Experiments with Visual Odometry for Hydrobatic Autonomous Underwater Vehicles

Somnath Balaji Suresh Kumar
Experiments with Visual Odometry for Hydrobatic Autonomous Underwater Vehicles

Somnath Balaji Suresh Kumar

Master's Programme, ICT Innovation, 120 Credits
(Specializing in Autonomous Systems)
Date: 2023-08-30

Supervisors: Sriharsha Bhat, Sowmya Swaminathan
Examiner: Mihhail Matskin
School of Electrical Engineering and Computer Science
Host company: SMaRC, Codot Studios
Swedish title: Experiment med visuell odometri för hydrobatiska autonoma undervattensfordon
Abstract

Hydrobatic Autonomous Underwater Vehicles (AUVs) are underactuated robots that can perform agile maneuvers in challenging underwater environments with high efficiency in speed and range. The challenge lies in localizing and navigating these AUVs particularly for performing manipulation tasks because common sensors such as GPS become very unreliable underwater due to their poor accuracy. To address this challenge, Visual Odometry (VO) is a viable technique that estimates the position and orientation of a robot by figuring out the movement of a camera and tracking the changes in the associated camera images taken by one or more cameras. VO is a promising solution for underwater localization as it provides information about egomotion utilizing the visual cues in a robot.

This research explores the applicability of VO algorithms on hydrobatic AUVs using a simulated underwater dataset obtained in Stonefish, an advanced open-source simulation tool specifically developed for marine robotics. This work focuses on the feasibility of employing two state-of-the-art feature-based VO frameworks, referred to as ORB-SLAM2 and VISO2 respectively since very little research is available for learning-based VO frameworks in underwater environments. The assessment is performed on a baseline underwater dataset captured by cameras of a hydrobatic AUV using the Stonefish simulator in a simulated algae farm, which is one of the target applications of hydrobatic AUVs. A novel software architecture has also been proposed for hydrobatic AUVs, which can be used for integrating VO with other components as a node stack to ensure robust localization. This study further suggests enhancements, including camera calibration and timestamp synchronization, as a future step to optimize VO accuracy and functionality.

ORB-SLAM2 performs well in the baseline scenario but is prone to slight drift when turbidity arises in the simulated underwater environment. VISO2 is recommended for such high turbidity scenarios but it fails to estimate the camera motion accurately due to advanced hardware synchronization issues that are prevalent in the dataset as it is highly sensitive to accurate camera calibration and synchronized time stamps. Despite these limitations, the results show immense potential of both ORB-SLAM2 and VISO2 as feature-based VO methods for future deployment in hydrobatic AUVs with ORB-SLAM2 being preferred for overall localization and mapping of hydrobatic AUVs in low turbidity environments that are less prone to drift and VISO2 preferred for high turbidity environments with highly accurate camera calibration and synchronization.

Keywords
Hydrobatic AUVs, VO, Stonefish, ORB-SLAM2, VISO2

Nyckelord

Hydrobatiska AUV:er, VO, Stonefish, ORB-SLAM2, VISO2
I began my university studies at KTH Royal Institute of Technology almost four years ago. After a wonderful time in Stockholm, I am now nearing the end of an important chapter of my life. I would like to express my gratitude to the eternal omnilight for reinstating the actual purpose of life and letting me pursue my dreams again. I would also like to dedicate this work to all those who have supported me in various ways and helped me grow professionally and personally over these years. I am forever indebted to each one of them.

First of all, I would like to sincerely thank my inspiration for the project Postdoc Sriharsha Bhat, who has been academically supervising me for initiating and finalizing the goals of the project, for instilling the confidence in me carrying out this research and for being extremely patient with me through his constructive feedback. I would also like to thank the whole SMaRC team for their open-source resources and helpful guidance initially in accessing the resources related to the project.

A very big thanks will go to my supervisor Sowmya Swaminathan, Chief Executive Officer of Codot Studios, who was interested in my research and taking it forward from my proposal that I pitched to her. Thanks for your trust in me and making me a part of the future research work in Codot Studios by providing your valuable feedback and sharing knowledge in assisting me with this research.

A big thank you will also go to my parents and brother for always being there for me over these years and allowing me to pursue my dreams, no matter what. Thank you for your unwavering trust in me.

Last but not the least, I would like to thank my examiner, Mihhail Matskin, for staying through the course of these years as an examiner and backing me up in any case. Thank you for your unwavering support in grading this work.

