Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Shuyuan Shen

Computer Vision Based Motion Estimation for ROVs

Master's thesis in Marine Technology

Supervisor: Martin Ludvigsen

June 2020

NTNU
Norwegian University of
Science and Technology

Shuyuan Shen

Computer Vision Based Motion Estimation for ROVs

Master's thesis in Marine Technology
Supervisor: Martin Ludvigsen
June 2020

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Norwegian University of
Science and Technology

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

MASTER THESIS IN MARINE CYBERNETICS

SPRING 2020

FOR

STUD. TECHN. SHUYUAN SHEN

Computer Vision Based Motion Estimation for ROVs

Work description

The remotely operated vehicle (ROV) has been widely used in underwater operations for many years. Compared with surface vessels, absolute global navigation, like GNSS, is unreliable for ROVs in deep water. To work on man-made seabed installations, ROVs need local navigation. To find desired features in an autonomous system, spatial references on centimetre accuracy and online data processing are required. The goal is use stereo camera to derive motion online. Use images from stereo camera as input, visual simultaneous localization and mapping (SLAM) is the main method to estimate ROV motion. A map of surroundings is constructed and updated while simultaneously the location of ROV is tracked within the map. The motion from visual SLAM is used in ROV control system for local navigation.

Scope of work

1. Review relevant literature within the field of computer vision and SLAM for positioning and navigation.
2. Detect, describe and match features from underwater stereo camera images.
3. Derive ROV motion online using visual SLAM.
4. Verify performance by simulations and discuss the results.

The report shall be written in English and edited as a research report including literature survey, description of mathematical models, description of algorithms, simulation results, discussion and a conclusion including a proposal for further work. Source code should be provided. It is supposed that Department of Marine Technology, NTNU, can use the results freely in its research work, unless otherwise agreed upon, by referring to the student's work.

The thesis should be submitted within June 2020.

Professor Martin Ludvigsen
Supervisor

Abstract

This project thesis presents a real-time visual simultaneous localization and mapping (SLAM) based motion estimation system of remotely operated vehicle (ROV) using a stereo camera. The main purpose is to increase local navigation and situation awareness of the ROV, and increase the ROV autonomy level.

A stereo camera that records sea floor images is equipped on the ROV. The geometric models used in this thesis are pin-hole camera model. The image pairs from the stereo camera are input data to estimate ROV motion based on computer vision techniques. The images are enhanced by contrast limited adaptive histogram equalization (CLAHE) algorithm to improve image contrast.

Map is the core of the SLAM problem. Map points and camera positions, which are called poses in computer vision, are stored in the map. Map points are points that their positions in world coordinate are known. The measurable pieces of data in the image is detected as feature. Oriented Features from Accelerated Segment Test and Rotated Binary Robust Independent Elementary Features (ORB) method is used for feature detector and descriptor. The features from left camera image in current processing frame are matched with map points. Distance filter and Random Sample Consensus (RANSAC) are two steps to remove mismatches.

Camera poses are estimated by Efficient Perspective-n-Point (EPnP) method. The results are optimized using graph based bundle adjustment. The current frame is set as a keyframe if the number of matched features is not large. Features in left and right camera images are matched to calculate new map points. The new keyframe and map points are added in the map for further use. A large bundle adjustment including several nearest keyframes camera poses and new map points is implemented to generate optimized trajectory and map.

A TCP client is included in the system. It establishes connection with TCP server in the ROV control system and receives initial ROV position. The client also reads estimated position results, converts local camera poses to local ROV position and transmits global ROV position to the control system.

The SLAM system is tested in three scenarios based on two different image sets. The results shows that the system is able to estimate ROV motion. But large errors occurs in orientation. The calculation time of one frame is 0.2 seconds on average, meaning the system can process frames at the frequency of 5 Hz.

In order to improve the performance of the system. Other better image enhancement methods should be studied and tested. Besides, loop closure should be considered to add in the system to reduce accumulated drift.

Preface

This master thesis is a part of the study program Marine Technology at Department of Marine Technology, Norwegian University of Science and Technology. The work was carried out in the spring semester of 2020, with Professor Martin Ludvigsen as supervisor. The object of this thesis is to use computer vision techniques and visual input from a stereo camera to estimate motion of a remotely operated vehicle (ROV).

The real-time visual simultaneous localization and mapping (SLAM) system is programmed with C++. The author of this thesis did not have any experience with computer vision and C++ language before the project was embarked. The learning progress has been steep in this semester. The system are not perfect, but works for the designed application.

Trondheim, June 11, 2020

Shuyuan Shen

Acknowledgements

I would like to thank my supervisor Professor Martin Ludvigsen for helpful guidance and suggestions. He provided me with feedback and discussion during meetings.

I would also like to thank to master students Fan Gao, Erlend Røilid Vollan and Signe Brirch Moltu. We worked together for the ROV autonomy framework, and they gave me useful suggestions for my system.

In addition, I would like to thank my friends and family for support and help during my master study.

Shuyuan Shen

Table of Contents

List of Tables

List of Figures

Abbreviations

ROV	=	Remotely Operated Vehicle
EKF	=	Extended Kalman Filter
SLAM	=	Simultaneous Localization and Mapping
OpenCV	=	Open Source Computer Vision Library
GNSS	=	Global Navigation Satellite System
IMU	=	Inertial Measurement Unit
AHE	=	Adaptive Histogram Equalization
CLAHE	=	Contrast Limited Adaptive Histogram Equalization
ORB	=	Oriented FAST Rotated BRIEF
FAST	=	Features from Accelerated Segment Test
BRIEF	=	Binary Robust Independent Elementary Features
FLANN	=	Fast Library for Approximate Nearest Neighbors
PnP	=	Perspective-n-Point
EPnP	=	Efficient Perspective-n-Point
BA	=	Bundle Adjustment
RANSAC	=	Random Sample Consensus
TCP	=	Transmission Control Protocol
BoG	=	Bag-of-Words

Chapter 1

Introduction

1.1 Background

1.1.1 ROV

The ocean is the lifeblood of Earth, covering approximately 71% of Earth's surface and 90% of the Earth's biosphere [1]. Human has been exploring the ocean for a long time in search of biological resources, mineral resources and oil resources. To explore the deeper ocean, many equipment and underwater vehicles including ROV are invented.

ROV stands for Remotely Operated Vehicle. Many deep-water environments are too harsh for humans to directly work in. ROVs are usually used in deep-water scientific research and operation where divers cannot reach. ROVs are commonly operated by a crew aboard a vessel via a reinforced umbilical cable. The umbilical provides both the electrical power and data transmission between the vessel and ROV. Motion of the ROV is controlled by several thrusters that allows translation and rotation in all directions. Most ROVs are equipped with a video camera and lights. Additional equipment is commonly added to expand the vehicle's capabilities of data acquisition and operation. These may include manipulators, sonars, magnetometers and instruments that measure water clarity, water temperature, water density.

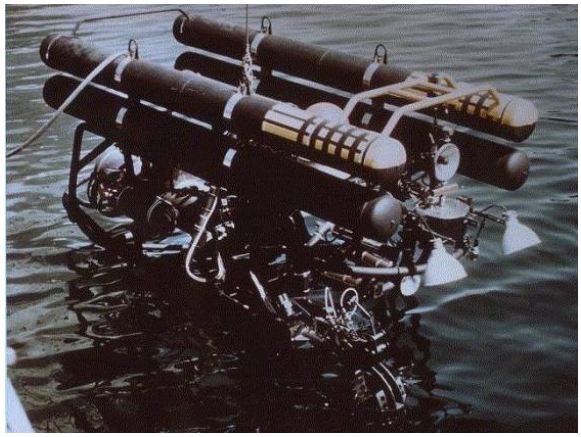


Figure 1.1: Cable-Controlled Underwater Research Vehicle in 1961

The first fully developed ROV, POODLE, was created by Dimitri Rebikoff in 1953 [2]. However, it was not until the United States Navy took an interest in ROVs that the technology really started to studied and developed. In 1961 the Cable-Controlled Underwater Research Vehicle (CURV) was created by the former Pasadena Annex of the Naval Ordnance Test Station [3]. CURV was used to find and retrieve lost sunken torpedoes, and it paved a brand new way in deep sea exploration. Since then, technological development in the ROV industry has accelerated. Today numerous tasks are perform by ROVs in many fields. ROVs are used for observation and operation assistance. Their tasks range from ocean scientific research, inspection of subsea structures and pipelines, to placing underwater platforms and connecting pipelines.

1.1.2 Visual SLAM

Simultaneous localization and mapping (SLAM) is the problem that builds or updates a map of an unknown environment and simultaneously track the motion of a vehicle within the map. Mapping and localization seems like a chicken-and-egg problem when a vehicle starts moving in a unknown place. Sensors equipped on the vehicle detect information from surrounding environment. Some methods are used to extract information that can uniquely describe environment from this sensor data. The information is generally described as the form of point, line, light change, etc. The map is constructed using this information. Meanwhile the infor-

mation is also used to localize the vehicle. This information is called landmark, and point type information is also called map point. The position and orientation of the sensor are called pose in computer vision. Several sensors can be used for SLAM, such as camera, lidar and sonar. Camera based SLAM is a extension of computer vision techniques, and is called visual SLAM.

A basic mathematical formulation of SLAM problem is:

$$\mathbf{x}_i = f(\mathbf{x}_{i-1}, \mathbf{u}_i, \mathbf{w}_i), \quad i = 1, ..., k \tag{1.1}$$

$$\mathbf{z}_{ij} = h(\mathbf{y}_j, \mathbf{x}_i, \mathbf{v}_{ij}), \quad (i, j) \in \mathcal{O} \tag{1.2}$$

The position of the vehicle $\mathbf{x}_i$ in step i is determined by the position in last step $\mathbf{x}_{i-1}$ and control input $\mathbf{u}_i$, which is influenced by noise $\mathbf{w}_i$. A general function $f(\cdot)$ describes the process. The sensor equipped on the vehicle sees a landmark $\mathbf{y}_j$ at $\mathbf{x}_i$ and generate an observation data $\mathbf{z}_{ij}$. The noise in this observation is $\mathbf{v}_{ij}$. The observation process is described with an abstract function $h(\cdot)$. $\mathcal{O}$ is a set that contains the information at which pose the landmark was observed.

The SLAM problem can be described as how to slove the estimate $\mathbf{x}$ and $\mathbf{y}$ problem with the noisy control input $\mathbf{u}$ and the sensor observation data $\mathbf{z}$. There are different approaches to solve the problem. The earliest SLAM system was developed based on Extended Kalman Filter (EKF) [4]. In order to overcome the shortcomings of EKF, like Gaussian noise assumptions and linearization error, other filter based methods are developed such as particle filters [5] and information filters [6].

Today, the mainstream approaches solving visual SLAM problem are state-of-the-art optimization techniques. The problem is represented by graph optimization. A graph is established whose vertices represent camera poses or landmarks, and the edge between two vertices represents a camera observation that constrains the vertices. The technique has been proposed by Lu and Milios [7]. More details are introduced in Section 2.7.4.

A typical visual SLAM framework includes the following parts: sensor data acquisition, front end, back end, loop closing and reconstruction. Sensor data acquisition mainly refers to camera images acquisition and pre-processing. The task of the front end is to estimate the camera motion between adjacent frames and generate a rough local map. The back end receives camera poses from front end, as well

as the results from loop closing, and implements optimization to generate a full optimized trajectory and map. Loop closing determines if the vehicle has returned to its previous position in order to reduce accumulated drift. Reconstruction constructs a task specific map based on the optimized trajectory and map in back end. In this thesis, loop closing is not included, and is set as part of future work due to limited time. No specific map is needed for motion estimation. Therefore, these two parts are not discussed and implemented in the project.

1.1.3 Software

In this section the software used to develop the SLAM system is presented.

OpenCV

OpenCV is an open source computer vision and machine learning software library. It is written natively in C++, and has C++, Python, Java and MATLAB interface, and supports Windows, Linux, Android and Mac OS. Wrappers in other languages such as C# and Ruby are also available.

OpenCV is originally developed by Intel, then support for OpenCV was taken over by a non-profit foundation OpenCV.org. The library is used by well-established companies like Google, Microsoft, Intel and IBM. The deployed uses of OpenCV spans the whole world. Some examples are intrusions detection in surveillance video in Israel, mine equipment monitoring in China and detection of swimming pool drowning accidents in Europe [8].

g2o

General Graph Optimization (g2o) is an open-source C++ framework for optimizing graph-based nonlinear least squares problems. g2o is developed by Rainer Kuemmerle et al. in 2011 [9]. The SLAM problems involve the minimization of a nonlinear error function that can be presented as a graph. The overall goal in these problems is to find the state variables, such as camera poses and map points position, that maximally explain a set of observations affected by Gaussian noise. g2o is designed to be easily extensible to a wide range of SLAM problems and a new problem typically can be specified in a few lines of code.

Sophus

Sophus is a C++ implementation of Lie groups that are commonly used for 2D and 3D geometric problems in computer vision or robotics applications. Sophus is developed and maintained by Hauke Strasdat. A basic introduction of Lie group is presented in Section 2.7.1.

Pangolin

Pangolin is a lightweight development library for managing OpenGL display and interaction, and is developed by Steven Lovegrove. Pangolin provides modularized 3D visualization based on OpenGL viewport manager. It also provides a mechanism for manipulating program variables and a flexible real-time plotter for visualizing graphical data.

1.2 Challenges

Several challenges of applying visual SLAM into underwater environment are introduced. SLAM can only be used when the ROV moves near sea floor due to low visibility of sea water. High level of turbidity can significantly reduce image clarity. Besides, the light conditions on seabed are poor, and the range of artificial light is not uniform, resulting in different contrast in different regions of one image.

Sea floor landforms are varying in different place. The sea floor covered with

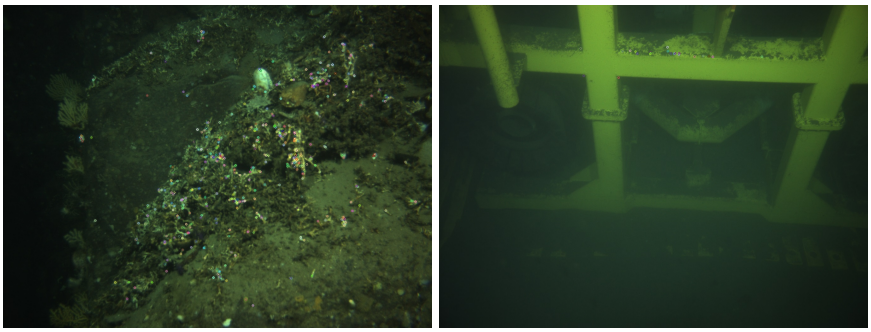


Figure 1.2: ORB keypoints detection in two different underwater images

rocks or coral is easy to detect features while the sea floor covered with sand is hard to find features. A test of detecting features is implemented using two different underwater images. The results are presented in Figure 1.2. Detected feature keypoints represented by small circles. A large number of features are detected from the left image. But few features are detected in the right image because the sea water is turbid the right image and sea floor is hard to be observed.

1.3 Objectives

The remotely operated vehicle (ROV) has been widely used in underwater operations for many years. To increase ROV autonomy, situation awareness and high accuracy motion estimation of the ROV is required. Compared with surface vessels, absolute global navigation, like global navigation satellite system (GNSS), is unavailable for ROV in deep water. To work on man-made seabed installations, ROV need local navigation. To find desired features in an autonomous system, spatial references on high accuracy and online data processing are required.

The challenges of ROV navigation is connected to the limitations of acoustic positioning from sonar and dead-reckoning navigation from inertial measurement unit (IMU). Computer vision techniques have been researched and developed in the last three decades. The techniques can offer new solution to the challenge, and help to increase autonomy level of the ROV.

The goal of this thesis is use computer vision techniques with stereo camera image input to derive ROV motion online. The estimated position from visual SLAM is used in ROV control system for local navigation.

1.4 Scope and Limitations

The scope of this thesis will be the development and implementation of a real-time visual SLAM motion estimation system for the ROV. Images from stereo camera equipped on the ROV are used as input data, and the output from the system is transmitted to ROV control system for local navigation. The system is programmed with C++ using several open source libraries and frameworks.

The SLAM system will be tested on two different image sets from previous mis-

sions. The results will be compared with navigation data from the ROV control system using other sensors measurements.

The SLAM system was meant to be included in the ROV autonomy framework developed by four master students Fan Gao, Signe Brirch Moltu, Erlend Røilid Vollan and the author of this thesis. However, due to the special situation of coronavirus in this semester, the framework was not accomplished, and sea test of the SLAM system was canceled.

1.5 Structure of Thesis

The structure of this thesis is presented in a classical manner. The structure is defined as:

- **Chapter 1: Introduction.** This chapter presents the introduction of this thesis.
- **Chapter 2: Theory.** The chapter introduces the theory used for system development. Camera models, feature descriptor based motion estimation methods and graph optimization are presented.
- **Chapter 3: Method.** This chapter presents the methods used for the SLAM system. System implementation are also described.
- **Chapter 4: Simulation.** This chapter presents simulation set-up, results and comments from three scenarios and TCP data transmission.
- **Chapter 5: Discussion.** This chapter includes discussion of the simulation results.
- **Chapter 6: Conclusion.** This chapter concludes the thesis and suggest further work.

1.6 Thesis Contribution

The main topic of this thesis is the application of visual SLAM techniques to ROV motion estimation, and the estimation should be used in the ROV control system.

The objective for this thesis is to increase ROV local navigation accuracy as well as the autonomy level.

Visual SLAM techniques have been extensively researched and widely used by ground vehicles. In this thesis the techniques have been taken into underwater environment, using stereo camera equipped on a ROV. The research presented investigates weather feature based visual SLAM can be applied in underwater to estimate ROV motion.

A real-time visual SLAM system is developed and implemented. This thesis gives valuable information in the development of visual SLAM application in underwater environment to improve ROV situation awareness and local navigation.

Chapter 2

Theory

2.1 Camera model

2.1.1 Pin-hole camera model

The processing of projecting a 3D point to a 2D image plane by a camera can be described by a geometric model. In this thesis, a basic model called pin-hole model is used. Recall the candle projection experiment, a lit candle is placed in front of a dark box, and the light of the candle is projected through a small hole in the dark box on the rear plane of the black box. Then an inverted candle image is formed on this plane. The small hole is able to project a candle in a three dimensional world onto a two dimensional imaging plane in this process. For the same reason, the simple pin-hole model can be used to describe imaging process of the camera.

From Figure 2.1 , the camera coordinate system is $O - x - y - z$. Put x axis to the right of the camera, y axis to the down and z axis to the front. O is the optical center of the camera, which is also the hole in the pin-hole model. The 3D point $\mathbf{P} = [X, Y, Z]^{\mathrm{T}}$ in camera coordinate, after being projected through the optical center, falls on the physical imaging plane $O' - x' - y' - z'$, and produces the image point $\mathbf{P}' = [X', Y', Z']^{\mathrm{T}}$ in image plane coordinate. The distance from camera plane to image plane is called focal length f. According to the similarity of the triangles, there are

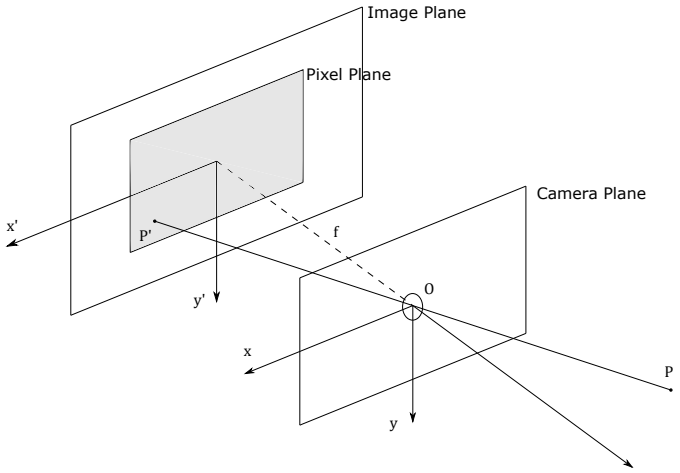


Figure 2.1: Pin-hole camera model

$$\frac{Z}{f} = -\frac{X}{X'} = -\frac{Y}{Y'} \tag{2.1}$$

The negative sign indicates that the image is inverted. But for convenience, the image obtained by a actual camera is not inverted. The model can be changed by placing the image plane symmetrically in front of the camera, as shown in Figure 1. Then Equation (2.1) becomes:

$$\begin{aligned} X' &= f\frac{X}{Z} \\ Y' &= f\frac{Y}{Z} \end{aligned} \tag{2.2}$$

The Equation (2.2) describes the spacial relationship between the world point P and its image. But in computer vision pixel coordinates $\mathbf{u} = [u, v]^{\mathrm{T}}$ of point P' is commonly used. A pixel plane $o-u-v$ is fixed on the physical imaging plane. o is set in the upper left corner of the image, u and v are parallel to x and y respectively. The relationship between the camera coordinate and the pixel coordinate is:

$$\begin{aligned} u &= f_x\frac{X}{Z} + c_x \\ v &= f_y\frac{Y}{Z} + c_y \end{aligned} \tag{2.3}$$

where $[c_x, c_y]^{\mathrm{T}}$ is optical center translation vector between image plane and pixel coordinate in pixels, f_x and f_y are focal length in pixels. In matrix form:

$$Z \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \mathbf{KP} \tag{2.4}$$

The matrix $\mathbf{K}$ is called camera intrinsic. It is generally believed that the intrinsic parameters are fixed after camera manufacturing.

