of macros.

Listing 1: Example launch file launching a Gazebo environment, an UUV robot object, an OctoMap server, and RViz for visualization.

```
<launch>
   <!-- launch Gazebo world -->
   <include file="$(find smarc_worlds)/launch/pipe_world.launch" />

   <!-- Add the RexROV vehicle to the simulation (namespace: rexrov) -->
   <include file="$(find uuv_descriptions)/launch/upload_rexrov_default.launch">
       <arg name="mode" value="sonar"/>
       <arg name="x" default="0"/>
       <arg name="y" default="-78"/>
       <arg name="z" default="-88"/>
       <arg name="yaw" default="-0.25"/>
   </include>

   <!-- launch octomap server -->
   <node pkg="octomap_server" type="octomap_server_node" name="octomap_server">
       <param name="resolution" value="0.5" />
       <param name="frame_id" type="string" value="world" />
       <!-- True for static map, false if no initial map -->
       <param name="latch" value="false" />
       <!-- maximum range to integrate (speedup!) -->
       <param name="sensor_model/max_range" value="7.0" />
       <!-- data source to integrate (PointCloud2) -->
       <remap from="cloud_in" to="/rexrov/points" />
       <param name="publish_free_space" value="true" />
   </node>

   <!-- Open RViz for visualization of sensor data and markers -->
   <node name="rviz" pkg="rviz" type="rviz" output="screen" args="-d $(find uuv_gazebo)/rviz/controller_demo.rviz"/>
</launch>
```

Listing 2: Example robot setup describing a model with a centered IMU and a mounted camera and image sonar.

```xml
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- IMU  -->
  <xacro:default_imu_macro
    namespace="${namespace}"
    parent_link="${namespace}/base_link"
    inertial_reference_frame="${inertial_reference_frame}">
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </xacro:default_imu_macro>

  <!-- Mount a camera -->
  <xacro:default_camera namespace="${namespace}" parent_link="${
      namespace}/base_link" suffix="">
    <origin xyz="1.15 0 0.4" rpy="0 0.6 0"/>
  </xacro:default_camera>

  <!-- Mount an image sonar -->
  <xacro:forward_looking_sonar
  namespace="${namespace}"
  suffix="/image_sonar"
  parent_link="${namespace}/base_link"
  topic="image_sonar"
  mass="0.1"
  update_rate="30"
  samples="256"
  fov="1.527"
  width="768"
  height="492">
  <inertia ixx="0.1" ixy="0.0" iyy="0.1" iyz="0.0" izz="0.1" ixz=
      "0.0"/>
  <origin xyz="1.15 0.0 0.5" rpy="0 0.6 0"/>
  </xacro:forward_looking_sonar>
</robot>
```

B Hardware Electronic schematics

This section includes the hardware rig wiring and breadboard schematics, as well as the wiring diagram for the STIM 300 and a table covering the voltage levels of the power supply used.

Table 1: Voltage levels for the power supply used.

Component	Voltage [V]
Pressure sensor	5
STIM300	5
Teensy 3.2	5
Camera	5-16
S1-locator	10-18
Lights	10-48
DC-DC Buck converter in	4-38
DC-DC Buck converter out	1.5-36

Max power supply: 18 V

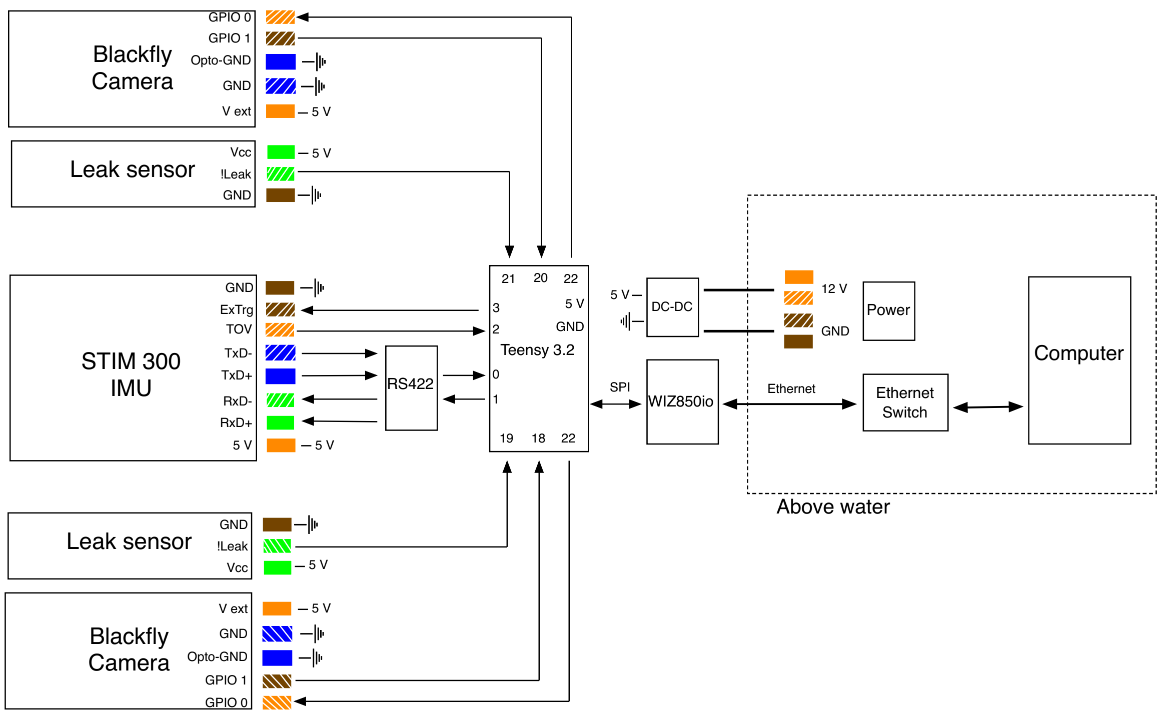


Figure B.1: Wiring

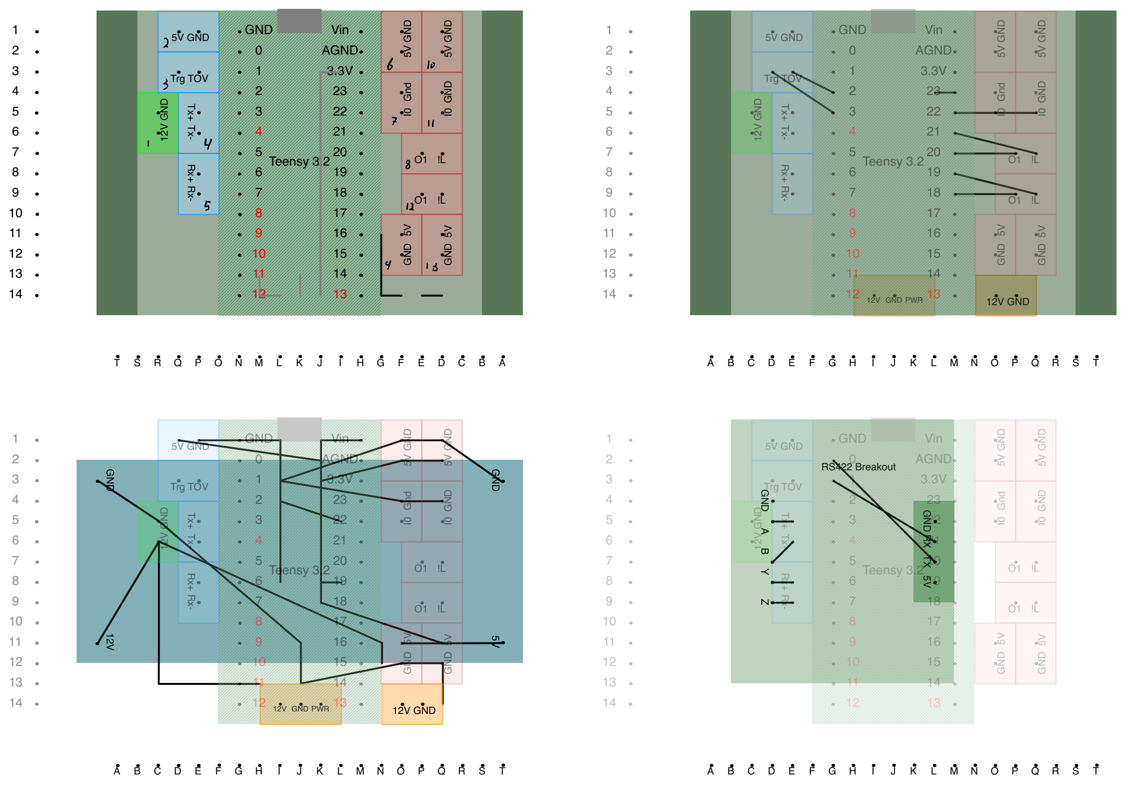


Figure B.2: Breadboard schematics

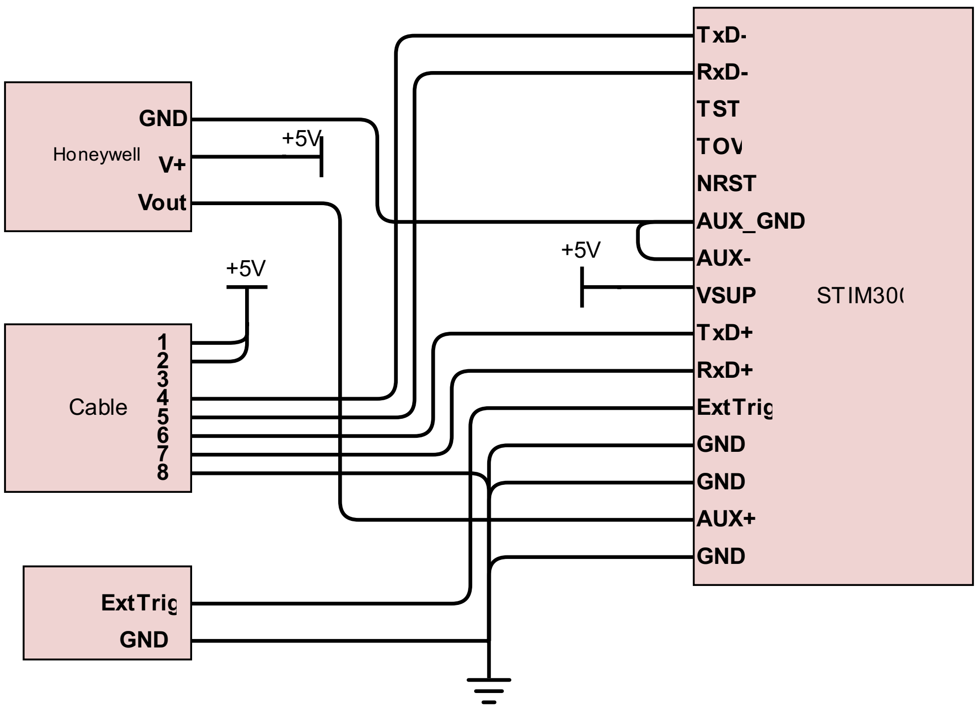


Figure B.3: STIM 300 and pressure sensor wiring.

C Proof: Path Curvature Criteria (for USM)

For any USM consisting of j rigid links attached by joints with a maximum joint angle $q_{i,\max}$, the path curvature κ_π must satisfy the constraints put forth in eq. (12.3). If any of these constraints are not upheld, it is kinematically impossible for the USM to follow the circular path exactly.

Proof. Given a USM with j joints q_i with intermediate links of length l_i following a circular arc and maximum joint angles of $q_{i_{\max}}$. The curvatures of a specific segment is then given by $\kappa_i = \frac{2\sin(q_i)}{l_i}$. For the first joint to cut with the circle segment $\frac{\theta_1}{2} \leq q_{1,\max}$ must be satisfied. This gives that $q_f + \frac{\theta_1}{2} = q1$. Due to the nature of the links, the subsequent joint angle has an offset equal to the joint angle of the previous joint (see fig. 12.14). This means that the next joint angle has to satisfy $\frac{\theta_2}{2} \leq q_{2,\max} - \frac{\theta_1}{2}$.
Continuing this argument for each subsequent joint completes the proof of the curvature constraints in eq. (12.3). □

Remark C.1. The argument put forth in eq. (12.3) and appendix C extends to non-circular paths as well, assuming that the maximum curvature of the path enclosed by the link is available (see fig. C.1). The argument from eq. (12.3) can then be approximated using $\kappa = \kappa_{i,\max}$ determined by over-estimating the joint angle.