Assume the world coordinates of $\mathbf{P}$ is $\mathbf{P}_w$. The conversion between $\mathbf{P}_w$ to $\mathbf{u}$ is based on the pose of the camera. The pose of the camera is described by a rotation matrix $\mathbf{R} \in SO(3)$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$, which are also called camera extrinsic. Then there are:

$$\begin{aligned} Z\mathbf{P}_{uv} &= \mathbf{K}(\mathbf{R}\mathbf{P}_w + \mathbf{t}) \\ Z\mathbf{P}_{uv} &= \mathbf{K}\mathbf{T}\mathbf{P}_w \end{aligned} \tag{2.5}$$

where the transform matrix $\mathbf{T} \in SE(3)$ is defined as:

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \tag{2.6}$$

When using $\mathbf{R}$ and $\mathbf{t}$ to describe camera pose, point is in non-homogeneous coordinates. When using $\mathbf{T}$, point is in homogeneous coordinates, in other words, point is described by a four-dimensional vector that add 1 at the end of its three-dimensional vector.

There are many ways to describe camera orientation. Each of these has advantages and disadvantages. Euler angle is the most straightforward way, but the main drawback is that it encounters gimbal lock. Quaternion can avoid the problem. The shortcoming is not intuitive and the operation is complicated. Rotation matrix is widely used in SLAM problems, but the matrix expression is redundant and the matrix itself has constrains.

The motivation that use $\mathbf{T}$ to describe camera pose is good transformation form and simple programming. Continuous transformation can be written as the multiply of transform matrices:

$$\mathbf{b} = \mathbf{T}_1\mathbf{a},\ \mathbf{c} = \mathbf{T}_2\mathbf{b} \quad \Rightarrow \mathbf{c} = \mathbf{T}_2\mathbf{T}_1\mathbf{a} \tag{2.7}$$

Compared with the intrinsic, the extrinsic changes with camera movement. It represents the trajectory of a vehicle, and is also the target to be estimated in SLAM.

2.1.2 Stereo camera model

The pin-hole camera model describes the imaging model of a single camera. But it is impossible to determine the specific location of a spacial point by a single pixel. This is because all points on the line from the optical center to the normalized plane can be projected onto that pixel. But the location can be computed by the pixels difference from stereo camera.

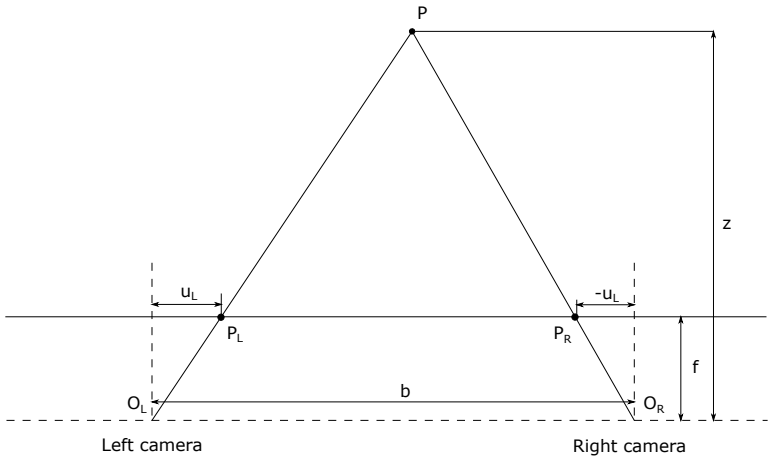


Figure 2.2: Stereo camera model

From Figure 2.2, consider a 3D point $\mathbf{P}$ that is projected into the left camera and right camera. The points are $\mathbf{P}_L$ and $\mathbf{P}_R$ on two imaging planes. Due to the presence baseline b , which is the distance between two optical center, these two positions are different. In the case that left and right cameras are only shifted on the x axis, the difference between the horizontal coordinates of left and right images is defined as disparity:

$$d = u_L - u_R \tag{2.8}$$

where u_L and u_R are the u axis pixel coordinates of $\mathbf{P}_L$ and $\mathbf{P}_R$ respectively. Note that u_R should be a negative value in Figure 2.2, the physical distance should be $-u_R$. According to the similarity of the triangles, there are:

$$\frac{z-f}{f} = \frac{b-u_L+u_R}{b} \tag{2.9}$$

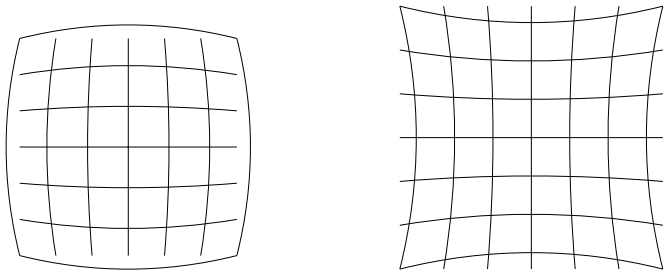
The depth of the point is:

$$z = \frac{fb}{d} \tag{2.10}$$

2.1.3 Distortion model

In order to get large field of view, a lens is normally added in front of the camera. The addition of the lens has an influence on the propagation of light during imaging. First, the shape of lens may affect the propagation way of light. Second, the lens and the imaging plane are not strictly parallel.

The distortion caused by the shape of lens is called radial distortion. In pin-hole camera model, a straight line projected onto the pixel plane is still a straight line. But in real photos, the lens of the camera turns a straight line in the real environment into a curve. Since the lenses are often center symmetrical, it makes the distortion radially symmetrical. The radially distortion can usually be classified as barrel distortion and pincushion distortion. They are shown in Figure 2.3.



(a) Barrel distortion **(b)** Pincushion distortion

Figure 2.3: Radical distortion

Tangential distortion is introduced during assembly of the camera because the lens

and the imaging plane cannot be completely parallel, as shown in Figure 2.4.

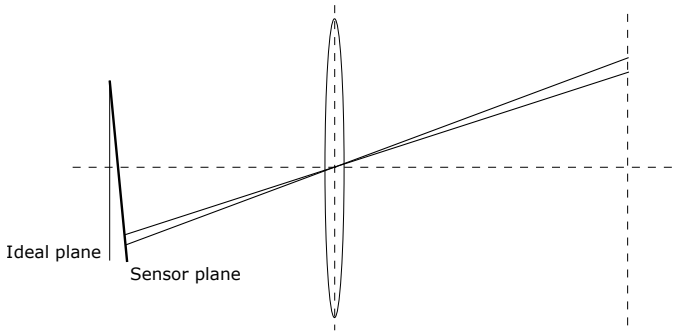


Figure 2.4: Tangential distortion

To correct distortion, Brown's distortion model [10] is used. The model corrects both radial distortion and tangential distortion. The model is shown as:

$$\begin{aligned} x' &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + 2p_1 xy + p_2(r^2 + 2x^2) \\ y' &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) + p_1(r^2 + 2y^2) + 2p_2 xy \end{aligned} \tag{2.11}$$

where $[x, y]^{\mathrm{T}}$ is the normalized coordinates of the distorted point, $[x', y']^{\mathrm{T}}$ is the normalized coordinates of the undistorted point, $r = \sqrt{x^2 + y^2}$ is the distance between the point and the origin of the coordinate system, k_1, k_2, k_3 are radial distortion coefficients and p_1, p_2 are tangential distortion coefficients.

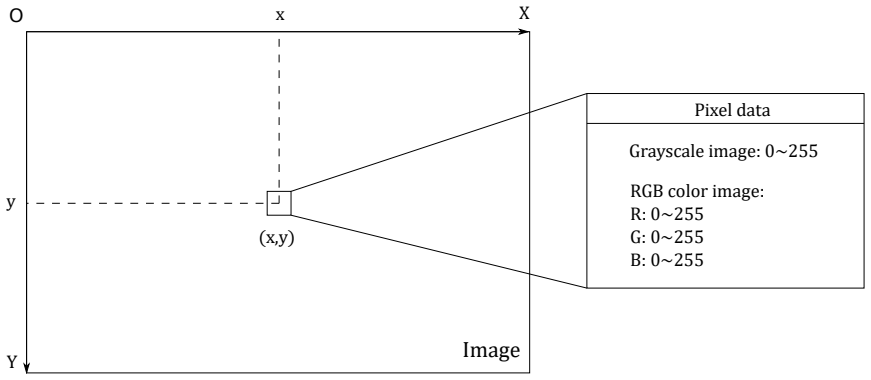


Figure 2.5: Image coordinate and pixel data

2.2 Digital image

Digital images are data sources that stored in the computer after cameras and lens converting the information in the 3D world to 2D pixels. As shown in Figure 2.5, a pixel coordinate for the image is defined as $O - X - Y$ in OpenCV, where O is the origin in the top left corner of the image, X axis is from left to right, Y axis is from top to bottom.

Two types of digital images are introduced in this part: grayscale image and color image. In a grayscale image, the gray scale of a pixel can be recorded as an 8-bit unsigned integer, which is a value of 0-255. To describe a color image, channel is required. Any color can be described as the combination of three basic colors: red, green and blue. For each pixel in computers, three values of R, G and B are recorded, and each of which is called a channel. The color image used in this thesis is the most common 24-bit color image that each channel is represented by an 8-bit unsigned integer in the range of 0 to 255. A RGB model cube is shown in Figure 2.6 [11]. The transformation from color image to grayscale image is:

$$Grayscale = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{2.12}$$

2.3 Image enhancement

The images from underwater environment often have different contrast in different image parts due to poor lighting condition. It is hard to detect features in low contrast part, so image enhancement is necessary before executing SLAM system. For

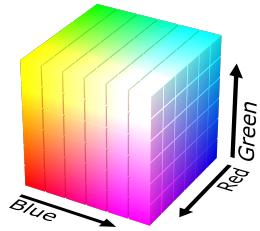


Figure 2.6: RGB model cube

grayscale images, histogram equalization is the most common method to enhance contrast. It works by mapping the histogram of the image to another histogram with a wider and more distribution of intensity values so the intensity values are spread over the whole range. The method is useful when the background and foreground of a image are both bright or both dark, in other words, the distribution of pixel values is similar throughout the image. If there are parts that are significantly lighter or darker than most of the image, the contrast in those parts will not be sufficiently enhanced [12]. makes it hard to detect features. An example is shown in Figure 2.7 [13], the contrast of the statue in initial image is high while the contrast of bookcase is low. After histogram equalization more details of the bookcase are displayed, but the statue loses details and becomes overly bright.

Figure 2.7: Image enhancement compare

Adaptive Histogram Equalization (AHE) [14] differs from the ordinary histogram equalization. AHE computes several histograms, each corresponding to a distinct section of the image, and uses them to redistribute the lightness values of the image, as shown in Figure 2.8 [12]. This method improves the local contrast, and the visual effect is better than that using the simple one.

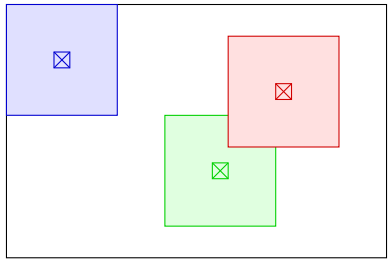


Figure 2.8: AHE neighbourhoods

The problem of AHE is that sometimes it improves the local contrast too much, which may lead to a loss of image quality and noise amplification. Contrast limited

adaptive histogram equalization (CLAHE) [15] is a variant of adaptive histogram equalization in which the contrast amplification is limited. The part of histogram that exceeds a limit value, which is called clip limit, is cut and redistributed equally among all histogram bins, which is shown in Figure 2.9 [12]. This redistribution will push some bins over the limit again. Then repeat the procedure until that part is negligible. In Figure 2.7, after enhancement, the contrast of bookcase is increased and more details are exposed both in statue and bookcase.

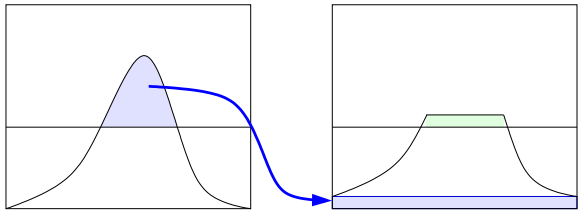


Figure 2.9: Excess redistribution in CLAHE

2.4 Feature detection and description

2.4.1 Feature descriptor method

The front end of SLAM system is concerned with the movement of a camera between adjacent image frames. Two mainstream methods are feature descriptor methods, which is also called indirect methods, and direct methods. Feature descriptor methods attempt to extract features, and then use these features to locate camera and build a map. In contrast, direct methods make use of pixel intensities directly. One important assumption of direct methods is brightness constancy [16], which is hard achieve in underwater environments. Therefore, direct methods are not discussed in this thesis. In computer vision, features are measurable pieces of data in the image that is unique to the specific objects. Good features should have the following properties:

- Repeatability: the same features can be detected and matched in different mages.
- Distinctiveness: different features should be described differently.

- Locality: the feature should be only related to a small piece of corresponding image area.

- Efficiency: the number of features should be far less than that of pixels.

Features can be detected in form of corners, blobs, edges or lines. After that, the features are described in some ways based on the information of their neighboring pixels, like pixel intensity, which makes it easy to recognition features in feature matching step. This process is called feature description. There are lots of feature detection and description algorithms. In this thesis ORB will be detailed described.

2.4.2 ORB

Oriented FAST and Rotated BRIEF (ORB) was developed by Ethan Rublee et al. in 2011 [17]. ORB combined modified FAST (Features from Accelerated Segment Test) detection and direction-normalized BRIEF (Binary Robust Independent Elementary Features) description methods. ORB algorithm has good performance under translation, rotation and scaling. Besides, keypoints detection and description are fast so the algorithm is suitable for real-time SLAM application.

FAST is a corner detection method that was developed by Edward Rosten et al. in 2006 [18]. FAST corner detector uses a circle of 16 pixels to classify whether a candidate point p is actually a corner. As shown in Figure 2.10, each pixel in this circle is labeled from integer number 1 to 16 clockwise. Assume the intensity of the pixel p is I_p, and an appropriate threshold value t is selected. If there exists a set of n contiguous pixels in the circle which are all brighter than $I_p + t$, or all darker than $I_p - t$ then point p is classified as corner. To make the detector fast, first compare the intensity of pixels 1, 5, 9 and 13. If at least 3 pixels out of 4 example pixels that are not brighter than $I_p + t$ or darker than $I_p - t$, the point p is not a corner. Otherwise check all 16 pixels and check if 12 or more pixels fall in the criterion.

The drawbacks are too large quantity and no orientation component. In ORB, Harris corner measure and intensity centroid are employed to improve the method. The intensity centroid assumes that the intensity of a corner is offset from its center which can be used to describe orientation. The moments of a patch is defined as:

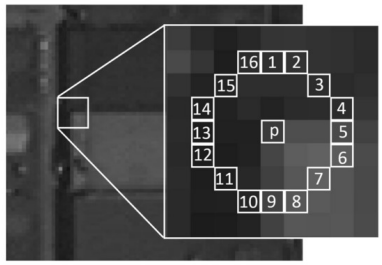


Figure 2.10: FAST corner detector

$$m_{pq} = \sum_{x,y} x^p y^q I(x,y) \tag{2.13}$$

with thees moments the centroid can be found as:

$$C = (\frac{m_{10}}{m_{00}}, \frac{m_{01}}{m_{00}}) \tag{2.14}$$

then a vector can be constructed from the center of the corner O to the centroid $\overrightarrow{OC}$. The orientation of the patch is:

$$\theta = \mathrm{atan2}(m_{01}, m_{10}) \tag{2.15}$$

where $\mathrm{atan2}$ is the quadrant-aware version of atan. To improve the rotation invariance, the moments are computed with x and y remaining within a circular region of radius r. r is chosen to be the patch size to make sure that x and y run from $[-r, r]$.

The BRIEF descriptor [19] is constructed from a set of binary intensity test, and it is a bit string description of an image patch. A binary test τ of a smoothed image patch $\mathbf{p}$ is defined as:

$$\tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) := \begin{cases} 1 : \mathbf{p}(\mathbf{x}) < \mathbf{p}(\mathbf{y}) \\ 0 : \mathbf{p}(\mathbf{x}) \geq \mathbf{p}(\mathbf{y}) \end{cases} \tag{2.16}$$

where $\mathbf{p}(\mathbf{x})$ is the intensity of $\mathbf{p}$ at a point $\mathbf{x}$. The feature is defined as a vector of n binary tests:

$$f_n(\mathbf{p}) := \sum_{1 \leq i \leq n} 2^{i-1} \tau(\mathbf{p}; \mathbf{x}, \mathbf{y}) \tag{2.17}$$

The vector length is chosen as $n = 256$ in ORB algorithm.

The construction and matching for ordinary BRIEF descriptor is fast, but the drawback is that this descriptor is unstable with rotation. In ORB, a more efficient and robust method called steered BRIEF is used instead. For any feature set of n binary tests at $(\mathbf{x}_i, \mathbf{y}_i)$, a $2 \times n$ matrix $\mathbf{S}$ is defined as:

$$\mathbf{S} = \begin{pmatrix} \mathbf{x}_1, ..., \mathbf{x}_n \\ \mathbf{y}_1, ..., \mathbf{y}_n \end{pmatrix} \tag{2.18}$$

The steered version $\mathbf{text}_\theta$ of $\mathbf{S}$ is constructed using the patch orientation θ and the corresponding rotation matrix $\mathbf{R}_\theta$ as:

$$\mathbf{S}_\theta = \mathbf{R}_\theta \mathbf{S} \tag{2.19}$$

Then the steered BRIEF descriptor becomes:

$$g_n(\mathbf{p}, \theta) := f_n(\mathbf{p}) | (\mathbf{x}_i, \mathbf{y}_i) \in \mathbf{S}_\theta \tag{2.20}$$

A lookup table of precomputed BRIEF patterns can be constructed. The angle is discretized to increments of $2\pi/30$ (12 degrees). The correct set of points $\mathbf{S}_\theta$ is used to compute its descriptor when the keypoint orientation θ is consistent across views.

2.5 Feature Matching

During feature matching, descriptors are compared between two images to identify similar features. The simplest method is Brute-Force Matcher. It takes the descriptor of one feature in first set and is matched with all features in second set using Hamming distance calculation. Then the closest one is returned as one matching result.

It is obvious that the computation increases with the number of features. An alternative is FLANN (Fast Library for Approximate Nearest Neighbors) based Matcher. FLANN was developed by Marius Muja et al. in 2009 [20]. This matcher contains a collection of algorithms that are optimized for fast nearest neighbor search in large datasets. It works faster than Brute-Force Matcher for large datasets.

2.6 Motion estimation

In feature matching stage, a set of matched 2D feature points and 3D map points is found. Estimating the pose of camera with matched 3D-2D points is called Perspective-n-Point (PnP) problem in SLAM. Consider a homogeneous world point $\mathbf{p}_w = [X, Y, Z, 1]^{\mathrm{T}}$ and the corresponding homogeneous pixel point $\mathbf{u} = [u, v, 1]$. From the pin-hole camera model:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}}_{\mathbf{K}} \underbrace{\begin{bmatrix} r_1 1 & r_1 2 & r_1 3 & t_1 \\ r_2 1 & r_2 2 & r_2 3 & t_2 \\ r_3 1 & r_3 2 & r_3 3 & t_3 \end{bmatrix}}_{[\mathbf{R}|\mathbf{t}]} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.21}$$

where s is a scale factor for pixel point, $\mathbf{K}$ is the camera intrinsic parameters, $\mathbf{R}$ and $\mathbf{t}$ are the camera extrinsic parameters that are being calculated.

There are some methods to solve PnP problem. Efficient PnP (EPnP) is a method developed by Lepetit, et al. in 2008 [21]. The solution has $O(n)$ complexity with n paired points, so the method is very efficient when n is large. The core idea is 3D point, which are called reference points $\mathbf{P}_i \, (i = 1, ..., n)$ in the original paper, can be expressed as a weight sum of four virtual control points$\mathbf{c}_j \, (j = 1, ..., 4)$:

$$\begin{aligned} \mathbf{P}_i^w &= \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^w \\ \mathbf{P}_i^c &= \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^c \end{aligned} \tag{2.22}$$

The ^{w} superscript is used if the point coordinates are expressed in the world coordinate system, and ^{c} superscript is used if those are expressed in the camera coordinate system. α_{ij} are homogeneous barycentric coordinates with:

$$\sum_{j=1}^{4} \alpha_{ij} = 1 \tag{2.23}$$

Then Equation (2.21) can be rewritten as:

$$s_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = \mathbf{K} \sum_{j=1}^{4} \alpha_{ij} \mathbf{c}_j^c \tag{2.24}$$

The homogeneous camera control point has the form $\mathbf{c}_j^c = [x_j^c, y_j^c, z_j^c]^{\mathrm{T}}$. Rewrite the camera reference point equation yields two linear equation for each reference point:

$$\begin{aligned} \sum_{j=1}^{4} \alpha_{ij} f_x x_j^c + \alpha_{ij}(c_x - u_i) z_j^c = 0 \\ \sum_{j=1}^{4} \alpha_{ij} f_y y_j^c + \alpha_{ij}(c_y - v_i) z_j^c = 0 \end{aligned} \tag{2.25}$$

Then using these two equations for all n reference points, a linear system $\mathbf{Mx} = \mathbf{0}$ can be generated, where $\mathbf{x} = [\mathbf{c}_1^{c\mathrm{T}}, \mathbf{c}_2^{c\mathrm{T}}, \mathbf{c}_3^{c\mathrm{T}}, \mathbf{c}_4^{c\mathrm{T}}]^{\mathrm{T}}$. The solution for the control points belongs to the null space of $\mathbf{M}$, and can be expressed as:

$$\mathbf{x} = \sum_{i=1}^{N} \beta_i \mathbf{v}_i \tag{2.26}$$

where N is the number of null singular values in M and the set $\mathbf{v}_i$ are the columns of the right-singular vector of M. After calculating the initial parameters β_i, the Gauss-Newton algorithm is used to refine them. Then the $\mathbf{R}$ and $\mathbf{t}$ can be calculated by minimizing the reprojection error of the world reference points p_i^w and their corresponding camera points p_i^c.

2.7 Graph optimization

2.7.1 Lie group and Lie algebra

After estimating camera poses, optimization is necessary to get better results. The poses obtained from EPnP method are rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$. It is obvious that $\mathbf{R}$ is an orthogonal matrix with a determinant of 1, which introduces additional constraints that makes optimization difficult. Through the transformation relationship between Lie group and Lie algebra, the pose estimation can be turned into an unconstrained optimization problem and simplify the solution. The notation of Timothy [22] is used in this section.

The three dimensional rotation matrix $\mathbf{R}$ constitutes special orthogonal group $SO(3)$, the Lie algebra of which is:

$$\mathfrak{so}(3) = \{\boldsymbol{\phi} \in \mathbb{R}^3, \boldsymbol{\Phi} = \boldsymbol{\phi}^\wedge \in \mathbb{R}^{3\times 3}\} \tag{2.27}$$

where the $^\wedge$ symbol represents the transformation from a vector to a skew-symmetric matrix:

$$\boldsymbol{\Phi} = \boldsymbol{\phi}^\wedge = \begin{bmatrix} 0 & -\phi_3 & \phi_2 \\ \phi_3 & 0 & -\phi_1 \\ -\phi_2 & \phi_1 & 0 \end{bmatrix} \tag{2.28}$$

The four dimensional transformation matrix $\mathbf{T}$ constitutes special Euclidean group $SE(3)$, the corresponding Lie algebra is:

$$\mathfrak{se}(3) = \left\{ \boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\phi} \end{bmatrix} \in \mathbb{R}^6, \boldsymbol{\rho} \in \mathbb{R}^3, \boldsymbol{\phi} \in \mathfrak{so}(3), \boldsymbol{\xi}^\wedge = \begin{bmatrix} \boldsymbol{\phi}^\wedge & \boldsymbol{\rho} \\ \mathbf{0}^{\mathrm{T}} & 0 \end{bmatrix} \in \mathbb{R}^{4\times 4} \right\} \tag{2.29}$$

In $\mathfrak{se}(3)$, a six dimensional vector is converted to a four dimensional matrix using the $^\wedge$ symbol, but no longer a skew-symmetric one. The relationship of Lie algebra to its associated Lie group is given by exponential map:

$$\mathbf{R} = \exp(\boldsymbol{\phi}^{\wedge}) \tag{2.30}$$

$$\mathbf{T} = \exp(\boldsymbol{\xi}^{\wedge}) \tag{2.31}$$

Detailed introduction and description can be found in [22].

A major motivation for using Lie algebra is to do optimization. The derivative is a necessary information in the optimization process. To find derivative on $SO(3)$, a practical method is using perturbation model, in other words, perturb $\mathbf{R}$ for$\Delta\mathbf{R}$ or $\mathbf{T}$ for$\Delta\mathbf{T}$ and see the change of the result relative to the disturbance. A left infinitesimal perturbation is multiplied on the Lie group multiplication, and use Lie algebra to describe the perturbation, then compute the derivative on this perturbation.

2.7.2 Bundle adjustment

Bundle adjustment (BA) is the problem of refining a visual reconstruction to produce jointly optimal 3D structure and viewing parameters (camera pose and calibration) [23]. Consider bundles of light rays leaving each 3D point, and they project in several image planes as pixel feature points. these light rays can converge on each camera optical center by making adjustment to the camera poses and 3D points. In SLAM, bundle adjustment boils down to minimize the projection error between observed and predicted image pixel locations.

The observation equation (1.2) can be described as: $\mathbf{x}_i$ is the pose of the camera $\mathbf{T}_i$, and its Lie algebra is $\boldsymbol{\xi}$, $\mathbf{y}_j$ is the 3D point $\mathbf{p}_j$, and the observed data $\mathbf{z}_{ij}$ is the corresponding pixel coordinates $[u_{ij}, v_{ij}]^{\mathrm{T}}$. Then the equation can be rewritten as:

$$\mathbf{z}_{ij} = h(\mathbf{T}_i, \mathbf{p}_j) \tag{2.32}$$

The error of this observation is given as:

$$\mathbf{e}_{ij} = \mathbf{z}_{ij} - h(\mathbf{T}_i, \mathbf{p}_j) \tag{2.33}$$

Now consider all observations in all times, a cost function can be established using

least-squares:

$$F(\mathbf{x}) = \frac{1}{2}\sum_{i}^{m}\sum_{j}^{n}\left\|\mathbf{e}_{ij}\right\|^2 = \frac{1}{2}\sum_{i}^{m}\sum_{j}^{n}\left\|\mathbf{z}_{ij} - h(\mathbf{T}_i, \mathbf{p}_j)\right\|^2 \tag{2.34}$$

Use first order Taylor expansion around variables to be optimized $\mathbf{x}$ to approximate the error function:

$$\mathbf{e}_{ij}(\mathbf{x} + \Delta\mathbf{x}) \approx \mathbf{e}_{ij}(\mathbf{x}) + \mathbf{J}(\mathbf{x})\Delta\mathbf{x} \tag{2.35}$$

where $\mathbf{J}_{ij}(\mathbf{x})$ is the Jacobian of $\mathbf{e}_{ij}(\mathbf{x})$. Then the cost function of single observation becomes:

$$F_{ij}(\mathbf{x} + \Delta\mathbf{x}) \approx \frac{1}{2}\left\|\mathbf{e}_{ij}\right\|^2 + \mathbf{e}_{ij}\mathbf{J}_{ij}(\mathbf{x})^{\mathrm{T}}\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^{\mathrm{T}}\mathbf{J}_{ij}(\mathbf{x})\mathbf{J}_{ij}(\mathbf{x})^{\mathrm{T}}\Delta\mathbf{x} \tag{2.36}$$

Then Equation 2.34 can be rewritten as:

$$\begin{aligned} F(\mathbf{x} + \Delta\mathbf{x}) &= \sum_{(i,j)\in\mathcal{O}} F_{ij}(\mathbf{x} + \Delta\mathbf{x}) \\ &\approx F(\mathbf{x}) + \mathbf{e}\mathbf{J}(\mathbf{x})^{\mathrm{T}}\Delta\mathbf{x} + \frac{1}{2}\Delta\mathbf{x}^{\mathrm{T}}\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^{\mathrm{T}}\Delta\mathbf{x} \end{aligned} \tag{2.37}$$

Minimize $\Delta\mathbf{x}$ by finding its derivative in Equation (2.37) and set it to zero:

$$\underbrace{\mathbf{J}(\mathbf{x})\mathbf{J}(\mathbf{x})^{\mathrm{T}}}_{\mathbf{H}(\mathbf{x})}\Delta\mathbf{x} = \underbrace{-\mathbf{J}(\mathbf{x})\mathbf{e}}_{\mathbf{g}(\mathbf{x})} \tag{2.38}$$

where $\mathbf{H}$ is the approximation of Hessian. Now the problem can be solved using Gauss-Newton algorithm.

In this thesis, two bundle adjustments are executed in both front end and back end parts. A small scale one in front end optimizes the processing frame camera pose, and 3D points are fixed. A large scale bundle adjustment in back end optimizes both several camera poses and map points. They will be discussed in the next two sections.

Algorithm 1 Gauss-Newton algorithm

1: set initial value $\mathbf{x}_0$
2: **while** $\mathbf{x}_k$ **do**
3: for iteration k, compute Jacobian $\mathbf{J}(\mathbf{x}_k)$ and error $\mathbf{e}(\mathbf{x}_k)$
4: solve $\mathbf{H}\Delta\mathbf{x}_k = \mathbf{g}$
5: **if** $\Delta\mathbf{x}_k$ is sufficiently small **then**
6: break
7: **else**
8: $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$
9: **end if**
10: **end while**

2.7.3 Front end optimization

After camera pose estimation using EPnP algorithm, a small scale optimization is executed. Only the camera pose of the processing frame is optimized. The observation equation becomes:

$$\begin{aligned} s_i\mathbf{u} &= \mathbf{K}\mathbf{P}' \\ \mathbf{P}' &= (\mathbf{T}\mathbf{P})_{1:3} \end{aligned} \tag{2.39}$$

$\mathbf{P}$ is a homogeneous map point, the first three rows are extracted before transforming camera coordinates to pixel coordinates to keeps dimension correct. Then there are:

$$\begin{aligned} u &= f_x\frac{X'}{Z'} + c_x \\ v &= f_y\frac{Y'}{Z'} + c_y \end{aligned} \tag{2.40}$$

To find the Jacobian, multiply a left infinitesimal perturbation $\delta\boldsymbol{\xi}$ on $\mathbf{T}$, which is represented by $\oplus$, and compute the derivative on this perturbation. The Jacobian is given as:

$$\mathbf{J} = \frac{\partial\mathbf{e}}{\partial\delta\boldsymbol{\xi}} = \lim_{\delta\boldsymbol{\xi}\to 0}\frac{\mathbf{e}(\delta\boldsymbol{\xi}\oplus\boldsymbol{\xi}) - \mathbf{e}(\boldsymbol{\xi})}{\delta\boldsymbol{\xi}} \tag{2.41}$$

Use the chain rule to computer two derivatives. The first part can be obtained by

Equation (2.40). The second part is the derivative of the transformed map point of the perturbation, the detail can be found in [22].

$$\begin{aligned}\frac{\partial \mathbf{e}}{\partial \delta \boldsymbol{\xi}} &= \frac{\partial \mathbf{e}}{\partial \mathbf{P}'} \frac{\partial \mathbf{P}'}{\partial \delta \boldsymbol{\xi}} = \frac{\partial \mathbf{e}}{\partial \mathbf{P}'} \frac{\partial (\mathbf{TP})_{1:3}}{\partial \delta \boldsymbol{\xi}} \\ &= - \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix} \begin{bmatrix} \mathbf{I} & -\mathbf{P}'^{\wedge} \\ \mathbf{0}^{\mathrm{T}} & \mathbf{0}^{\mathrm{T}} \end{bmatrix}_{1:3} \\ &= - \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} & -\frac{f_x X' Y'}{Z'^2} & f_x + \frac{f_x X'^2}{Z'^2} & -\frac{f_x Y'}{Z'} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} & -f_y - \frac{f_y Y'^2}{Z'^2} & \frac{f_y X' Y'}{Z'^2} & \frac{f_y X'}{Z'} \end{bmatrix} \end{aligned} \tag{2.42}$$

Graph optimization is the problem that combines nonlinear optimization and graph theory. It describes optimization problem as a graph. The graph is made up of vertices/nodes that are connected by edges. For SLAM problems, optimized variables are presented by vertices while errors are presented by edges.

To start graph optimization, vertices and edges should be defined first. As shown in Figure , set the estimated camera pose $\mathbf{T}$ as one vertex, set every projection point $\mathbf{z}_i$ in image plane as edges, and map points $\mathbf{P}_i$ are fixed and not optimized.

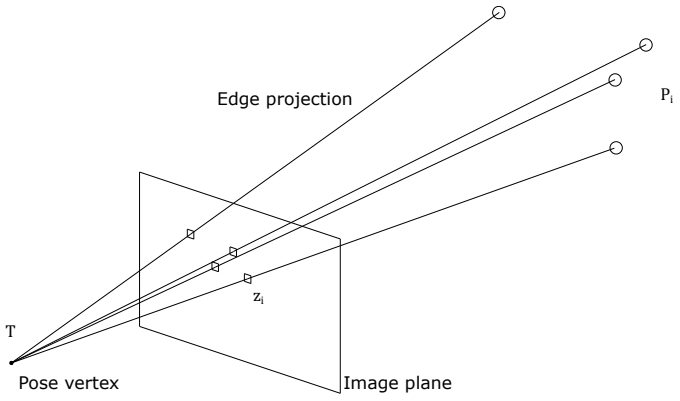


Figure 2.11: Front end optimization

2.7.4 Back end optimization

In back end part, both estimated camera poses and map points are optimized, as shown in Figure . The variables to be optimized is defined as:

$$\mathbf{x} = [\underbrace{\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_m}_{\mathbf{x}_c}, \underbrace{\mathbf{p}_1, ..., \mathbf{p}_n}_{\mathbf{x}_p}]^{\mathrm{T}} \tag{2.43}$$

The derivative of observation error of Lie algebra form camera pose is given in Equation (2.42). The derivative of observation error of 3D map point can be computed as:

$$\frac{\partial \mathbf{e}}{\partial \mathbf{p}} = \frac{\partial \mathbf{e}}{\partial \mathbf{p}'} \frac{\partial \mathbf{p}'}{\partial \mathbf{p}} = - \begin{bmatrix} \frac{f_x}{Z'} & 0 & -\frac{f_x X'}{Z'^2} \\ 0 & \frac{f_y}{Z'} & -\frac{f_y Y'}{Z'^2} \end{bmatrix} \mathbf{R} \tag{2.44}$$

The objective function becomes:

$$\begin{aligned} F(\mathbf{x}) &= \frac{1}{2} \sum_{i}^{m} \sum_{j}^{n} \left\| \mathbf{e}_{ij} + \mathbf{F}_{ij} \Delta \boldsymbol{\xi}_i + \mathbf{E}_{ij} \Delta \mathbf{p}_j \right\|^2 \\ &= \left\| \mathbf{e} + \mathbf{F} \Delta \mathbf{x}_c + \mathbf{E} \Delta \mathbf{x}_p \right\|^2 \end{aligned} \tag{2.45}$$

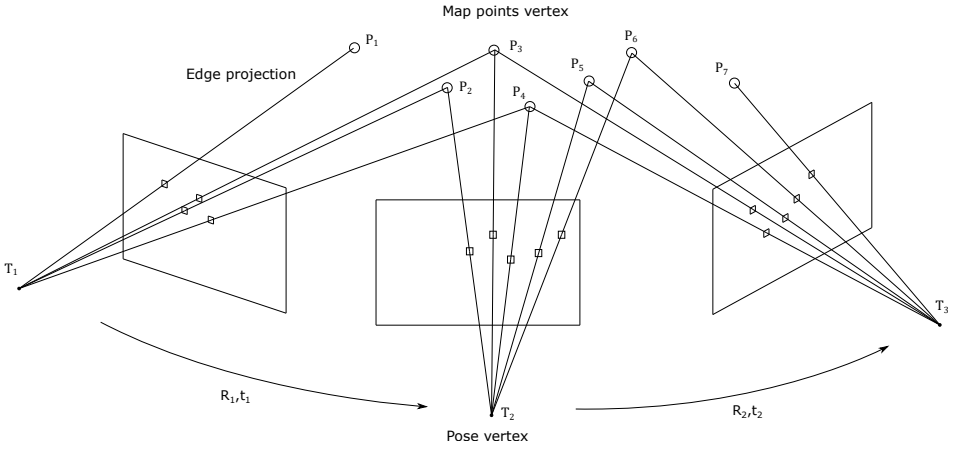


Figure 2.12: Back end optimization

where $\mathbf{F}_{ij}$ is the derivative of camera pose and $\mathbf{E}_{ij}$ is the derivative of map point. The full Jacobian is $\mathbf{J} = [\mathbf{F}\ \mathbf{E}]$. The Hessian in Gauss-Newton is:

$$\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{J} = \begin{bmatrix} \mathbf{F}^{\mathrm{T}}\mathbf{F} & \mathbf{F}^{\mathrm{T}}\mathbf{E} \\ \mathbf{E}^{\mathrm{T}}\mathbf{F} & \mathbf{E}^{\mathrm{T}}\mathbf{E} \end{bmatrix} \tag{2.46}$$

The direct computation of matrix $\mathbf{H}$ inversion is time-consuming because the complexity of matrix inversion is $O(n^3)$ and generally there are hundreds of map points so the matrix dimension is large. But the structure of matrix $\mathbf{H}$ is sparse, which can be used to speed up computation process.

Consider a simple example including 2 camera poses and 6 map points. Camera C_1 observes map points P_1, P_2, P_3 and P_4, and camera C_2 observes P_3, P_4, P_5 and P_6. The poses of 2 camera are $\mathbf{T}_i, i = 1, 2$ and the position of 6 map points are $\mathbf{p}_i, i = 1, \cdots, 6$. The problem can be described as a graph in Figure 2.13.

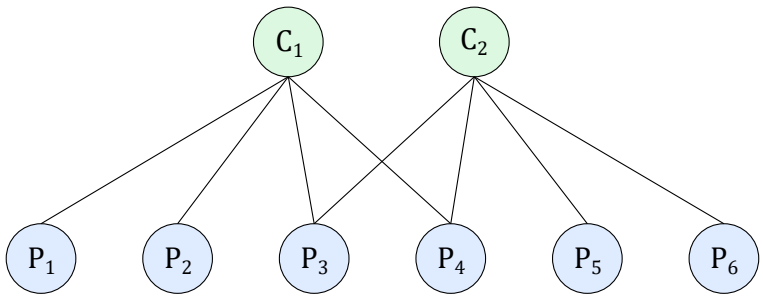


Figure 2.13: Represent camera poses and map points by a graph

The Jacobian of each error components $\mathbf{e}_{ij}$ is $\mathbf{J}_{ij}$. For $\mathbf{J}_{11}$, it is clear that the derivatives of $\mathbf{e}_{11}$ of camera pose $\boldsymbol{\xi}_2$ and map points $\mathbf{p}_2, \cdots, \mathbf{p}_6$ are zero:

$$\mathbf{J}_{11} = \frac{\partial \mathbf{e}_{11}}{\partial \mathbf{x}} = \Big(\frac{\partial \mathbf{e}_{11}}{\partial \boldsymbol{\xi}_1}, \mathbf{0}_{2\times 6}, \frac{\partial \mathbf{e}_{11}}{\partial \mathbf{p}_1}, \mathbf{0}_{2\times 3}, \mathbf{0}_{2\times 3}, \mathbf{0}_{2\times 3}, \mathbf{0}_{2\times 3} \Big) \tag{2.47}$$

For convenience to express sparsity patterns of $\mathbf{J}$ and $\mathbf{H}$, non-zero entries in matrices are indicated by $*$. The full Jacobian is:

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_{11} \\ \mathbf{J}_{12} \\ \mathbf{J}_{13} \\ \mathbf{J}_{14} \\ \mathbf{J}_{23} \\ \mathbf{J}_{24} \\ \mathbf{J}_{25} \\ \mathbf{J}_{26} \end{bmatrix} = \begin{array}{c} \begin{matrix} C_1 & C_2 & P_1 & P_2 & P_3 & P_4 & P_5 & P_6 \end{matrix} \\ \begin{bmatrix} * & & * & & & & & \\ * & & & * & & & & \\ * & & & & * & & & \\ * & & & & & * & & \\ & * & & & * & & & \\ & * & & & & * & & \\ & * & & & & & * & \\ & * & & & & & & * \end{bmatrix} \end{array} \tag{2.48}$$

Also, the Hessian of Gauss-Newton is:

$$\mathbf{H} = \mathbf{J}^{\mathrm{T}}\mathbf{J} = \left[\begin{array}{cc|cccccc} * & & * & * & * & * & & \\ & * & & & * & * & * & * \\ \hline * & & * & & & & & \\ * & & & * & & & & \\ * & * & & & * & & & \\ * & * & & & & * & & \\ & * & & & & & * & \\ & * & & & & & & * \end{array}\right] = \begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{E}^{\mathrm{T}} & \mathbf{C} \end{bmatrix} \tag{2.49}$$

The Hessian can be divided into four blocks. $\mathbf{B}$ and $\mathbf{C}$ are block diagonal matrices. In SLAM problems, generally the number of camera poses are far less than that of map points, thus $\mathbf{B}$ is smaller than $\mathbf{C}$. The shape of $\mathbf{H}$ looks like a arrow, and this structure is often referred to as an arrowhead matrix.