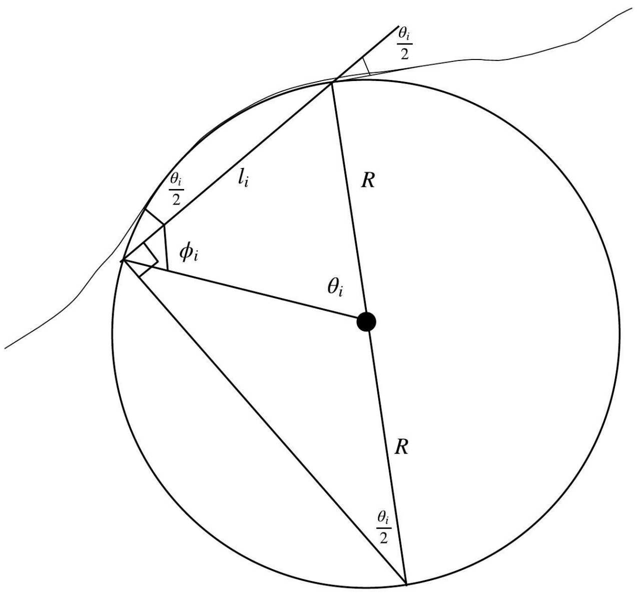


Figure C.1: The curved path with a circle drawn at the point of maximum curvature. The angles $\frac{\theta_i}{2}$ shows that the link cutting the circle over-estimates the necessary joint angle. The extended line across the circle provides the geometric relations necessary to show how the angles propagate.

D 3D Point Cloud Classification

The segmentation performed in this thesis is performed on images fused with depth data from the SLAM algorithm. The concept of performing this directly on the point cloud was briefly explored, however. Some of the potential basis literature will, therefore, be left here as a starting point for potential future improvements.

Three-dimensional point cloud sensors have in recent years become one of the de-facto standards due to their performance and availability. These types of 3D sensors include stereo cameras, LiDAR, structured light, and time-of-flight, among others. Seeing as point cloud data is a very general, and canonical, way of representing sensor information, it is of interest to be able to directly utilise this data in autonomous tasks. Due to this, point cloud techniques have become increasingly important for autonomous robotics, as this data structure is very close to raw sensor data. Contextual information regarding the robot's surrounding have most often been extracted from visual sensors. This is the case in the main body of this thesis. Other examples include semantic segmentation of camera images. This can be achieved in many different ways, but the most prominent one is through *Convolutional Neural Networks* (CNNs). One much used network architecture to perform semantic segmentation is showcased in the U-Net [218]. This network is essentially divided into two parts: *i) an encoder* and *ii) a decoder*. The encoder part acts like a normal CNN, learning feature representations at increasing levels of abstraction . The decoder part, on the other hand, performs successive up-sampling combining the output from previous convolutions in the encoder. These kind of networks, and their successors, can give very good results on image segmentation problems. However, problems arise when these concepts are to be applied on three-dimensional point cloud data.

Point cloud data, in this context, is usually represented as a multitude of points $\mathbf{p} = (x, y, z)$, alternatively $\mathbf{p} = (x, y, z, r, g, b)$[5]. These sets of points are usually unordered and irregular, which results in problems when it comes to feeding the data to a neural network. These problems arise due to the geometric properties convolutional networks require to, for example, perform proper weight sharing. Workarounds do exist, such as the conversion from point clouds to voxel grids [219]–[221], although this tend to increase the data volume. Recent methods have, despite the inherent structural difficulties, managed to train end-to-end networks solely on point cloud data, however. PointNet [222] is one such method.

The main contribution from PointNet, was the fact that it solved the problem of rotational and permutational invariance. As mentioned, point cloud data is unordered, which means that there is no implicit permutation of the points that characterise the detected object.