Now Equation (2.38) becomes:

$$\begin{bmatrix} \mathbf{B} & \mathbf{E} \\ \mathbf{E}^{\mathrm{T}} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_c \\ \Delta\mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix} \tag{2.50}$$

Schur complement method [22, 24] can be used to manipulate Equation (2.50) into a form that is more efficiently solved. This can be seen by premultiplying both sides by:

$$\begin{bmatrix} \mathbf{I} & -\mathbf{E}\mathbf{C}^{-1} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{2.51}$$

so that:

$$\begin{bmatrix} \mathbf{B} - \mathbf{E}\mathbf{C}^{-1}\mathbf{E}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{E}^{\mathrm{T}} & \mathbf{C} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}_c \\ \Delta\mathbf{x}_p \end{bmatrix} = \begin{bmatrix} \mathbf{v} - \mathbf{E}\mathbf{C}^{-1}\mathbf{w} \\ \mathbf{w} \end{bmatrix} \tag{2.52}$$

Now $\Delta\mathbf{x}_c$ can be easily solved since $\mathbf{C}$ is block-diagonal, $\mathbf{C}^{-1}$ is cheap to compute. The complexity of each solve down from $O((i+j)^3)$ without sparsity to $O(i^3 + ij^2)$. Generally j is far larger than i, so the computation time can be reduced using the structure of sparsity.

$$\begin{aligned} (\mathbf{B} - \mathbf{E}\mathbf{C}^{-1}\mathbf{E}^{\mathrm{T}})\Delta\mathbf{x}_c &= \mathbf{v} - \mathbf{E}\mathbf{C}^{-1}\mathbf{w} \\ \Delta\mathbf{x}_p &= \mathbf{C}^{-1}(\mathbf{w} - \mathbf{E}^{\mathrm{T}}\Delta\mathbf{x}_c) \end{aligned} \tag{2.53}$$

For real application, robust kernels are better than least-squares. When using least-squares objective function, if a bad matching exists, the observation data is wrong. However, the optimization algorithm cannot recognize it. So an edge with a large error is added to the graph, and its gradient is large, which means that adjusting the variables related to it causes the objective function to drop more. The algorithm tries to adjust the estimated values of the vertices connected to this edge so that they conform to the unreasonable requirements of this edge. Since the error of this edge is large, the algorithm tends to focus on optimize the wrong edge and smooth out the influence of other correct edges.

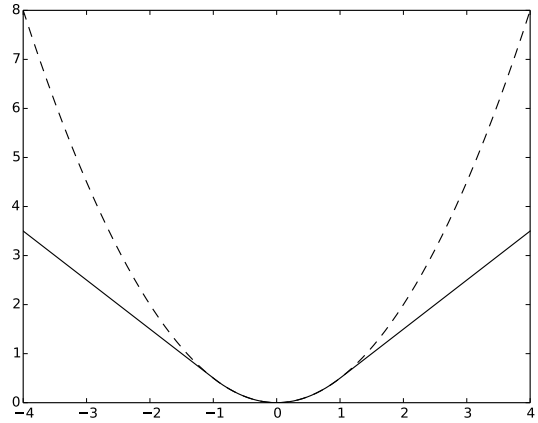


Figure 2.14: Huber kernel (solid line) and quadratic function (dotted line)

The reason of the problem is that quadratic objective function increases rapidly when error are large. There are some robust kernels to avoid the problem. A commonly used one is called Huber kernel:

$$H(e) = \begin{cases} \frac{1}{2}e^2 & \text{for } |e| < \delta \\ \delta\left(|e| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{2.54}$$

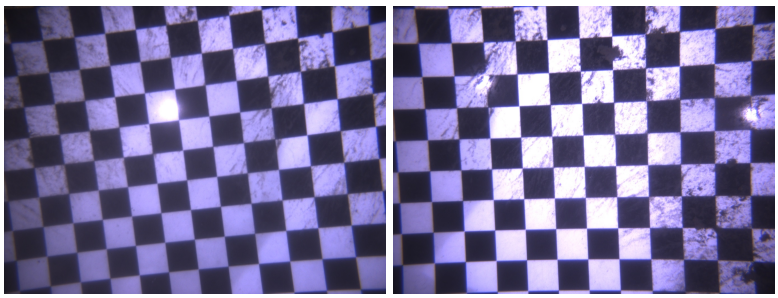
When error e is larger than the given threshold δ, objective function growth changes from quadratic form to primary form, in other words, the maximum value of gradient is limited. Figure 2.14 [25] shows the comparison between Huber kernel and quadratic function. It can be seen that the growth of Huber kernel is significantly lower than that of quadratic function when the error is large.

Chapter 3

Method

3.1 Camera calibration

Camera calibration is the process to estimate camera intrinsic parameters and distortion coefficients, which are given by Equation 2.4 and Equation 2.11. Camera installation location and work environment may be different in different missions, camera calibration is necessary before experiments. Generally the camera calibration procedure is implemented by finding a calibration target first. The most popular calibration target form is the chessboard pattern. An example of underwater environment chessboard is shown in Figure 3.1. The pictures were shot by stereo camera in the mission at Referansevraket in 2014. The pattern is mounted



(a) Left camera **(b)** Right camera

Figure 3.1: Camera calibration by chessboard pattern

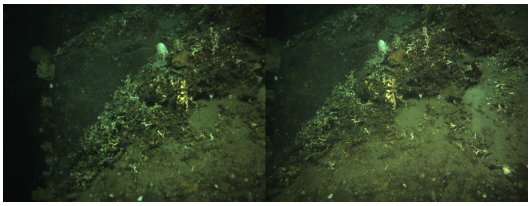
onto a rigid flat surface and is placed in the work environment. Then the camera takes 10-20 pictures of the target at different distances and orientations. These pictures are used to compute the parameters.

There are many tools have the ability to process the pictures and calibrate cameras, such as OpenCV and MATLAB. OpenCV provides functions to find the position of internal corners of the chessboard, and use the coordinates of all corners to compute intrinsic parameters by Zhang's camera calibration method [26] and distortion coefficients by Brown's distortion model [10], which is described in Section 2.1.3. To successfully detect desired corners in the first step, the function require uniform and proper light condition when the pictures are taken. Occlusion may also cause detection failure. After test, it is found that the chessboard in underwater environment, like Figure 3.1, is hard to detect corners using OpenCV. The tool that used for the NTNU ROV is the commercial software Agisoft Metashape. The software can automatically process the images and calculate calibration parameters.

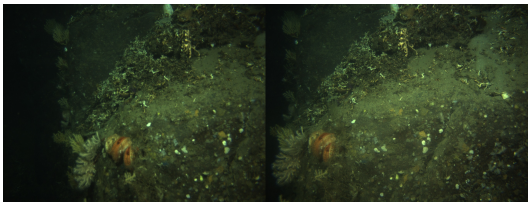
3.2 Feature detection and description

There are many feature detection and description algorithms. Good algorithms should be invariant to rotation, scale and viewpoint change, so that the features can be detected in different images. SIFT, SURF, ORB, BRISK, KAZE and AKAZE are some popular algorithms. In [27], these algorithms are compared from the aspects of quantity, computation time, model fitting, repeatability and error between recovered results and ground-truths. The results are that ORB and BRISK are the most efficient algorithms that can detect huge amount of features, the overall accuracy of SIFT and BRISK is the highest and SIFT is concluded as the most accurate algorithm. All these algorithms are provided in OpenCV except SIFT.

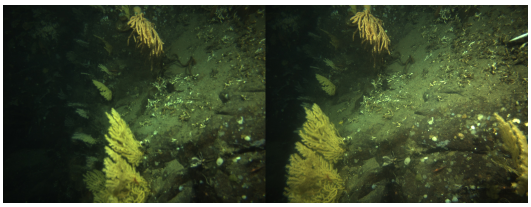
ORB and BRISK were tested in the project report in last semester. Three pairs images from a ROV mission at Stokkbergneset in 2017 were used. The images show complex seabed, so it is easy to detect features. They were processed by histogram equalization before testing. All parameters in OpenCV are default, excepting the following two parameters. The maximum number of features in ORB function is set to 5000. The AGAST detection threshold score of BRISK is 50 (default value 30). Under this value more features can be detected by BRISK.



(a) A



(b) B



(c) C

Figure 3.2: Three pairs images for ORB and BRISK tests

Method	BRISK			ORB		
Images	A	B	C	A	B	C
Number of features in first image	5862	3049	790	5000	5000	5000
Number of features in second image	10226	6177	3065	5000	5000	5000
Number of matches	538	315	178	422	477	733
Number of matches after RANSAC	451	260	148	383	395	432
Detection & description time [ms]	358.85	232.48	137.25	220.51	230.47	195.64
Matching time [ms]	130.13	41.34	5.60	52.11	51.06	51.01
RANSAC time [ms]	0.78	0.66	0.51	0.57	0.75	1.13
Total time [ms]	530.63	315.67	183.67	313.70	323.60	289.44

Table 3.1: Comparison of ORB and BRISK results

The results are listed in Table 3.1. Both two algorithms can detect large number of features. ORB algorithm reaches the upper limit in all tests while BRISK algorithm detects less features in test C. It can be seen that for all three tests the number of features detected by BRISK in first image are far less than those in second image.

The reason is the left parts of three first images are too dark to detect features. As for computation time, it totally takes 0.3s on average using ORB algorithm and 0.2s to 0.6s using BRISK algorithm. The time highly depends on feature numbers, since feature detection and description occupies most of time, up to 65% - 75%. It can be concluded that both two algorithms work well in three tests. ORB performs better than BIRSK in feature detection. The difference in computation time is not significant. Considering robustness and efficiency, ORB is chosen for the SLAM system.

A big drawback of OpenCV ORB algorithm is that the keypoints are always concentrated in some parts of a image that have rich texture. One example is shown in Figure 3.4a , most of the keypoints are detected from the central part of the image. If a fish cover this region in next frame, only small number of matches can be found for pose estimation, which may reduce estimation accuracy. A improved method is split images using grids, and detect a reasonable amount of keypoints in every cell. So the distribution of keypoints is uniform. For the cells that is hard to detect keypoints due to poor texture, decrease the FAST threshold in these cells and detect again. The algorithm is shown in Algorithm 2.

Algorithm 2 Keyoints detection

1: Set default FAST threshold and minimum FAST threshold
2: Read processed image
3: Draw grids, the size is 30×30
4: Detect FAST corner in every grid using default FAST threshold
5: **if** No corner is detected **then**
6: Detect FAST corner in every grid using minimum FAST threshold
7: **end if**
8: Filter FAST corner based on quadtree
9: **return** All keypoints

After keypoints are detected. A quadtree based method is implemented to remove concentrated keypoints [28]. A example is shown in Figure 3.3. The aim of this example is to keep up to 25 keypoints. A region that are split by lines is called a node. In every step 1 node is divided into 4 nodes. If the number of nodes that contain keypoints is smaller than 25, repeat the same step. Otherwise, every node has at least one keypoints, then keep one keypoint that has the best quality in every node and discard others.

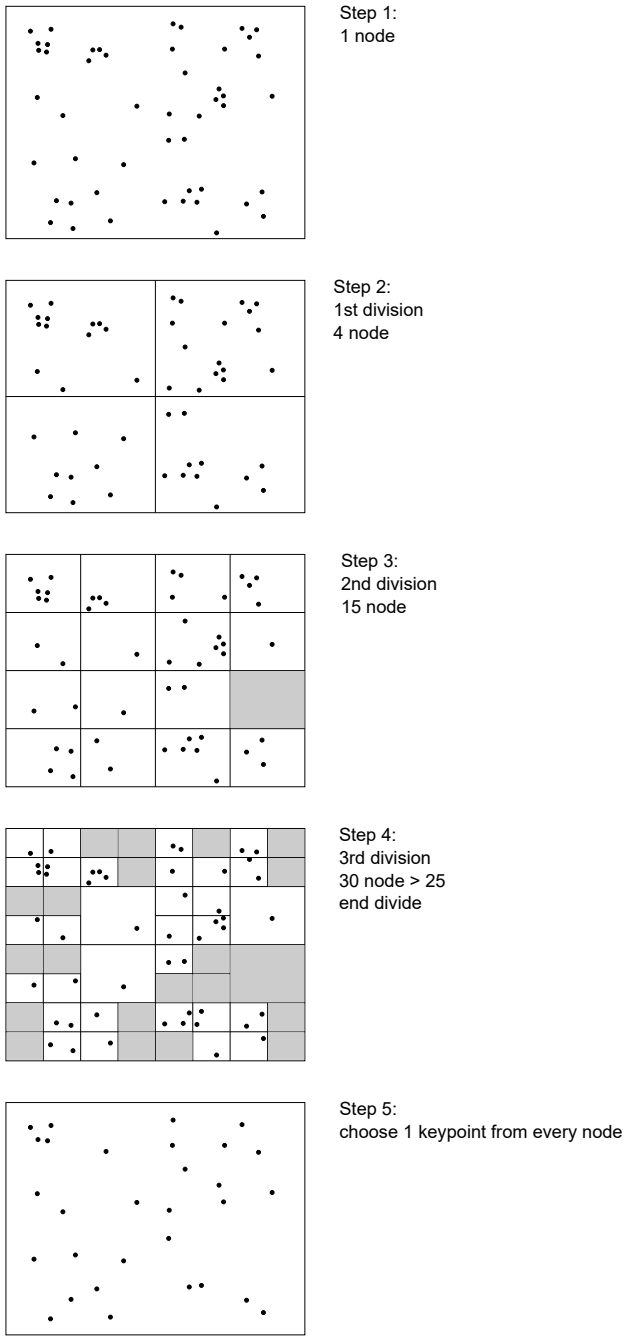
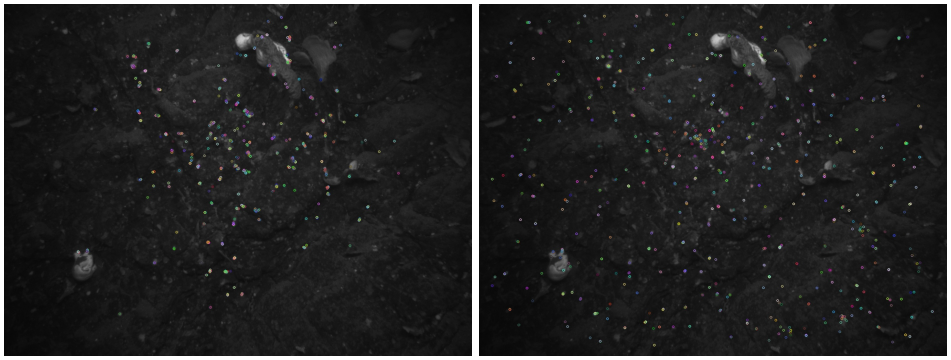


Figure 3.3: An example of filter keypoints based on quadtree



(a) OpenCV ORB algorithm **(b)** Quadtree based ORB algorithm

Figure 3.4: Keypoints detection results comparison between two algorithms

A comparison of keypoints detection results between the OpenCV ORB algorithm and quadtree based ORB algorithm is shown in Figure 3.4. It is clear that the distribution of keypoints becomes uniform using the improved algorithm.

3.3 Feature matching

Feature matching is implemented twice during one keyfreme processing. First, the descriptors of left image features and map points are used to match image 2D pixel points and 3D map points. Next, the descriptors of left and right images features are used to find the projected pixel points in left and right image of the same world points. FLANN based matcher is used to speed up the matching process.

Generally the matching results are not good enough to estimate camera pose or compute map points because mismatching is hard to avoid. So it is necessary to remove these wrong matches. Two methods are used together in the SLAM system: distance filter and RANSAC.

Distance filter can filter the matches that their matching distances are much larger than those of correct matches. The minimum distance is found firstly. A matching ratio is generally chosen between 1.5 and 2.5. If the distance of one match is less than the minimum distance times the matching ratio, it is labeled as a good match. Sometimes the distance of wrong matching may be small, so a minimum threshold is needed. It is a empirical parameter that the value is usually between 20 to 40. If the minimum distance timing the matching ratio is smaller than this threshold, use

the threshold instead to make decisions.

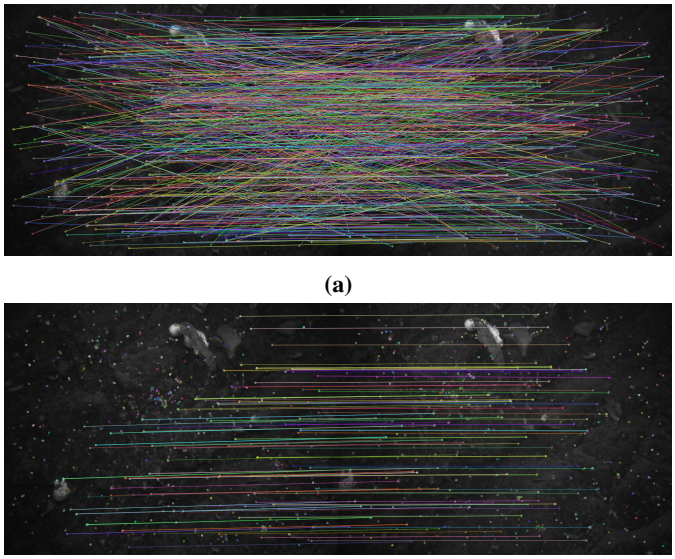
The Random Sample Consensus (RANSAC) is an iterative method to detect outliers from a set of observed data. Detailed introduction of RANSAC can be found in [29]. The outliers in feature matching are the mismatches. In OpenCV, *findHomography* function can find the perspective transformation between points in two planes. RANSAC is used to estimate the outlier of the transformation. Since the points are matched, RANSAC also estimates the outlier of matches.

Algorithm 3 Feature matching

```
Read left image descriptors vector
for i = 1 : m, m is the number of all map points do
    if Map points is observed in this frame then
        Add the descriptor of this map point to vector
    end if
end for
Match features using two descriptor vector
Compute minimum distance
Set min_dist = 0
for i = 1 : n, n is the number of matches do
    if Matcher distance < min_dist then
        min_dist = match distance
    end if
end for
Set matching ratio r
Set minimum matching threshold t
for i = 1 : n, n is the number of matches do
    if matcher distance < max(min_dist · r, t) then
        Good match
    end if
end for
Compute RANSAC status s
for i = 1 : k, k is the number of good matches do
    if s ≠ 0 then
        Correct match
    end if
end for
```

The matching algorithm between left image and map points is given in Algorithm 3. As for stereo matching, the algorithm is is almost same, and the difference is match objects. Map points are replaced by right image features. To judge whether map points are observed by current frame, all map points are projected from world coordinate to pixel coordinate. Since the pose of current frame is unknown, it is set equal to the pose of last frame temporarily. If the pixel coordinates $[x, y]^{\mathrm{T}}$ of map points are in the range of $[0, 0]^{\mathrm{T}}$, $[0, w]^{\mathrm{T}}$, $[l, w]^{\mathrm{T}}$ and $[l, 0]^{\mathrm{T}}$, where l is the length of the image and w is the weight, the map points are considered within observation.

The results of left and right images feature matching are shown in Figure 3.5. All mismatches can be removed after distance filter and RANSAC.



(a)

(b)

Figure 3.5: Feature matching result before (a) and after (b) removing mismatches

3.4 Camera pose estimation

After feature matching, the set of matched left camera image 2D keypoints and 3D map points is found. The EPnP algorithm is implemented to estimate the transformation from map points to their corresponding pixel points in the left camera image. The method is described in Section 2.6. The OpenCV function *solvePnP* is used and the outputs are rotation vector and translation vector. The rotation $\mathbf{r}$

vector can be described by a vector $\mathbf{n}$ in the direction of the rotation axis, and the angle of rotation θ. The rotation vector $\mathbf{r}$ can be converted to a rotation matrix $\mathbf{R}$ using Rodrigues's Formula:

$$\begin{aligned} \theta &= \mathrm{norm}(\mathbf{r}) \\ \mathbf{n} &= \frac{\mathbf{r}}{\theta} \\ \mathbf{R} &= \cos\theta\mathbf{I} + (1-\cos\theta)\mathbf{n}\mathbf{n}^{\mathrm{T}} + \sin\theta\mathbf{n}^{\wedge} \end{aligned} \tag{3.1}$$

where *norm* is the function to calculate the norm of $\mathbf{r}$.

The optimization in both front end and back end is implemented using g2o [9]. The theory of graph optimization is described in Section 2.7. Camera poses and map points position are set as g2o vertices, and the pixel coordinates of observed map points are set as edges. The sparsity property is used when process map points vertices to reduce computation time.

3.5 System implementation

The SLAM system is programmed in C++ based on the framework by Xiang Gao [30]. Some third party libraries are introduced in Section 1.1.3. An overall flow chart of the system is shown in Figure 3.6. The system contains two separated program. The main program visual odometry is designed to estimate camera pose. The other program TCP client is used to convert coordinates and communicate with the ROV system.

The visual odometry program uses the theory described in chapter 2 and the methods described in section 3.2 to 3.4. The components of visual odometry is shown in Figure 3.7 . Several C++ calsses are defined to store and process different types of data. *camera* class saves cameras data, including camera intrinsic parameters and distortion coefficients. It also contains the function of three coordinates transformation: world, camera and pixel. *dataset* is designed to find image files and initialize cameras. *feature* class defines the feature pointer to save detected ORB keypoints and the corresponding descriptors, observed frames and matched map points. *frame* defines the frame pointer. When a new frame is added, all data is stored in this class, including id, camera pose, left and right images, keyframe marker and keyframe id. *map* class includes all map points and keyframes, and

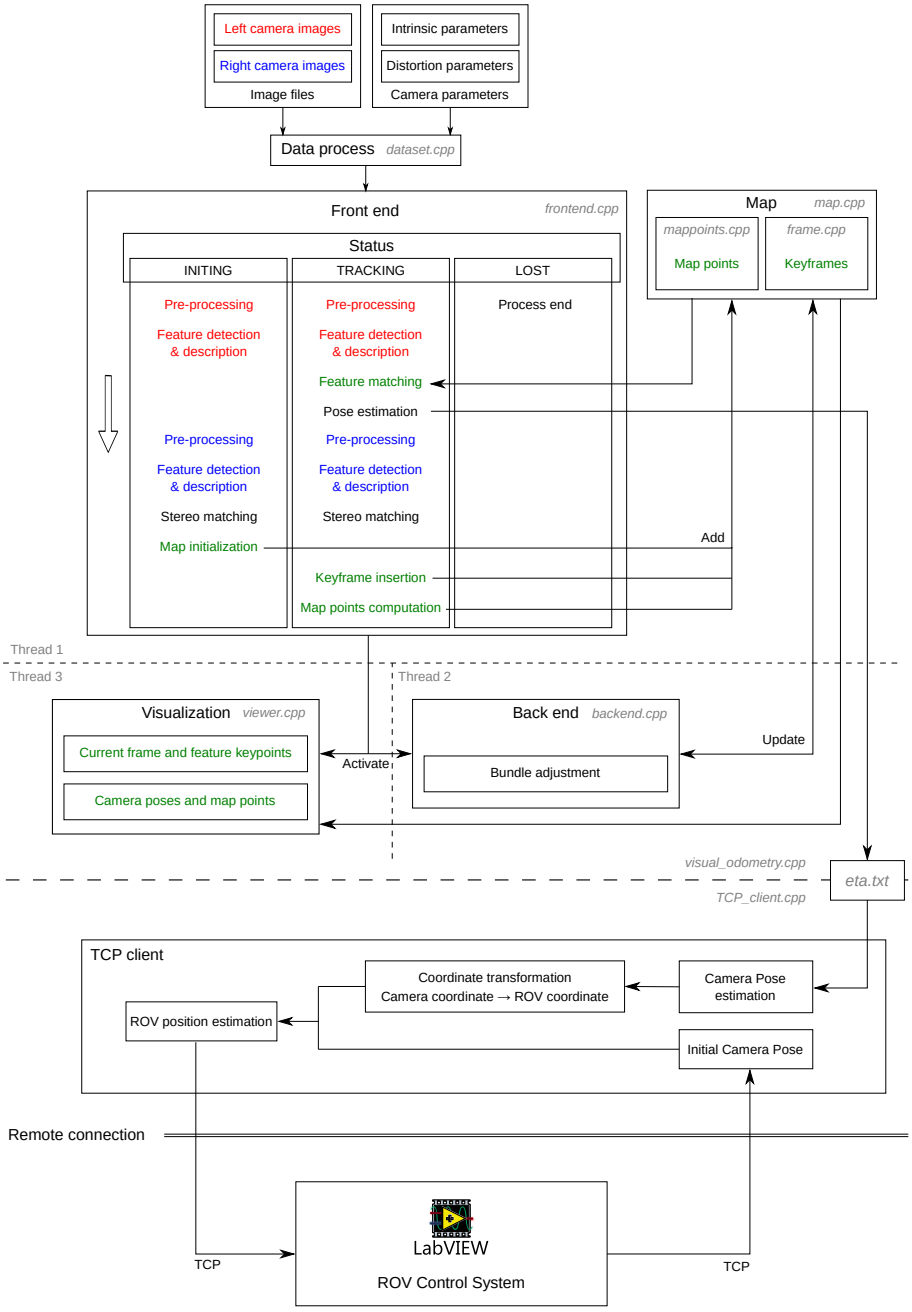


Figure 3.6: Flow chart of the SLAM system

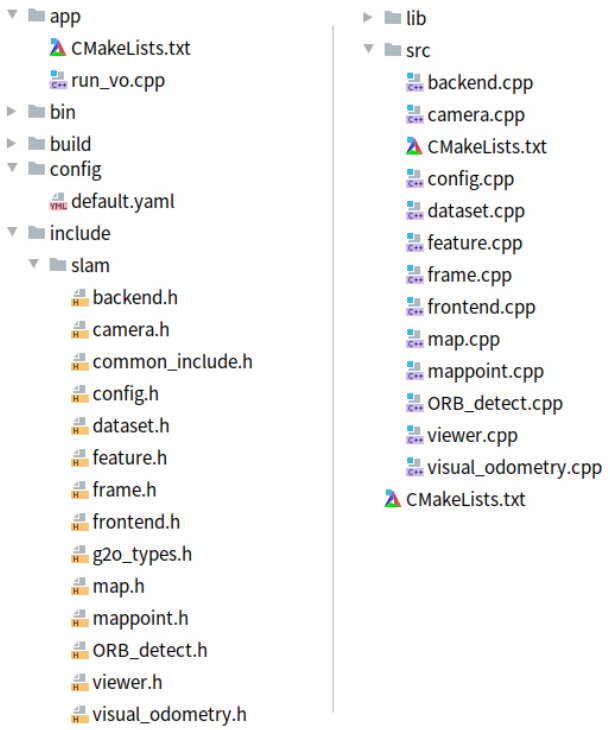


Figure 3.7: SLAM system source files

functions about adding and removing operation. *config* is designed to find the configuration file *default.yaml*. This file has important parameters, such as camera intrinsic parameters and ORB parameters. It is separated from other code so that the values can be edited without building the whole program. In *g2o_types*, the vertices, edges and relative functions used in optimization are defined. In *ORB_detect*, the algorithm of ORB keypoints detection is included. *frontend* and *backend* are the most important classes. *frontend* has all functions to process image, detect and describe features, features matching, pose estimation, new map point computation and activate back end thread and visualization thread. *backend* has the functions that optimize keyframes and map points. *viewer* is designed to visualize features, camera poses and map points. *visual_odomatry* is used to run the program and pass images to front end.

When the program starts running, a txt file including the directories of all image files is read and stored in memory. Camera information is read from the configuration file and camera initialization is executed and sent to front end. The first

left image and right image are also sent to front end as current frame. There are 3 different statuses in front end: INITING, TRACKING and LOST. The status is INITING when front end is activated, both left and right images are pre-processed. They are converted from color images to grayscale images. After that CLAHE is used to enhance image contrast so that it is easier to detect features. Then the improved ORB algorithm is used to detect and describe features. FLANN based matcher matches features between left image and right image. Distance filtering and RANSAC algorithm is used to remove outliers. The corresponding map points of matched features are computed using the pin-hole camera model and stereo camera model. If there are enough map points, set current frame as a keyframe and build the initial map. Add current frame and map points into the map. Then the status is changed to TRACKING. If the number of map points is small, the initial map is not good enough to start tracking because the following frame cannot match enough map points to get a good estimation result. In this case, map initialization is failed and the next frame repeats INITING algorithm. The algorithm of INITING status is outlined in Algorithm 4.

Algorithm 4 Front end algorithm of INITING status

1: $status$ = INITING
2: Read new left and right images
3: Pre-process left image
4: Detect and describe features from left image
5: Pre-process right image
6: Detect and describe features from right image
7: Match features from left and right images
8: Initialize map, Calculate new map points
9: **if** number of map points $>$ required minimum number **then**
10: Set current frame as keyframe
11: Add current frame and all map points to map
12: Activate back end
13: Update viewer
14: $status$ = TRACKING
15: **end if**

Map points computation algorithm is shown in Algorithm 5. The stereo camera model is used to find the depth of map points. After that, map points are calculated from the view of left camera, so left camera intrinsic parameters and distortion coefficients are used in the algorithm. Then transform points from camera coordinate

to world coordinate using the estimated camera pose.

Algorithm 5 Map points computation

n matched pixel points P_L and P_R
for $i = 1 : n$ **do**
  Disparity $d = u_L - u_R$
  Depth $Z = \frac{fb}{d}$
  $X = \frac{u_L - c_x}{f} Z$
  $Y = \frac{v_L - c_y}{f} Z$
  Compute normalized coordinates
  $x = \frac{X}{Z}$
  $y = \frac{Y}{Z}$
  Undistort
  $r = \sqrt{x^2 + y^2}$
  $x' = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$
  $y' = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$
  Compute undistorted 3D points
  $X = x' Z$
  $Y = y' Z$
  Map point in camera coordinate $P = [X, Y, Z]^{\mathrm{T}}$
  Coordinates transformation
  Map point in world coordinate $P_w = T^{-1} P$, T is the pose of current frame
end for
return All 3D map points

If initialization is successful, front end status becomes TRACKING. Program starts to estimate camera pose and send data to TCP client program. The algorithm for this status is given in Algorithm 6. Left image is processed to find keypoints and descriptors. All observed map points and left image features are matched to find corresponding relationship. After removing mismatches, EPnP method is implemented to estimate left camera pose. Then make decision whether the result is trustworthy. If the number of correct matches is smaller than a minimum value, which is generally set between 10 to 20, the results is considered to be not trusted. In other words, the estimation is bad and tracking is lost. Change status to LOST and stop processing current frame. Otherwise, the result is considered to be good. Then make another decision. the number of correct matches is larger than a minimum value that required a new keyframe, which is 50 in this program, the change of camera pose is small, so there is no need to add new map points. The process

of current frame is finished. Otherwise, set current frame as keyframe. Detect features from right image and match them with left image features. Use Algorithm 5 to calculate new map points and add current frame and map points into map. Activate back end to optimize camera poses and map points. Keep the status unchanged.

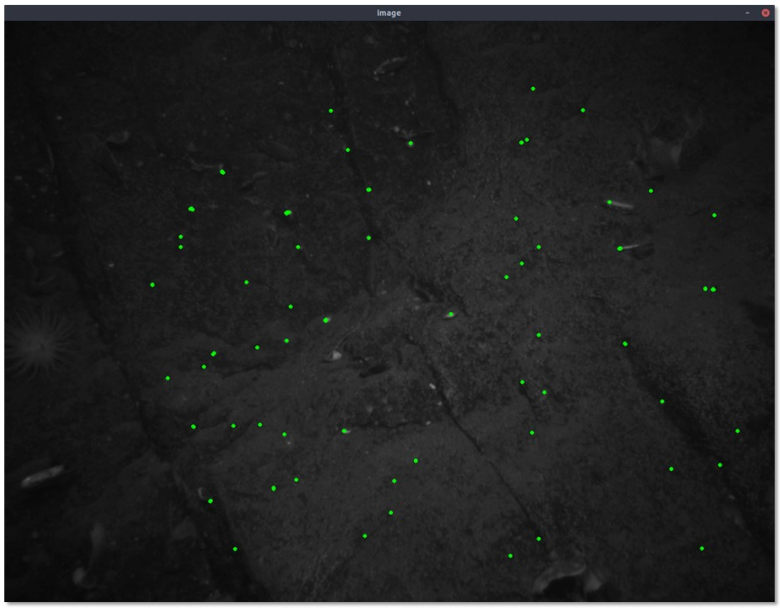
Algorithm 6 Front end algorithm of TRACKING status

1: $status$ = TRACKING
2: Read new left and right images
3: Pre-process left image
4: Detect and describe features from left image
5: Feature matching, n = number of matched features
6: Camera pose estimation
7: Write camera pose into "eta.txt"
8: Read the value of minimum features number for tracking n_t
9: **if** $n < n_t$ **then**
10: Tracking is not good
11: $status$ = LOST
12: **end if**
13: Read the value of minimum features number that require a new keyframe n_k
14: **if** $n < n_k$ **then**
15: Set current frame as keyframe, add it to the map
16: Pre-process right image
17: Detect and describe features from right image
18: Match features from left and right images
19: Calculate new map points
20: Add all new map points to map
21: Activate back end
22: Update viewer
23: **end if**

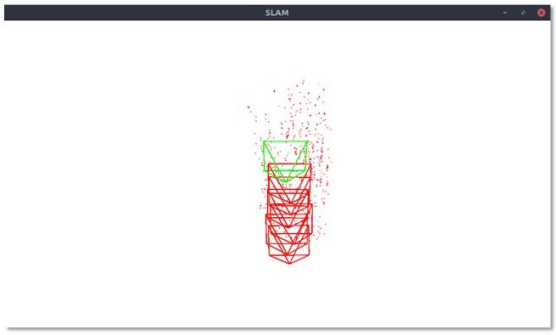
As for LOST status, the tracking is lost so the program cannot process next frame. An error message is sent and the program is stop.

Back end is activated by front end to optimize camera poses and map points. It runs in another thread, so front end operation is not affected. It is slow to optimize all camera poses and map points because the size of them increases with time. In this SLAM system only 7 latest keyframes and new observed map points are optimized. These keyframes are set as active keyframes, and all observed map points are set as active map points. Other keyframes and map points are fixed.

When a new keyframe is added into map, it is labeled as active keyframe while one keyframe is removed from active keyframe. The method that decides which one should be chosen is calculating the distance between the new keyframe and all other keyframes. If the minimum distance is smaller than a threshold, which means these two keyframes are too close, remove the nearest keyframe. Otherwise remove the farthest keyframe. Optimization is implemented using g2o library.



(a)



(b)

Figure 3.8: Visualization

To make it easier to monitor the program state, a Pangolin library based viewer for visualization is included in the SLAM system. The visualization thread is activated when the program starts running. In Figure 3.8, one window shows camera poses of active keyframes and all active map points. The latest keyframe is green while all others are red. Another window shows left image of current frame and feature keypoints (green points). Every time a new keyframe and new map points are added into the map in front end thread, viewer updates the display.

The TCP client program is used to send data to ROV control system remotely. The outputs of visual odometry program are estimated camera poses, which is written in file *eta.txt*. The form is: $x\ y\ z\ \phi\ \theta\ \psi$. TCP client read the data from that file. The reason to for this is two programs can run separately to avoid the interaction. If visual odometry crashes due to unexpected error, TCP lient can keep communicating with remote TCP server.

Once the TCP connection is successfully established, the program receives the ROV position in string form when the SLAM system starts running. The form is: $\$x, y, z, \phi, \theta, \psi$. The program read camera pose from *eta.txt*. The pose of camera is converted to the position of ROV based on the left camera coordinates in ROV body frame. Then convert ROV local position to global position by adding the initial position. After that, the global position is sent as an array to TCP server every 2 seconds.

Algorithm 7 TCP communication

```
1:  Read server IP address and port number
2:  Try to connect with server
3:  if Connected then
4:      if No initial value then
5:          Receive initial value from server
6:      end if
7:      loop
8:          Read camera pose from "eta.txt"
9:          Convert camera pose to ROV position
10:         Convert local position to global position
11:         Send ROV global position to server
12:     end loop
13:     Stop connection
14: end if
```

The TCP server is developed in LabVIEW on Windows platform by MSc student Fan Gao. When TCP server revives estimated ROV position from the SLAM system, it process the data and send it to ROV HIL simulator for further use.

Chapter 4

Simulation

The performance of the SLAM system is tested through three different simulations using two data sets. The ROV position data transmission between the SLAM system and the ROV control system via TCP is also tested. This chapter presents simulation set-up, results and comments in three different scenarios.

4.1 Stokkbergneset data set

The first two simulations are based on the image set and navigation data from a mission at Stokkbergneset in February 2017 by ROV SF-30k. The information of this ROV is described in Appendix 6.2. The cameras used to record images are two Allied Vision Prosilica model GC-1380 cameras. The specifications of the cameras are described in Appendix 6.2. Two cameras are installed at the front of the ROV, and are put downwards to collect seabed image data. The position of two cameras in the ROV body frame are given as:

$$\begin{aligned} \mathbf{p}_L &= [1.21, -0.190, 0.05]^{\mathrm{T}} \\ \mathbf{p}_R &= [1.21, 0.155, 0.05]^{\mathrm{T}} \end{aligned} \tag{4.1}$$

Camera calibration was implemented by Agisoft software before the mission to obtain camera intrinsic parameters and distortion coefficients. The calibration re-

sults are listed in Table 4.1.

	Left camera	Right camera
f	1692.56	1701.64
c_x	-9.79	-4.96
c_y	14.38	-8.30
k_1	0.5163	0.4995
k_2	0.6001	0.7297
k_3	2.9574	2.6416

Table 4.1: Camera intrinsic parameters and distortion coefficients estimation of Stokkbergneset data set

4.1.1 Scenario 1

In this scenario, the ROV moves in a lawnmower pattern in horizontal plane and moves up and down in vertical direction. The images are captured at a depth of 100m. An left and right camera image pair example in this image set is shown in Figure 4.1. The seabed in this image set is rocky and little creatures can be found. 60 pairs images with a time interval of 2 seconds are used, giving a total 120 seconds simulation time.

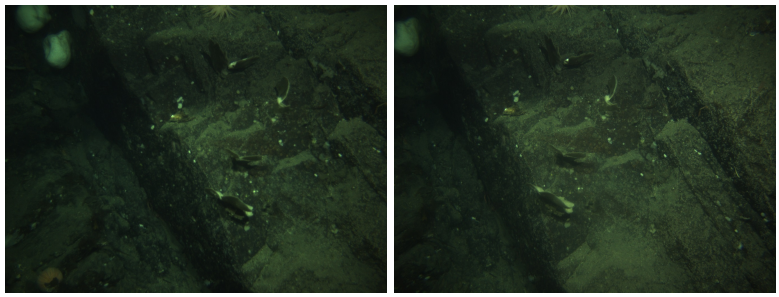


Figure 4.1: Left and right camera image pair example of Stokkbergneset data set at a depth of 100m

The results from this scenario are presented in Figure 4.2, 4.3 and 4.4. The North-East plot of ROV trajectory estimation is presented in Figure 4.2. The estimated change of ROV in position is presented in Figure 4.3, and estimated change in orientation in Figure 4.4. The navigation data from ROV control system is also plotted for comparison.

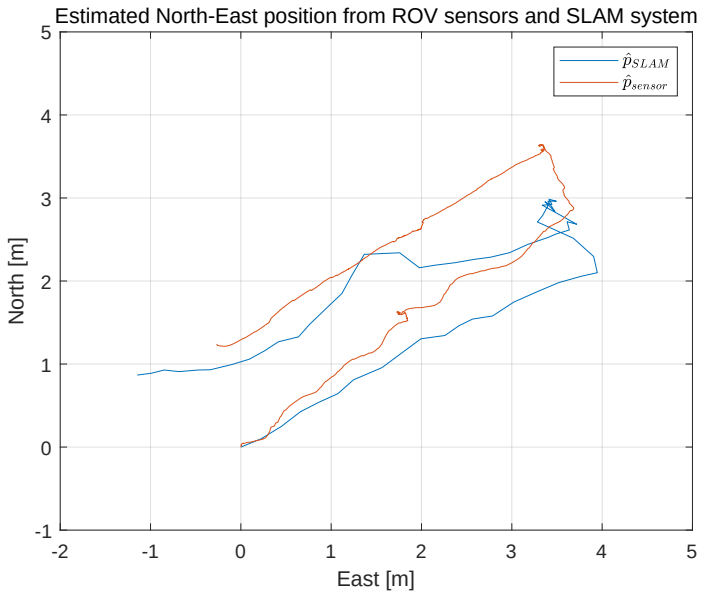


Figure 4.2: Simulation results scenario 1: North-East plot of estimated trajectory of the ROV from SLAM system and ROV sensors

From Figure 4.2, the SLAM system successfully estimate North-East position of the ROV in horizontal plane in this scenario. But the results are not completely correct, and errors exist both North and East direction. The error in surge becomes large after 60 seconds. The error in sway direction increases in the first 40 seconds and the last 30 seconds. The reason is that estimated orientation of ROV is not good when simulation starts. From Figure 4.4, the estimation in yaw angle increases in the first 20 seconds.

Besides, heave estimation is not good. The ROV moves up 10m and moves down to the original depth in this scenario. But the results shows that the ROV have slight heave change. Instead, significant change occurs in pitch angle.

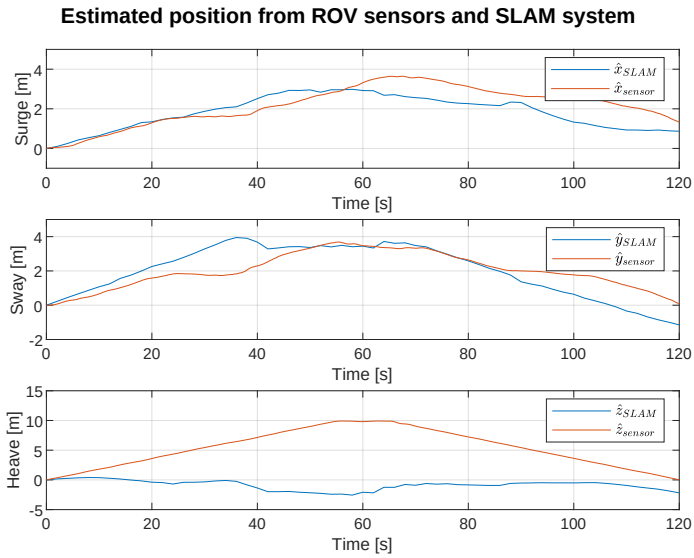


Figure 4.3: Simulation results scenario 1: plot of estimated surge, sway and heave from SLAM system and ROV sensors

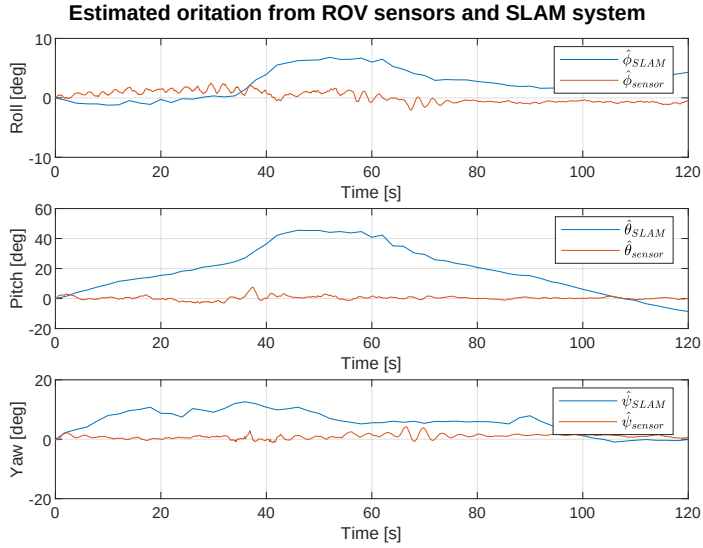


Figure 4.4: Simulation results scenario 1: plot of estimated orientation from SLAM system and ROV sensors

4.1.2 Scenario 2

In this scenario, the ROV movement pattern is the same as first scenario. The images are captured at a depth of 250m. An image pair example in this depth is shown in Figure 4.1. Lots of coral and shellfish grow on the sea floor in the observation area, and fish cover seabed in some images. Totally 90 pairs images with a time interval of 2 seconds are used, giving a total 180 seconds simulation time.