[5]The exact data fields differ depending on means of acquisition, but can, for example, also include intensity values or normal vector data.

Permutational invariance is achieved by relying on symmetric functions[6]. PointNet realises this by combining a single variable and max pooling to extract a set of *critical points* to describe the input point cloud.

The other problem PointNet solved was how to obtain invariance with regard to transformations. I.e. the classification result remains unaltered if the same object is fed to the network with different spatial orientations. To overcome this problem, PointNet utilises a small transformation network to estimate a transformation directly on the input data which allows for automatic data alignment.

At the end of the segmentation pipeline, the PointNet outputs a *per-point* class label, which is calculated based on point-by-point local and global features. As pointed out in [107], the point aggregation performed by the PointNet results in information only being passed between points contained in the same block. The way PointNet limits its information utilisation, by restricting the network to work with singular blocks, was tried improved upon in [107]. The presented network takes the PointNet one step forward, by evaluating points using *multi-scale blocks*. These blocks are all sampled from the same position, but taken at different scales, broadening the network's *field of view*.

Point cloud segmentation can, of course, be based solely on clustering principles. This can be done by utilizing the intrinsic properties of different objects. I.e. their shape, color, intensity, etc. By using these *simpler* segmentation techniques, the runtimes are expectedly faster compared to neural network approaches. Although, in general, AI-based methods tend to outperform the purer geometric clustering techniques, due to them being able to make use of contextual information. Therefore, the perhaps most promising method for segmentation of point clouds, especially in larger environments, is segmentation using a graph convolutional network based on superpoint graphs [223]. A superpoint graph (SPG) is an attributed directed graph where the nodes represent geometric shapes and the edges represent the nodes' adjacency based on rich edge features. This way of representing the data avoids classifying individual points of voxels and does instead consider the object as a whole. An advantage of this method, is that the size of the graph is not dependant on the total number of points, but rather the number of structures in the scene. All in all, this method of point cloud segmentation obtains a higher accuracy compared to other state-of-the-art methods. Thus, applying this to underwater sensor data could be of great interest, as it could allow for safer navigation by using the semantic information to characterize safe and unsafe areas. This would, however, require the labelling of many large-scale point clouds, as no such thing is publicly available at time of writing.

[6]A function $f(\cdot)$ is symmetric if $f(x_1, x_2, ..., x_\mathrm{n}) = ... = f(x_\mathrm{n}, ..., x_2, x_1$ for any permutation of $\mathbf{x}$.

E Path Projection

Definition E.1. A parametric curve in $\mathbb{R}^n$ is a smooth function $\pi : \Theta \longrightarrow \mathbb{R}^n$, where $\Theta \subset \mathbb{R}$ is an interval.

Remark E.1. In 3D definition E.1 implies, provided sufficient smoothness, that the position, velocity and acceleration vectors of the curve can be expressed by

$$\begin{aligned} \pi(\varpi) &= (x(\varpi), y(\varpi), z(\varpi)) \\ v(\varpi) &= (x'(\varpi), y'(\varpi), z'(\varpi)) \\ a(\varpi) &= (x''(\varpi), y''(\varpi), z''(\varpi)) \end{aligned} \tag{1}$$

where $\varpi \in \Theta$ is the is the curvilinear path progress variable.

The closest point between a curve $\pi(\varpi)$ and the position of the USM, $\eta_{Ib,1}$, is required. In the general case, this is equivalent to solving the problem

$$\varpi^* = \text{argmin}_{\varpi} \left(\eta_{Ib,1} - \pi(\varpi)\right)^2 \tag{2}$$

which again is equivalent to finding the orthogonal projection of the current position onto the curve, meaning $\varpi^* = \varpi_\perp$. This can be computed using Newton's method which constitutes of computing the *Newton step*, which is summarized in the following lemma.