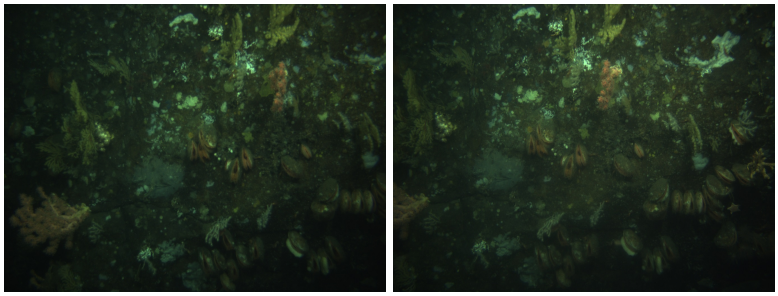


Figure 4.5: Left and right camera image pair example of Stokkbergneset data set at a depth of 250m

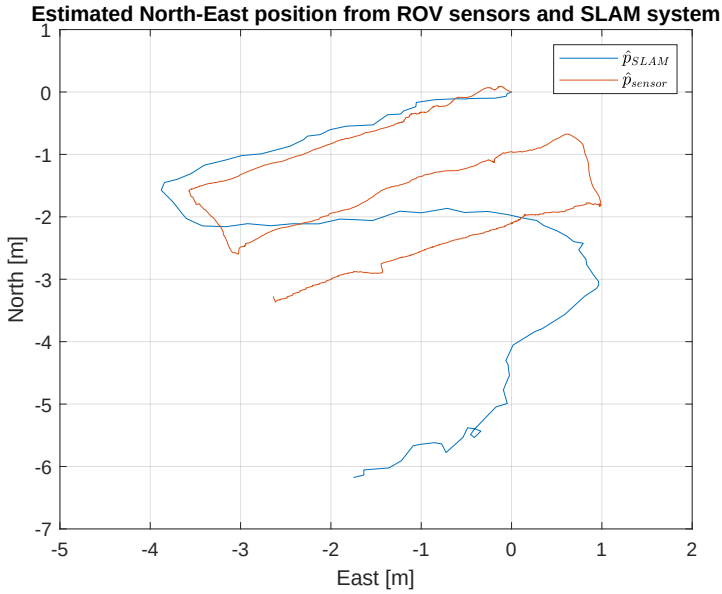


Figure 4.6: Simulation results scenario 2: North-East plot of estimated trajectory of the ROV from SLAM system and ROV sensors

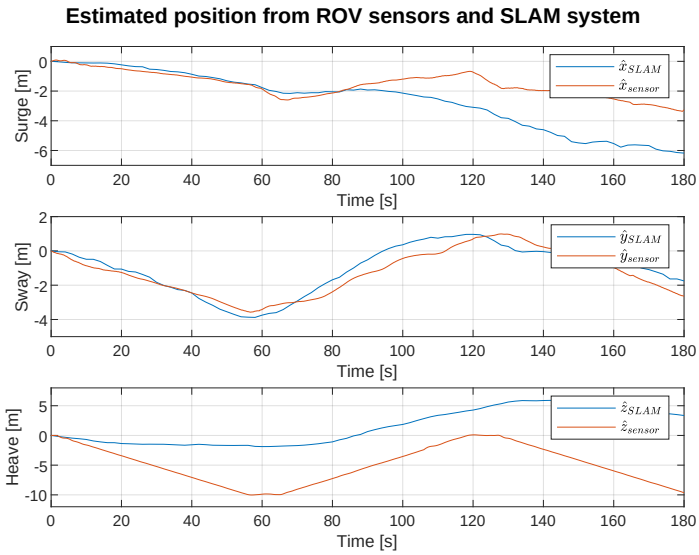


Figure 4.7: Simulation results scenario 2: plot of estimated surge, sway and heave from SLAM system and ROV sensors

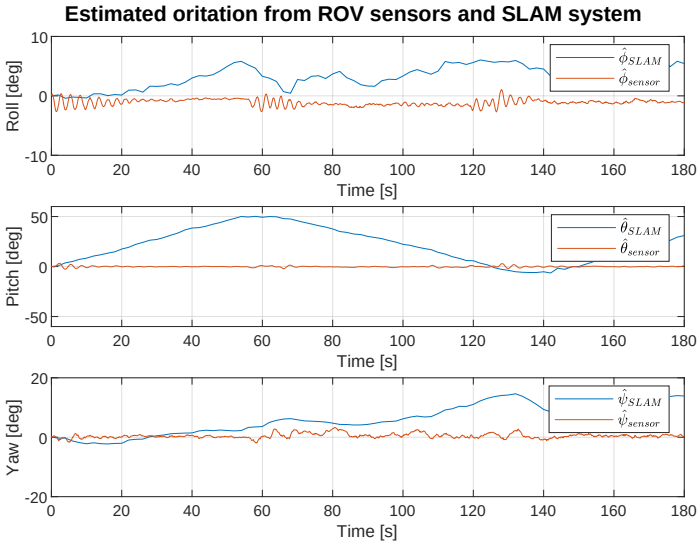


Figure 4.8: Simulation results scenario 2: plot of estimated surge, sway and heave from SLAM system and ROV sensors

The North-East plot of ROV trajectory estimation from this scenario is presented in Figure 4.6. The estimated change of ROV in position is presented in Figure 4.7, and estimated change in orientation in Figure 4.8.

The results shows that the SLAM system estimates North-East position of ROV well during the first 60 seconds. Then the estimated trajectory deviates from true trajectory. The reason is yaw angle estimation error increases after 60 seconds, which is the same as before. The wrong estimations of heave motion and pitch angle also exit in the simulation. A possible reason for this is discussed in Chapter 5.

4.2 Referansevraket data set

The third simulation is based on the data set from a mission at Referansevraket in August 2014. This mission used the same ROV and stereo camera as the mission at Stokkbergneset. Camera calibration results from Agisoft software are listed in Table 4.2.

	Left camera	Right camera
f	1698.41	1785.32
c_x	-9.40	-10.41
c_y	18.01	-12.21
k_1	0.2063	0.2252
k_2	0.6929	1.2036
k_3	2.8805	2.9083

Table 4.2: Camera intrinsic parameters and distortion coefficients estimation of Referansevraket data set

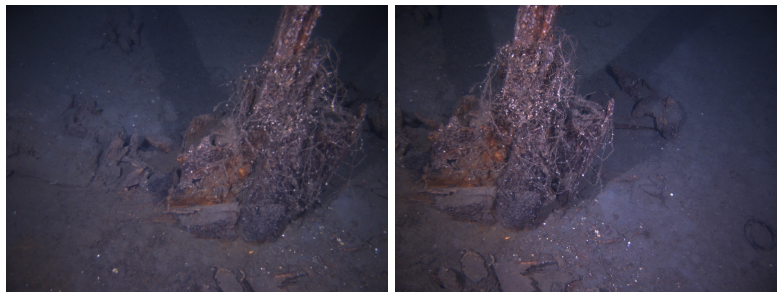


Figure 4.9: Left and right camera image pair example of Referansevraket data set

In this mission, the ROV moves around a broken artificial frame. An image pair example in this image set is shown in Figure 4.9. The light conditions in this scenario are good so that images are clear. But the images from left camera are blurrier than the images from right camera.

60 paired images are used in the third scenario, and the time interval of these images is 2 seconds. But the SLAM system stops after processing 19 pairs images because the system fails to get enough matches between image features and map points, and estimated position is untrustworthy. Then front end status changes to LOST and the program exits. The reason is that too few map points are computed from left and right cameras images. The clarity of two cameras images are different, so the difference of feature descriptors are too big. Most matches are not correct and are removed in stereo matching step.

The North-East trajectory, position change and orientation change of the ROV are presented in Figure 4.10, 4.11 and 4.12. The estimation results are good, and errors in position and orientation are not large.

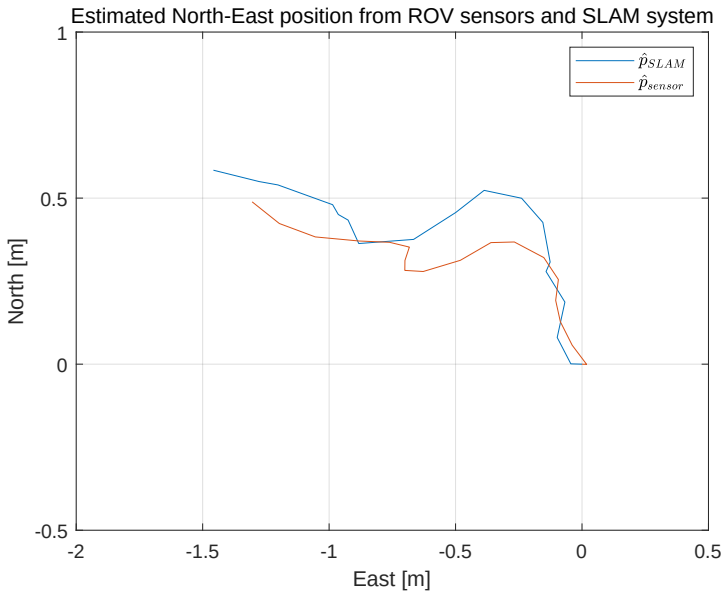


Figure 4.10: Simulation results scenario 3: North-East plot of estimated trajectory of the ROV from SLAM system and ROV sensors

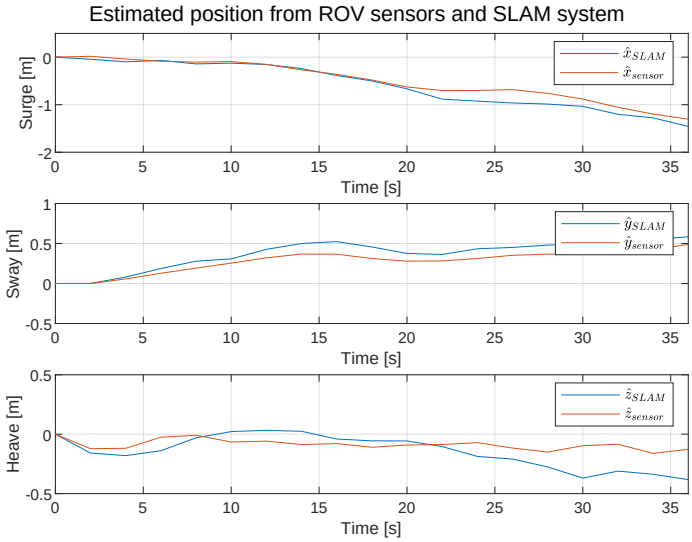


Figure 4.11: Simulation results scenario 3: plot of estimated surge, sway and heave from SLAM system and ROV sensors

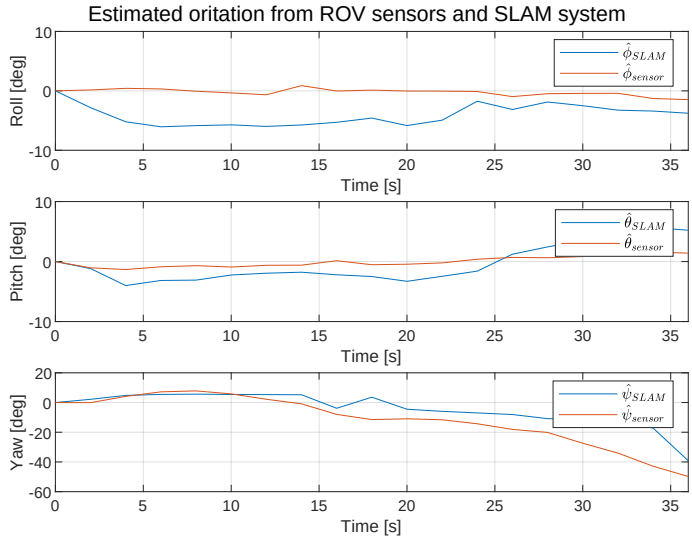


Figure 4.12: Simulation results scenario 3: plot of estimated surge, sway and heave from SLAM system and ROV sensors

4.3 TCP data transmission

TCP data transmission is implemented between two different computer. One computer runs C++ TCP client on Ubuntu 18.04 and another runs LabVIEW TCP server on Windows 10. The public IP address and port number are required to run TCP client. In simulation, the TCP client successfully establishes communication with the TCP server and receives initial position and orientation of the ROV from the server. The TCP server correctly receives estimated global position and orientation from the SLAM system. The reception delay in simulation is approximately 1 second.

Chapter 5

Discussion

5.1 Simulation

The results from three scenarios show that the SLAM system can estimates ROV position using stereo image set. The system gets good estimation results in surge and sway motion in all simulations. The errors between estimated positions and true positions are often less than 1m. But the estimations of heave motion are not accurate in the first two scenarios. The ROV moves up and down in these simulations, but the heave changes estimated by the SLAM system are much smaller. The heave changes are detected by the depth changes between camera and map points since cameras are placed downwards. To compensate these depth changes the system estimates large pitch angle changes instead of heave changes. Therefore, the estimated pitch angle orientation changes in the same pattern as true heave motion.

There are small yaw angle errors in first two scenarios, so the estimated trajectories deviates from true trajectories at the beginning of first simulation and after 60 seconds in second simulation. The errors increase with time, and the maximum value is around 15 degrees. Figure 4.6 shows that inaccurate yaw angle can significantly affect trajectory results.

In third scenario, estimation results in all positions and orientations are good. But the system only processes part of the image set and stops. As discussed in Section 4.2, few map points are found in each frame so that most image features cannot

find corresponding map points to estimate camera pose. The left camera images are blurrier than the right images, so the descriptor of keypoints in left and right camera images that are projected from the same world point may be different. Therefore, feature matching algorithm fails to find correct match between these keypoints.

The time interval of two frames are 2 seconds in both image sets. Often, the relative motion between two adjacent frames are large. So most frames in three simulations are set as keyframes. The process time of one keyframe is 0.1 to 0.3 seconds in first two scenarios and under 0.1 second in the third scenario. Therefore, the SLAM system can process stereo images and estimate position in real time with frequency of 3 to 10 Hz.

Due to the special situation, it is impossible to transmit estimated position data to the ROV control system in one computer or two computers in the same local area network (LAN). The data transmission is implemented via remote TCP connection. TCP communication is stable once connection is established. But the transfer delay of remote TCP is not negligible, and should be taken into account in real application.

5.2 General comments

Generally, a complete visual SLAM motion estimation system is huge and complex. It is challenging to find proper methods and algorithms to make the system accurate and efficient. A balance should be found between accuracy and efficiency because most algorithms with high accuracy often require high computation and long computation time, which are not suitable for real-time system. But motion estimation is sensitive to error data, so the methods used in the system are required to have sufficient accuracy and some filter must be used to remove wrong data. In this SLAM system, images are enhanced using the effective CLAHE algorithm to reveal more texture detail. ORB is proved to be a good feature detection and description algorithm for real-time application. In feature matching, there are lots of mismatches exist. Distance filter and RANSAC are two necessary steps to remove these wrong matches.

The process time highly depends on the number of detected keypoints and map scale. Keypoints detection and feature matching cost more than half the time in

simulations. Every time a keyframe is processed, tens and even hundreds of new map points are added into the map. So the scale of map increases with time. It is time-consuming to find map points that may be observed by the processed frame from all map points. So map points search is implemented in all active map points.

The performance of the SLAM system is affected by seabed landform and the quality of stereo camera images. Normally the system can detect sufficient features. But detection results may be bad on bare ocean floor. ORB algorithm can detect and describe features from blurry images, but the descriptors from blurry image and clear image that describe the same feature corner are different. So the number of correct matches is small. The movement pattern of the ROV also affects estimation results. If the ROV moves within a small area, some map points may be observed several times so that the results are good. If the ROV moves along a straight line or in a lawnmower pattern, like in first two simulations, cameras may only observe the map points from last frame. Other map points from older frames are not observed, and are removed from active map points set. When the ROV moves back to the same area again, these map points are not considered being observed because they are not active. The establishment of active map points set significantly reduces computation time, but most map points may not be fully used.

Chapter 6

Conclusion

6.1 Concluding remarks

This thesis has presented how visual SLAM techniques can be used in underwater environment to estimate the motion of ROV with a stereo camera. A real-time SLAM system is developed in C++ using OpenCV, g2o and other open source libraries and framework. The underwater images often have different contrast in different sections, which makes it hard to detect features. CLAHE method is used to improve image contrast. ORB algorithm is used for feature detector and descriptor, and features are matched using FLANN based matcher. The camera pose is estimated using EPnP method and optimized with graph based bundle adjustment. Back end graph optimization is activated when new keyframes and new map points are added in the map. A independent C++ TCP client is also included to transmit ROV motion to the ROV control system.

The system was tested through three different scenarios using two image sets. The results in Chapter 4 implies that the system can successfully estimate ROV motion. However, the rotation errors are large in some scenarios. Especially, yaw angle errors results in wrong heave estimations in first two scenarios. The process time of one frame is between 0.1 to 0.3 seconds, which is considered meets the real-time requirements.

The TCP client successfully received ROV position and orientation data from the

TCP server in LabVIEW and transmits estimated ROV motion to the ROV control system.

6.2 Further work

This thesis has implemented a ROV motion estimation real-time SLAM system using stereo camera. The system was tested by several image sets, and the simulation results are good. However, there are other methods have potential to improve the performance of the system.

CLAHE is a fast and universal method to enhance images from different environment. The method works well in most sea floor images. But the improvement is limited in poor feature area like bare seabed. There are other image enhancement algorithms are provided in OpenCV. Some methods specialized for underwater images can also be studied. A better method can extend the application scope of the SLAM system so that the system can be robust in different work environments.

Drift is hard to avoid during long estimation. The problem can be solved by adding loop closing and full bundle adjustment in the system. A general way to implement loop closing is establish a Bag-of-Words (BoW) to store all map points for detection. Once loop closing is detected, a full bundle adjustment including all keyframes camera poses is implemented to remove accumulative error.

Bibliography

[1] How much of the ocean have we explored. https://https://oceanservice.noaa.gov/facts/exploration.html. Accessed: June 5, 2020.

[2] Remotely operated vehicle. https://exploration.marinersmuseum.org/object/rov/. Accessed: June 5, 2020.

[3] Pioneer work class rovs (curv-i) – part 1. https://www.marinetechnologynews.com/blogs/pioneer-work-class-rovs-(curv-i-iii)-e28093-part-1-700495. Accessed: June 5, 2020.

[4] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167–193. Springer, 1990.

[5] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[6] Ryan M Eustice, Hanumant Singh, and John J Leonard. Exactly sparse delayed-state filters. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2417–2424. IEEE, 2005.

[7] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[8] Open source computer vision library. https://opencv.org/about/. Accessed: May 30, 2020.

[9] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011.

[10] Duane C Brown. Decentering distortion of lenses. *Photogrammetric Engineering and Remote Sensing*, 1966.

[11] Wikipedia contributors. Rgb color model. `https://https://en.wikipedia.org/wiki/RGB_color_model`. Accessed: May 15, 2020.

[12] Wikipedia contributors. Adaptive histogram equalization. `https://en.wikipedia.org/wiki/Adaptive_histogram_equalization`. Accessed: May 17, 2020.

[13] Shreenidhi Sudhakar. Histogram equalization. `https://towardsdatascience.com/histogram-equalization-5d1013626e64`. Accessed: May 17, 2020.

[14] Robert Hummel. Image enhancement by histogram transformation. Technical report, MARYLAND UNIV COLLEGE PARK COMPUTER SCIENCE CENTER, 1975.

[15] Stephen M Pizer, E Philip Amburn, John D Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 39(3):355–368, 1987.

[16] Deqing Sun, Stefan Roth, and Michael J Black. Secrets of optical flow estimation and their principles. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2432–2439. IEEE, 2010.

[17] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *2011 International conference on computer vision*, pages 2564–2571. Ieee, 2011.

[18] Edward Rosten, Reid Porter, and Tom Drummond. Faster and better: A machine learning approach to corner detection. *IEEE transactions on pattern analysis and machine intelligence*, 32(1):105–119, 2008.

[19] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.

[20] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[21] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o(n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[22] Timothy D. Barfoot. *State Estimation for Robotics*. Cambridge University Press, 2017.

[23] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[24] Fuzhen Zhang. *The Schur complement and its applications*, volume 4. Springer Science & Business Media, 2006.

[25] Wikipedia contributors. Huber loss. `https://en.wikipedia.org/wiki/Huber_loss`. Accessed: May 20, 2020.

[26] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 1, pages 666–673. Ieee, 1999.

[27] Shaharyar Ahmed Khan Tareen and Zahra Saleem. A comparative analysis of sift, surf, kaze, akaze, orb, and brisk. In *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*, pages 1–10. IEEE, 2018.