Lemma E.1. *Let $\pi(\varpi) : I \longrightarrow$, $\varpi \in I$ be a regular curve — at least twice contiuously differentiable — and $\eta_{Ib,1}$ be a point in 3D. Then the* ***Newton step*** *can be computing as*

$$\Delta\varpi = \frac{\langle v(\varpi), \eta_{Ib,1} - \pi(\varpi)\rangle}{\langle a(\varpi), \pi(\varpi)\rangle + \langle v(\varpi), v(\varpi)\rangle - \langle \eta_{Ib,1}, a(\varpi)\rangle} \tag{3}$$

Proof. Consider the minimization problem (2) with $f(\varpi_k) = (\eta_{Ib,1} - \pi(\varpi))^2$. Then by the chain rule

$$\frac{d}{d\varpi_k} f(\varpi_k) = -2v(\varpi_k)(\eta_{Ib,1} - \pi(\varpi_k)) \tag{4}$$

Applying the chain rule once more gives

$$\begin{aligned} \frac{d^2}{d\varpi_k^2} f(\varpi_k) &= -2a(\varpi_k)(\eta_{Ib,1} - \pi(\varpi_k)) + 2v(\varpi_k)^2 \\ &= 2v(\varpi_k)^2 - 2a(\varpi_k)\eta_{Ib,1} + 2a(\varpi_k)\pi(\varpi_k) \end{aligned} \tag{5}$$

Computation of the newton step gives the desired result

$$
\begin{aligned}
\Delta\varpi_k &= \frac{f'(\varpi_k)}{f''(\varpi_k)} \\
&= \frac{v(\varpi_k)(\eta_{Ib,1} - \pi(\varpi_k))}{v(\varpi_k)^2 - a(\varpi_k)\eta_{Ib,1} + a(\varpi_k)\pi(\varpi_k)} \\
&= \frac{\langle v(\varpi), \eta_{0b,1} - \pi(\varpi)\rangle}{\langle a(\varpi), \pi(\varpi)\rangle + \langle v(\varpi), v(\varpi)\rangle - \langle \eta_{Ib,1}, a(\varpi)\rangle}
\end{aligned}
\tag{6}
$$

□

A second order optimization problem can be formulated based on lemma E.1 and a line search algorithm.

Theorem E.1. *Let $\pi(\varpi) : I \longrightarrow,\ \varpi \in I$ be a regular curve and $\eta_{Ib,1}$ be a point in 3D. Furthermore, define α as a line search scaling parameter. Scaling the current Newton step by α^n where n is determined by the squared distance*

$$\langle \pi(\varpi_k + \alpha^i \Delta\varpi_k) - \eta_{Ib,1}, \pi(\varpi_k + \alpha^i \Delta\varpi_k) - \eta_{Ib,1}\rangle < d \tag{7}$$

where

$$d = \langle \pi(\varpi_k + \Delta\varpi_k) - \eta_{Ib,1}, \pi(\varpi_k + \Delta\varpi_k) - \eta_{Ib,1}\rangle \tag{8}$$

is the squared distance of the between the previous iterate $\delta_{k-1} + \Delta\varpi_{k-1}$ and the current position $\eta_{Ib,1}$. Furthermore, convergence can be guaranteed provided that the existence of a strict minimizer ϖ^*, see theorem 2.4, and that $f''(\varpi_k)$ is Lipschitz continuous, see theorem 3.5. Lastly, provided that the α^i fulfills the role of theorem 3.6, superlinear convergence can be guaranteed after a certain number of iterations.

A proof of theorem E.1 follows directly from the theorems mentioned above with;

- Quadratic rate of convergence
- Quadratic convergence of the sequence of gradient norms $||f'(\varpi_k)||$ to zero.

Remark E.2. The conditions of theorem 2.4 in Theorem E.1 is not always satisfied due to the general nature of a path. Think for example of the current position, $\eta_{Ib,1}$, located at the center of a circle. Then, there exists a continuum of optimum points.

To avoid situations in remark E.2 occurring to often, the initial condition from the current rate of progress will usually be used as an initial condition for the next projection step. By assuming that the desired path is reasonably close to the robot, the above will not cause any problems in practice.

Remark E.3. With a closed spline, the Newton step have to account for the transition between the end of the curve and the start of the curve. This is handled with bisection to cover both cases.

Remark E.4. When projection reaching the end of a spline, the algorithm have to be stopped and the spline ending is the desired point of reference. Alternatively, the curve can be extrapolated.