[28] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

[29] Konstantinos G Derpanis. Overview of the ransac algorithm. *Image Rochester NY*, 4(1):2–3, 2010.

[30] Xiang Gao, Tao Zhang, Yi Liu, and Qinrui Yan. *14 Lectures on Visual SLAM: From Theory to Practice*. Publishing House of Electronics Industry, 2017.

Appendix

ROV SF-30k

ROV SF-30k is a work class vehicle on NTNU's research vessel Gunnerus. The specifications of this ROV are listed in Table A. A picture of ROV SF-30k on the deck of Gunnerus is presented in Figure A.

Manufacturer	Spaerre AS
Model	SUB-fighter 30k
Frame	Anodized aluminum
Year	2004
Vehicle dimensions	LWH 2.6 $\times$ 1.5 $\times$ 1 m
Weight in air	1800 kg
Payload	60 kg
Max depth	1000 m
Power input	230 VAC, 3 phase, 40 kW
Thruster	Horizontal 2 $\times$ 3000 W Vertical 3 $\times$ 3000 W Lateral 1 $\times$ 3000 W
Max speed	2.1 knot in surge
Umbilical	diameter 27 mm length 600 m
Manipulators	Kraft Raptor mechanical manipulator
Cameras	2 $\times$ AVT GC1380 cameras
Lights	4 $\times$ 250 W halogen lamps 2 $\times$ 400 W HMI lamps
Scanning sonar	Kongsberg Mesotech MS1000

Table A: ROV SF-30k specifications

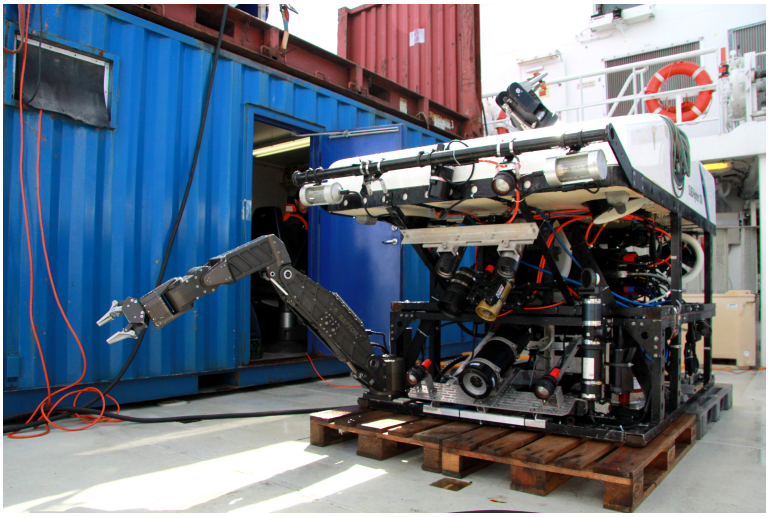


Figure A: ROV SF-30k

Camera

The image sets in two ROV missions used for simulation are recorded by two Allied Vision Prosilica model GC-1380 cameras. A picture of the camera is shown in Figure B. The specifications of these camera are listed in Table B.

Resolution	1360 × 1024
Sensor	2/3" CCD ExView HAD progressive scan Sony ICX285
Pixel size	6.45 × 6.45 µm
Max frame rate	20 fps
Lens mount	C-mount with adjustable back focus
Digital Interface	GigE Vision 1.0
Interface Type	IEEE 802.3 1000baseT
Power consumption	3.3 W (12V)
Max operating temperature	50 °C
Camera dimensions	LWH 59 × 46 × 33 mm
Weight	104 g

Table B: Camera AVT GC1380 specifications

Figure B: Camera AVT GC1380

Increased Autonomy and Situation Awareness for ROV Operations

Increased Autonomy and Situation Awareness for ROV Operations

Fan Gao
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway

Signe B. Moltu
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway

Erlend R. Vollan
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway

Shuyuan Shen
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway

Martin Ludvigsen
Department of Marine Technology
Norwegian University of Science and technology, NTNU
Trondheim, Norway
martin.ludvigsen@ntnu.no

***Abstract*—This paper proposes a semi-autonomous mission planning and management architecture for a Remotely Operated Vehicle (ROV). The work has focused on three main aspects to increase the autonomy of ROV inspections: developing a plan-based hybrid mission management architecture, integrating path planning as a refinement of actions in the agent architecture, and real-time communication between the mission management system and visual VSLAM (VSLAM) systems for obstacle detection and motion estimation. The architecture is inspired by the hybrid agent architecture [1] using a deliberative and a reactive layer to perform planned tasks and handle contingencies. A novel design of layered mission management architecture for ROV global navigation and local intervention is presented in this paper.**

***Index Terms*—autonomy, obstacle avoidance (OA), mission planning, visual-SLAM, path planning**

I. Introduction

Enabling autonomy in ROV operations will increase efficiency and save costs [2]. The general purpose of the work in this paper is to develop a mission planning and management system for underwater navigation and operation, and integrate vision-based situation awareness in ROV localization and obstacle detection, making the ROV more robust and capable for exploring the ocean. Under manual operation, human operators face risks of wrongly analyzing the situation, performing unnecessary or even fatal operations, increasing operation risks and costs. A plan-based mission management system can significantly avoid the above problems and simplify the execution procedure.

By placing a docking station on the seabed, long-term resident ROVs are feasible. With a docking station placed on the seabed, ROVs can charge their batteries and receive and transfer data to the operators onshore. Creating functionality for this is a big step in the direction of fully autonomous long-term missions.

Furthermore, in order to increase ROV autonomy, situation awareness and accurate local positioning are required. Existing acoustic positioning systems covers the global position of ROVs well, but tends to lack adequate local positioning precision. Stereo cameras are cheap sensors and can be used for ROV local navigation when paired with computer vision techniques. Visual simultaneous localization and mapping (VSLAM) is the problem of using visual inputs to concurrently construct a map of an unknown environment while estimating its location within it. VSLAM has been extensively researched on land-based vehicles proving good results. Developing subsea-based VSLAM systems could increase the local positioning accurately and simultaneously grant local situational awareness by providing a local map.

An overview of existing ROV standards considering underwater vehicle autonomy is presented in [2]. Four levels of autonomy were suggested in [3], being Manual Operation, Management by consent, Management by exception, and Fully autonomous. The level of situational awareness, decision making and control are increased with increasing levels of autonomy. This paper aims to increase the level of autonomy towards level three (Management by exception) for ROV subsea intervention. A new definition of symbolic actions for ROV mission planning and management is proposed, enabling automated planning to guide and integrate with mission management systems for task execution.

A. Related Work

This paper is an extension of the work done in [4]. The extension is concerned with implementing a governing mission

planner, deciding what the sequence of actions to carry out to reach the goal states. In addition, a path planner is added to ensure safe transit around known obstacles. VSLAM is further included to detect and warn about unknown obstacles.

A similar approach to enhance path planning was tested in [5]. An A* algorithm was proposed for the path planning of an autonomous underwater vehicle (AUV) in a partially-known environment with promising results. VSLAM was suggested for localization, where only a simple VSLAM method was used with a forward-facing sonar.

The benefits of choosing the A* algorithm as the path planning method for underwater vehicles were described in [6]. Some of them were that the algorithm has high maturity, is easy to implement and store, and has a low cost. The downsides, however, were that the algorithm has low efficiency and is not suitable for large scale space searches.

A large variety of SLAM solutions are available in the literature. The most classical approaches are filtering based approaches. They model the problem as an online position estimation, which is augmented and refined by incorporating the new measurements. Kalman filters [7], particle filters [8] and information filters [9] are some popular approaches in this category. Another way to solve the SLAM problem is the graph-based formulation. A graph is constructed whose vertices represent vehicle positions or landmarks and the edge between two vertices represents a sensor observation that constrains the vertices. Lu and Milios have proposed the technique in 1997 [10]. The sparse linear algebra makes it efficient to solve the optimization of the error minimization problem. With the improvement of computer computing power, graph-based SLAM has been mainstream in recent decades.

The outline of the paper is as follows: Section II presents the ROV mission management architecture, the mission planning methods, path planning for homing and docking, and vision-based motion estimation and obstacle detection strategy. The simulation results are presented and discussed in Section III. Section IV concludes on the performance of the proposed semi-autonomous mission planner and suggests modifications for further work.

II. Method

A. ROV Control System and HIL Simulator

The ROV control system used for simulations has been developed by the Applied Underwater Robotics Laboratory (AUR Lab) since 2010. The control system was firstly developed for DP and trajectory tracking [11]. As shown in Fig. 1, the control system is built on two basic modules: Frigg Graphical User Interface (GUI) and Njord Control system. The Frigg GUI enables high-level control and mode selection of missions, while the Njord control system performs low-level control, including a guidance system and a Nonlinear PID-controller. The proposed Autonomy Framework is added in Frigg GUI, providing autonomous mission execution command for lower-level control.

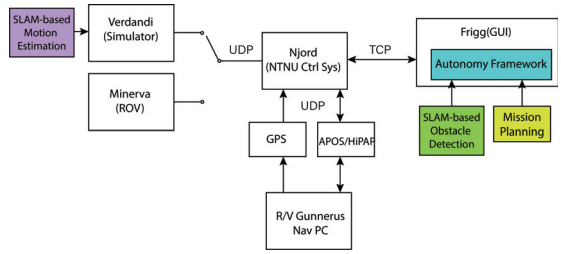


Fig. 1. Structural chart of the ROV control system

B. Mission Planning and Management Architecture

A design of hybrid mission planning and management system is proposed, consisting of a deliberative layer performing planned actions to achieve mission goals, a reactive layer responding fast to non-predictable events and a control execution layer acting as a coordinate mechanism that determines if actions from either the deliberative or the reactive layer should be executed by the lower-level controllers. Fig. 2 is a diagram of the hybrid mission planning and management system.

The deliberative layer performs autonomous behaviors based on known situations and events. The mission planner in this layer enables the ROV to plan tasks automatically and performs necessary re-planning. For the given user commands and known environment derived from sensors and a world model, the mission planner can recognize and categorize the tasks, generating a plan that fulfills the mission requests, and deliver a set of actions or commands to the mission controller. The mission will be re-planned automatically due to events such as failures in control, environment changes (such as the encounter with unexpected obstacles), and changes of mission target. In the deliberative layer, a layered design is implemented for global navigation and local operations as sub-missions. With actions of *Station Keeping*, *Launch*, *Descent*, *Transit* and *Operation*, global navigation enables the vehicle to approach a target location where local operation is to be performed. After the performed operation, the ROV is asked to go back to the predefined ending location. The sub-planner acts as the refinement of *Local Operation*. Possible actions for *Local Operation* are *Mapping*, *Sampling* and reactive *Charging*. A fast-forward search [12] method for mission planning is implemented to generate a near-optimal plan that fulfills the goals. An A* path planning algorithm for homing and general path planning is implemented in operation as refining of sampling and charging. A docking option is also implemented to simulate full homing and docking behavior.

The reactive layer responds to contingencies by analyzing sensor data, reasoning unexpected or unknown situations, modifying and interrupting missions. Exceedance of cable tension and obstacle avoidance are the two reactive behaviors implemented in the reactive layer. For actions that require path planning, such as sampling and charging, new detected obstacles are updated to the map, and re-planning is called

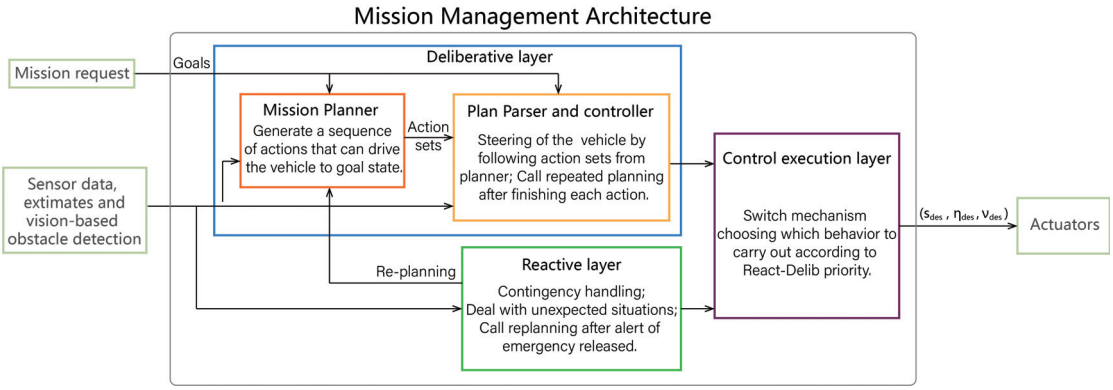


Fig. 2. Mission Planning and Management Architecture

to generate a new path that avoids both the known and the detected obstacles. For the other actions, a new waypoint is created based on the distance to the obstacle, the size of the obstacle and safety parameters for avoiding obstacles, as seen in Fig. 3.

A method of heading change is used to deal with this situation. When encountering a new obstacle, the direction of heading change is determined by β and ψ. β demonstrates the relative direction of the obstacle to the vehicle, and ψ is the heading angle of the vehicle. If the heading angle ψ is smaller than β, the new heading angle is $\psi_{new} = \beta - \alpha$. Otherwise, the new heading angle is $\psi_{new} = \beta + \alpha$. The angle α is calculated as $sin(\alpha) = \frac{R}{S}$, based on the diameter of the dangerous region and the distance between the vehicle and the obstacle. d is the diameter of the obstacle, and e is a safety parameter. Thus, $R = \frac{D}{2} + e$ is the radius of the dangerous region. L is the calculated length from the vehicle to the new waypoint. The position of the new waypoint is calculated based on the new heading angle and length L.

The global coordination function evaluates the output behaviors from the two architectures and will transfer inputs from the reactive architecture to the executor once it is activated. Since only one deliberative action is activated at a time in the deliberative layer and a coordinate mechanism is applied in the functional reactive layer to select the behavior with the highest priority among all activated behaviors under this layer. The control execution layer is thus organized as a selection of outputs between the deliberative layer and the reactive layer. It is responsible for deciding which layer and which action is activated and summing up the necessary parameters required by the control system.

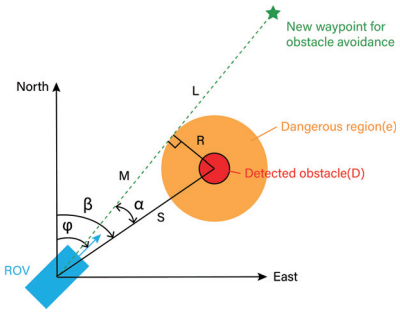


Fig. 3. Obstacle avoidance behavior

C. Integration of Path Planning

The A* algorithm, created by [13], was selected for path generation during missions that require path planning because it is easily adaptable, simplistic and performs well in known environments with low obstacle density [14]. Euclidean distance, presented in (1), was used for the heuristic function since it allows for movement in all eight neighboring tiles in a 2D grid space.

$$\sqrt{(x_{current} - x_{goal})^2 + (y_{current} - y_{goal})^2} \quad (1)$$

As the entire mission uses depth-control, a 2D path is justified. However, as there will be fluctuations and uncertainties in depth, known obstacles above and below two meters of the desired depth will be seen as relevant obstacles by the algorithm. This is also reasonable considering the specifications of the simulated ROV.

The path created by the A* algorithm can often include unnecessary turns for an underwater vehicle (UUV), which can move in any direction, not just in a straight path from the center of one tile to the center of the next. To make the path more smooth and avoid unnecessary turns, the smoothing method *Moving Average Filter* was used, implemented with the equation shown in (2).

$$ys(i) = (y(i-2) + y(i-1) + y(i) + y(i+1) + y(i+2))/5 \quad (2)$$

Because of the Constant Jerk Guidance [15] used in the ROV control system, a high number of waypoints would mean

a high number of stop's and go's. For this reason, collinear waypoints were firstly removed altogether from the generated path from the A* algorithm. However, as suggested in [16], the distance between waypoints should be seen in relation to the requirement of position computation. With longer path segments, the possibility of drift is higher. For this reason, a max waypoint distance is set. A comparison between the original path and the modified path can be seen in Fig. 4.

1) Docking Behavior: As the intended docking station for the mission is a platform docking station, a simple docking behavior is created. A pre-defined path, which can be seen in Fig. 5, is calculated from the size and orientation of the docking station. The path is intended to guide the UUV from the endpoint of the homing path to a waypoint with a fixed distance from the entrance of the dock, then to the next waypoint above the intended dock position, before descending down to land on it. The exact docking mechanism has not been addressed in this paper.

D. Integration of VSLAM-based motion estimation

The flow diagram of the VSLAM system is presented in Fig. 6. The VSLAM system for motion estimations is programmed with C++ using open-source libraries OpenCV and g2o. Images from underwater environments often have different contrast in different image parts due to poor lighting conditions. It is hard to detect features in the low contrast part, so image enhancement is necessary. Every time a new frame is sent to the system, contrast limited adaptive histogram equalization (CLAHE) [17] is implemented to enhance the contrast of stereo images. It works by mapping the histogram of the image to another histogram with a wider distribution of intensity values, so the intensity values are spread over the whole range.

ORB algorithm [18] is implemented to detect and describe features. Then feature matching is performed by using Hamming distance for ORB descriptors to find projection relationships between 2D pixel features and 3D map points from the VSLAM map. Generally, the matching results are not good enough to directly estimate the camera position because mismatching is hard to avoid. Distance filter and the RANSAC algorithm [19] are used together to remove mismatches. Next,

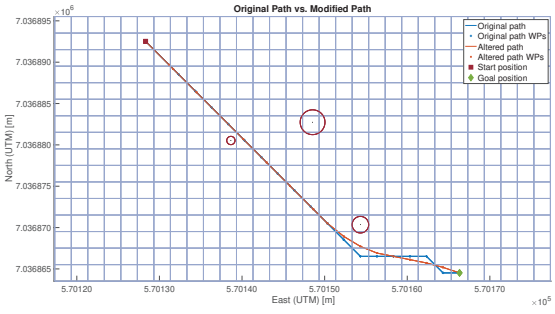


Fig. 4. Comparison between original path and improved path

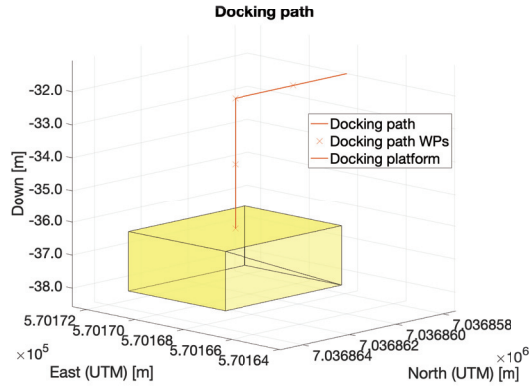


Fig. 5. Docking path

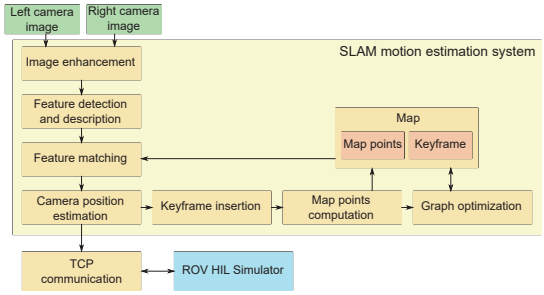


Fig. 6. Flow chart for VSLAM-based motion estimation

the camera position is estimated by using the EPnP algorithm [20] based on the matched 3D-2D points in feature matching.

The map of the VSLAM system stores all map points and keyframes. It is initialized when the first stereo image frame is processed. Then the map is updated when new keyframes are added. A keyframe is chosen if the number of matched features in the current frame is small, and new map points are required. Features on the left and the right camera images are matched to compute new map points. Graph optimization [21] is implemented to minimize the projection error between observed and predicted image pixel locations to find the best map point positions and keyframe camera positions.

The local positions of the camera are converted to the local positions of the ROV based on the camera coordinates in the ROV body frame. The ROV positions are transferred to the ROV mission planner using TCP communication. The system obtains the initial position of the ROV from the mission planner when it starts running so that the local position can be converted to a global position.

E. Integration of VSLAM-based Obstacle Detection

The real-time VSLAM-based Obstacle Detection revolves around utilizing the outputs of the renowned Visual VSLAM method ORB-SLAM2 [22]. The outputs of ORB-SLAM2 are the estimations of the ROV pose and point clouds of the surrounding environment of the ROV. From these two outputs, the closest detected obstacle is inferred. The obstacle detection system is implemented in C++ using the framework Robot Operating System (ROS). It is comprised of: a camera driver for running the stereo camera rig of the ROV Minerva, an image processing part using the open-source library OpenCV, an existing ROS implementation of ORB-SLAM2 [1], a point cloud processing part using the open-source library PCL and a communication part communicating with the ROV Autonomy Framework providing the closest detected obstacle. The communication is conducted using TCP connection. The system architecture is displayed in Fig. 7.

The stereo camera rig of Minerva consists of two Allied Vision Prosilica GC1380C mounted horizontally displaced. By considering the overlapping field of view and expected disparity values, the horizontal displacement, or baseline, is set to 0.2 meters. The cameras are configured in binned mode, reducing the resolution, but increasing the signal to noise ratio. The image processing undistorts and rectifies the stereo image pairs based upon obtained underwater calibration parameters, and additionally, contrast enhances the images using CLAHE. The point cloud processing first removes point cloud points associated with the seabed by fitting a plane using RANSAC; the remaining points are clustered using the Euclidean based clustering method of [23]. The obstacle sizes $\mathbf{o}_d$ and position $\mathbf{o}_p$ are inferred using

$$\begin{aligned} \mathbf{o}_d &= \mathbf{c}_{max} - \mathbf{c}_{min} \\ \mathbf{o}_p &= \frac{\mathbf{c}_{max} + \mathbf{c}_{min}}{2} \end{aligned} \tag{3}$$

where $\mathbf{c}_{max}$ and $\mathbf{c}_{min}$ are the maximum and minimum point position of the cluster. The closest detected obstacle is determined by finding the obstacle with the shortest Euclidean distance to the current estimated ROV pose. The LabView communication handles the TCP connection with the Autonomy Framework and generates messages containing information about the closest detected obstacle on the format in (4) if the distance threshold of five meters is violated.

$$\begin{bmatrix} x_o^g, & y_o^g, & z_o^g, & d_o \end{bmatrix} \tag{4}$$

The message is an array of doubles containing the obstacle position in the NED frame x_o^g, y_o^g and z_o^g, and the obstacle spherical diameter d_o. The spherical diameter is determined by selecting the largest estimated dimension of $\mathbf{o}_d$.

III. Simulation Results

A. VSLAM-based motion estimation

The VSLAM system performance is tested on a seabed image set from a mission at Stokkbergneset in February 2017 by NTNU ROV SF-30k. For the simulation, 60 paired left and right camera images with a time interval of 2 seconds are used. The result presented in Fig. 8 is simulated on this image set and the corresponding navigation data from the ROV control system.

The result shows that the VSLAM motion estimation successfully estimates the position of ROV in this simulation. The estimated orientation of ROV movement is slightly different from the true orientation, which causes errors in both north and east positions.

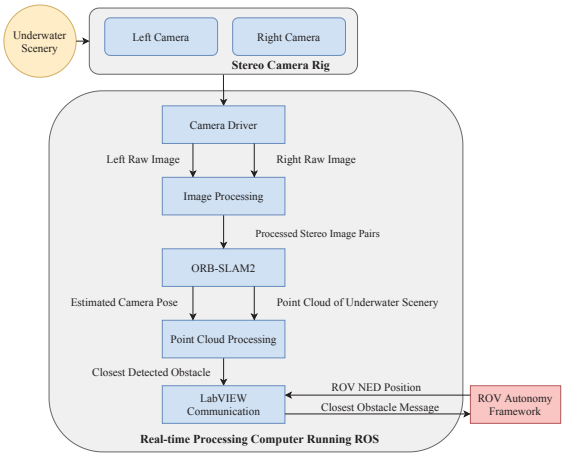


Fig. 7. System architecture of the vision based obstacle detection system

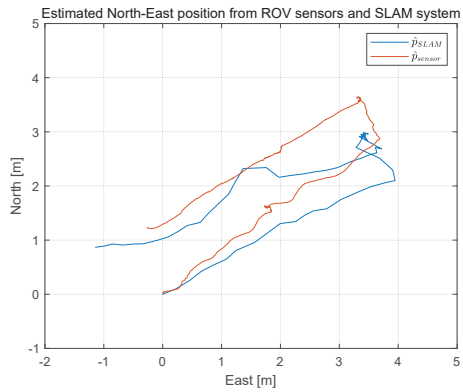


Fig. 8. Simulation of VSLAM-based motion estimation

[1] http://wiki.ros.org/orb_slam2_ros

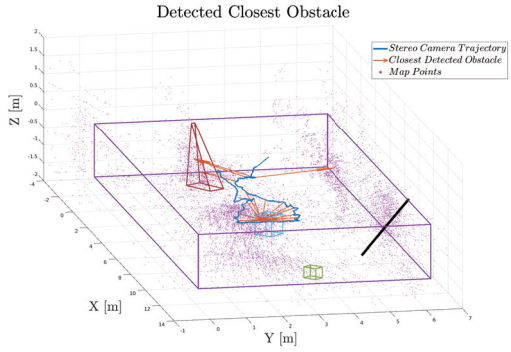


Fig. 9. Laboratory experiment of VSLAM-based obstacle detection.

TABLE I
OBSTACLES IN SIMULATION OF OBSTACLE AVOIDANCE

Obstacle	North	East	Depth	Diameter	[-]
01	7036914	570120	15	2	m
02	7036912	570114	15	1	m
03	7036922	570112	15	1	m

TABLE II
OBSTACLES IN SIMULATION OF MAPPING, CHARGING AND OA

Obstacle	North	East	Depth	Diameter	[-]
01	7036900	570144	15	2	m
02	7036886	570125	15	2	m

B. VSLAM-based obstacle avoidance

The VSLAM-based obstacle detection system is tested at the Marine Cybernetics Lab (MC-Lab) at Marin teknisk senter, NTNU. The stereo camera rig is mounted to a rod and moved through a measured underwater obstacle course installed in the basin of MC-Lab. The results are presented in Fig. 9 where the measured obstacle course is plotted together with the estimated trajectory, estimated point cloud and the currently estimated closest obstacle outputted at every 7.6 seconds. The obstacle course consisted of two boxes, a stepladder and rod.

Fig. 10 presents position estimates and desired positions of the vehicle in the North-East (NE) plane, giving satisfactory simulation results of reactive obstacle avoidance. A VSLAM-based obstacle detection code is programmed to test the distance between the vehicle's current position and obstacles. When the distance becomes shorter than five meters, the vehicle will perform reactive obstacle avoidance. Table III-B shows the location for three detected obstacles. Under some circumstances, more than one obstacle might be encountered at a time. The system chooses the nearest one as the current obstacle and executes the obstacle avoidance correspondingly.

The vehicle is performing the *Transit* action when the first obstacle is detected. The vehicle firstly moves to the desired position and then detects the *Obstacle 01*, performing obstacle avoidance correspondingly. Before reaching the waypoint for avoiding *Obstacle 01*, the vehicle detects a new *Obstacle 02* and generates a new waypoint to avoid it. The third obstacle is detected after reaching the waypoint for avoiding *Obstacle 02*. The test result shows that the obstacle avoidance algorithm successfully guides the vehicle to avoid collisions during mission execution. The mission planning and management system can encounter more than one obstacle at a time and update its waypoints based on the position of the closest obstacle. The mission planning and management system is also able to tackle obstacle avoidance when performing other actions. Reactive obstacle avoidance is also tested in Section III-C.

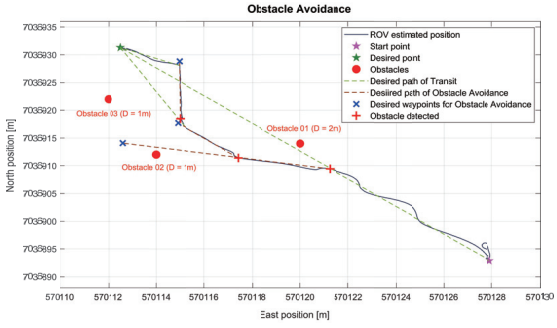


Fig. 10. Simulation of obstacle avoidance

C. Autonomous mission planning and management

1) Mapping, charging and OA: Obstacle Avoidance and *Charging* are tested during *Mapping* as reactive actions. The positions of the two manually set obstacles are listed in Table III-C1. The testing is carried out with hardware-in-the-loop (HIL) simulation. Fig. 11 shows that there are a lot of fluctuations when the vehicle changes its heading. The performance of waypoint tracking is mainly determined by the low-level control system.

In the mission, the vehicle starts at (7036892, 570128, 10) in UTM coordinates and then moves towards waypoints generated from the mission planner sequentially. The vehicle firstly performs *Launch* and *Descent* to the desired depth of operation. Then, the vehicle transits to the starting position of mapping and performs mapping, tracking six waypoints sequentially, found in Table III-C1. The desired trajectory of mapping is defined before starting the mission, while the path planner plans the waypoints of charging for homing once the *Charge* action is activated. During mapping, 'low battery' is manually set, indicating that the vehicle has to abort mission and move to the docking station to charge. The charging station is set at (7036882, 570140, 27).

TABLE III
WAYPOINTS OF MAPPING

Waypoint	North Position	East Position	Depth	[-]
01	7036912	570143	15	m
02	7036887	570143	15	m
03	7036887	570138	15	m
04	7036912	570138	15	m
05	7036912	570133	15	m
06	7036887	570133	15	m

TABLE IV
OBSTACLES IN FULL MISSION

Obstacle	North Position [m]	East Position [m]	Depth [m]	Diameter [m]
01	7036900	570144	15	2
02	7036907	570122	16	1
03	7036900	570112	15	1
04	7036893	570105	15	2

2) Full mission with OA: A similar mission as above, now with all four global states and three local states executed, was simulated. The resulting plots can be seen in 2D in Fig. 12 and in 3D in Fig. 13. Four "unknown" obstacles were detected by the VSLAM obstacle avoidance. The positions of theses obstacles are presented in Table III-C2. The action sequence of the full mission is *Launch-Descent-Transit-Mapping (OA and Charging)-Sampling (OA)-Transit (OA)-Descent*.

This simulation includes testing of obstacle avoidance for

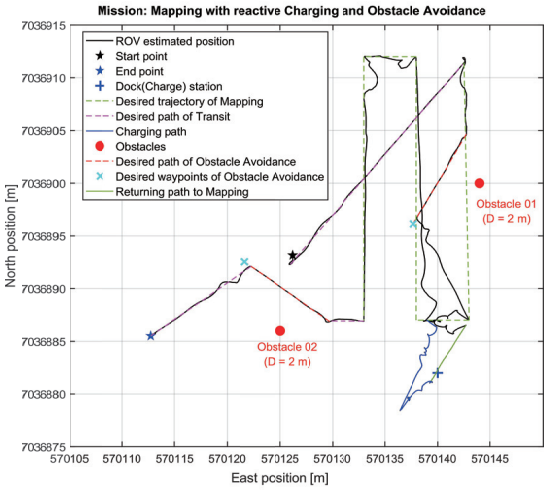


Fig. 11. Simulation of mapping, charging and OA

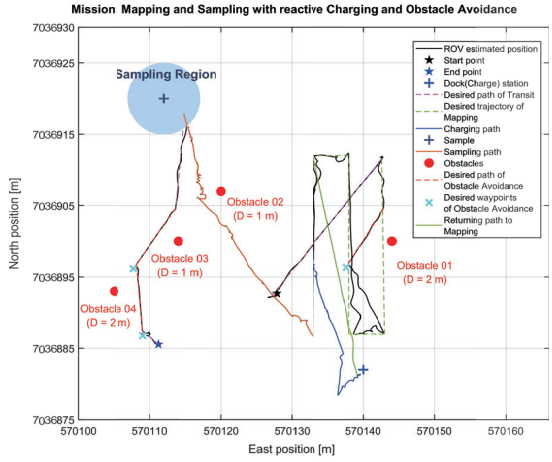


Fig. 12. 2D simulation of mission employing all states

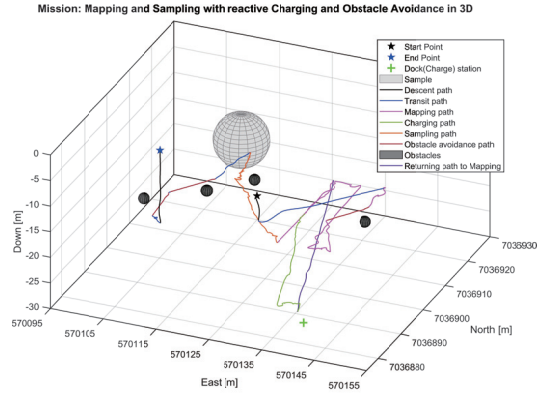


Fig. 13. 3D simulation of mission employing all states

actions *Transit*, *Mapping* and *Sampling*. For *Transit* and *Mapping*, the reactive behavior is turning its heading and move to a temporary waypoint for obstacle avoidance. During *Sampling*, obstacle avoidance is implemented by adding the newly detected obstacle in the obstacle list and replan to generate a new path with the A* algorithm which avoids the new obstacle.

IV. CONCLUSION

We have successfully implemented the autonomous mission planning and management system in the ROV control system, integrating vision-based situation awareness for motion estimation and obstacle detection. The system uses mission planning, guiding the vehicle to achieve the mission requests, and path planning is applied for local operations.

The use of the A* algorithm in an autonomous mission planning and management proved to be a good choice, as

it was convenient to implement it in the already existing framework. Adapting it to react to newly detected obstacles was also done in a successful manner, creating a robust path planner for the intended short range missions.

The VSLAM-based obstacle detection system performs well being capable of providing the currently closest obstacle in its locally estimated surroundings. However, as the laboratory experiments results imply, the positional consistency of the newly and previously detected obstacle degrade over the running time due to accumulated drift.

The VSLAM-based motion estimation presents good simulation results, using images derived from previous sea-trials. The testing result is valid in short periods. The deviation increases with the testing process. However, real-time VSLAM-based motion estimation has not be tested yet.

Since conducting the joint missions for this paper, both the docking behavior and A* path planning behaviors have been altered. A safety distance around obstacles have been added in the A* algorithm, and an ultra-short-baseline transducer has been simulated to situate at the dock to ensure more reliable position measurements when docking.

A. Further work

To verify the capability and performance of the autonomous mission planning and management system, sea-trails should be conducted. During simulations, sensor noise and environmental forces (such as current and waves) are not simulated. Also, the simulations use depth control during the execution of the mission. For further improvement, a combination of depth and altitude control should be designed such that the ROV could execute actions more accurately also relying on measurements from a doppler velocity log.

Up to now, the mission planning and management system is only capable of performing observations. Sampling and docking behaviors are simplified as trajectory tracking generated by path planning. More refinement should be designed for the actual realization of these actions.

The path planning algorithm is developed in a 2D plane while the ROV moves in a 3D underwater environment. The path planning in 3D can be further optimised including a full 3D model to the path generation procedure.

The VSLAM system for motion estimation is tested in a short time simulation. Loop closure techniques are expected to be added in the VSLAM system to reduce estimation error for long time position estimation.

REFERENCES

[1] I. Rist-Christensen, "Autonomous robotic intervention using rov," Master's thesis, NTNU, 2016.

[2] J. Hegde, I. B. Utne, and I. Schjølberg, "Applicability of current remotely operated vehicle standards and guidelines to autonomous subsea imr operations," in *ASME 2015 34th International Conference on Ocean, Offshore and Arctic Engineering*. American Society of Mechanical Engineers Digital Collection, 2015.

[3] C. Operations, N. Board, D. Sciences, and N. Council, *Autonomous vehicles in support of naval operations*, 09 2005.

[4] T. O. Fossum, M. Ludvigsen, S. M. Nornes, I. Rist-Christensen, and L. Brusletto, "Autonomous robotic intervention using rov: An experimental approach," in *OCEANS 2016 MTS/IEEE Monterey*. IEEE, 2016, pp. 1–6.

[5] J.-H. Li, M.-J. Lee, S.-H. Park, and J.-G. Kim, "Real time path planning for a class of torpedo-type AUVs in unknown environment," in *2012 IEEE/OES Autonomous Underwater Vehicles (AUV)*, Sep. 2012, pp. 1–6, iSSN: 2377-6536.

[6] D. Li, P. Wang, and L. Du, "Path Planning Technologies for Autonomous Underwater Vehicles-A Review," *IEEE Access*, vol. 7, pp. 9745–9768, 2019, conference Name: IEEE Access.

[7] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles*. Springer, 1990, pp. 167–193.

[8] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," *Aaai/iaai*, vol. 593598, 2002.

[9] R. M. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*. IEEE, 2005, pp. 2417–2424.

[10] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, no. 4, pp. 333–349, 1997.

[11] A. Sørensen, F. Dukan, M. Ludvigsen, D. Fernandes, and M. Candeloro, *Development of dynamic positioning and tracking system for the ROV Minerva*, 01 2012, pp. 113–128.

[12] J. Hoffmann and B. Nebel, "The ff planning system: Fast plan generation through heuristic search," *Journal of Artificial Intelligence Research*, vol. 14, pp. 253–302, 2001.

[13] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, 1968.

[14] A. Koubâa, H. Bennaceur, I. Chaari, S. Trigui, A. Ammar, M.-F. Sriti, M. Alajlan, O. Cheikhrouhou, and Y. Javed, *Robot Path Planning and Cooperation*. Springer, 2018, vol. 772.

[15] F. Dukan, "ROV motion control systems," Ph.D. dissertation, Norwegian University of Science and Technology, Faculty of Engineering Science and Technology, Department of Marine Technology, Trondheim, 2014.

[16] J. Yuh, T. Ura, and G. A. Bekey, *Underwater Robots*. Springer Science & Business Media, Dec. 2012, google-Books-ID: YwfaBwAAQBAJ.

[17] S. M. Pizer, E. P. Amburn, J. D. Austin, R. Cromartie, A. Geselowitz, T. Greer, B. ter Haar Romeny, J. B. Zimmerman, and K. Zuiderveld, "Adaptive histogram equalization and its variations," *Computer vision, graphics, and image processing*, vol. 39, no. 3, pp. 355–368, 1987.

[18] 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, 2011, pp. 2564–2571.

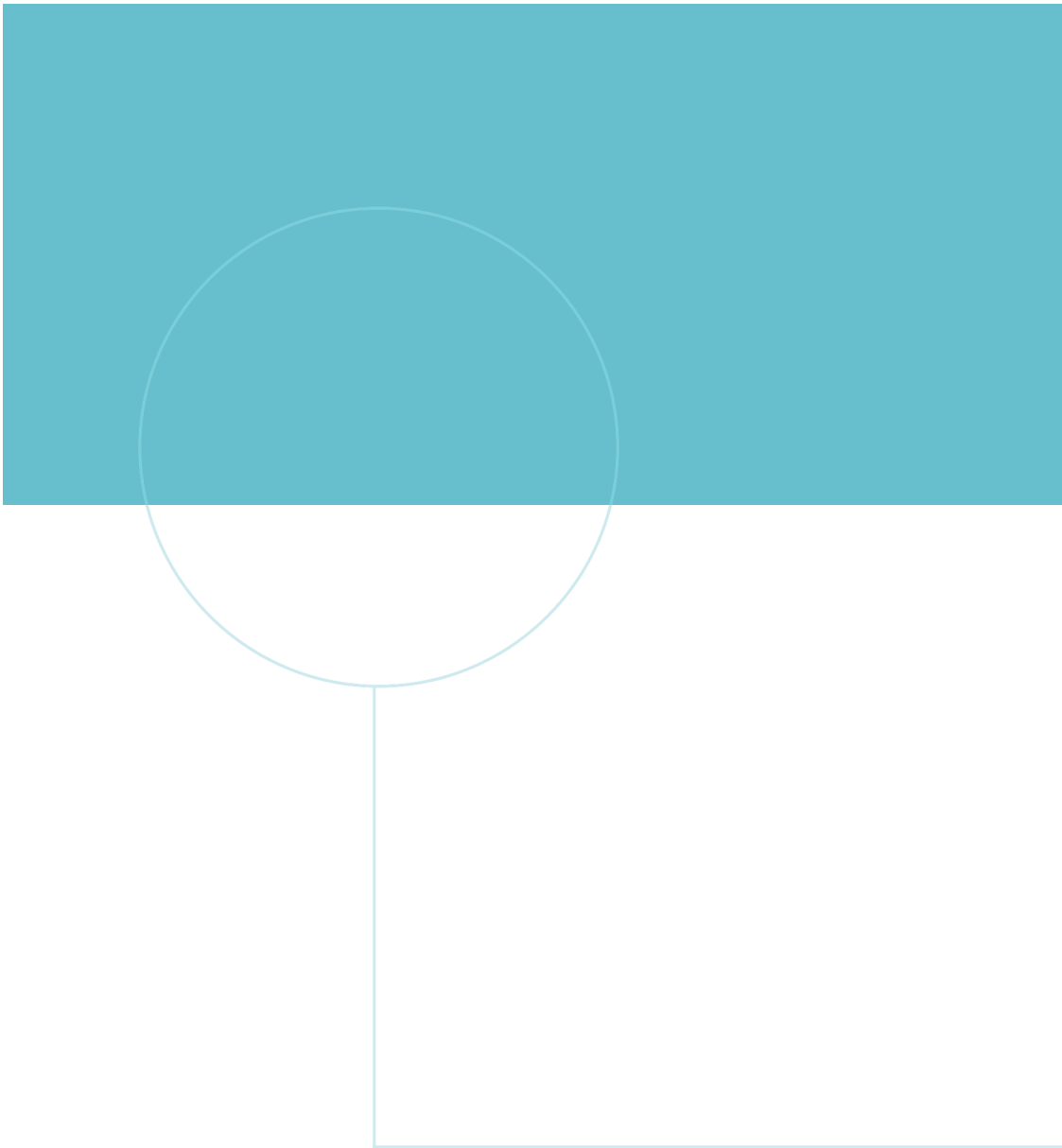
[19] K. G. Derpanis, "Overview of the ransac algorithm," *Image Rochester NY*, vol. 4, no. 1, pp. 2–3, 2010.

[20] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *International journal of computer vision*, vol. 81, no. 2, p. 155, 2009.

[21] R. Kümmerle, 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, 2011, pp. 3607–3613.

[22] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-d cameras," vol. 33, no. 5, pp. 1255–1262.

[23] R. B. Rusu, *Clustering and Segmentation*. Springer Berlin Heidelberg, vol. 85, pp. 75–85, series Title: Springer Tracts in Advanced Robotics.



NTNU

Norwegian University of
Science and Technology