Stockholm, June 2023
Somnath Balaji Suresh Kumar
# Table of contents

Abstract........................................................................................................................................... i
Keywords ......................................................................................................................................... i
Sammanfattning ................................................................................................................................. ii
Nyckelord ....................................................................................................................................... ii
Acknowledgement ............................................................................................................................ iii
Table of contents .............................................................................................................................. iv
List of Figures .................................................................................................................................... vi
List of Tables ..................................................................................................................................... vii
List of acronyms and abbreviations ................................................................................................. viii
Chapter 1 .......................................................................................................................................... 1

## Introduction

1.1 Background ............................................................................................................................. 1
1.2 Problem ................................................................................................................................. 2
1.3 Purpose ................................................................................................................................... 3
1.4 Goals ...................................................................................................................................... 4
1.5 Research Methodology .......................................................................................................... 4
1.6 Delimitations .......................................................................................................................... 4
1.7 Structure of the thesis ............................................................................................................ 4

## Chapter 2

### Background

2.1 Visual Odometry in Underwater Scenarios ......................................................................... 7
  2.1.1 Traditional VO Approach .............................................................................................. 9
  2.1.2 Learning-based VO Approach ...................................................................................... 12
  2.1.3 Traditional VO vs Learning-based VO ........................................................................ 14
2.2 Hydrobatic AUVs .................................................................................................................. 15
  2.2.1 SAM AUV ..................................................................................................................... 16
2.3 Stonefish Simulator ............................................................................................................... 19
2.4 Summary ............................................................................................................................... 21

## Chapter 3

### Methods

3.1 Baseline Method: ORB-SLAM2 ......................................................................................... 22
3.2 LIBVISO2 ............................................................................................................................ 24
List of Figures

Figure 2.1: Example of a Monocular Visual Odometry System
Figure 2.2: AUV Performance Trade-offs
Figure 2.3: SAM AUV in real-time and simulation
Figure 2.4: SAM Hardware Components
Figure 2.5: SAM-Gripper Model
Figure 2.6: Simulated World in Stonefish with inbuilt control panel for checking sensor information, controlling turbidity and external factors
Figure 2.7: SAM AUV simulated in Stonefish, along with the SAM Web GUI
Figure 4.1: Software Architecture Stack for VO Integration
Figure 4.2: Experimental Pool Scenario created for SAM AUV
Figure 4.3: Simulated Pool Scenario in Stonefish
Figure 4.4: Left Camera Stream of SAM AUV from the Algae Farm Dataset
Figure 4.5: Simulated Algae Farm in Stonefish
Figure 5.1: ORB-SLAM Map Initiation
Figure 5.2: Tracked Features of Algae with slight drift and loss of tracking
Figure 5.3: Matched keyframes with ORB features and loop closures from the fully traversed map of ORB-SLAM2
Figure 5.4: Ground Truth – Odometry derived from SAM AUV sensors (yellow lines indicate the traversed path based on the sensor information in SAM AUV)
Figure 5.5: Monocular VISO2 – Covariance and Pose Estimation with Sparse Features (red line indicates the initial camera pose and green dots indicate the features)
Figure 5.6: Monocular VISO2 - Odometer Values (visualized in RQT) (x represents time, y represents the feature matches)
Figure 5.7: Ground Truth – SAM AUV inbuilt odometry information (visualized in RQT) (x presents time, y represents the features)
Figure 5.8: Warning Messages for Stereo VISO2 in Terminal

Figure 5.9: Disparity Image – Features Tracked from All Cameras

Figure 5.10: Right Camera View – Rectified Image Frame in Algae Farm Dataset

Figure 5.11: Left Camera View – Rectified Image Frame in Algae Farm Dataset

Figure 5.12: RQT Plot – Results of Stereo VISO2 (empty values indicate odometer not running properly for the received camera images)
List of Tables

Table 3.1: Requirements for Monocular VISO2
Table 3.2: Requirements for Stereo VISO2
Table 5.1: Numerical Evaluation Metrics considered for ORB-SLAM2
Table 5.2: Echoed Data /mono_odometer/info - Mono VISO2 on Algae Farm Dataset
Table 5.3: Echoed Data /stereo_odometer/info - Stereo VISO2 on Algae Farm Dataset
List of acronyms and abbreviations

AHRS  Attitude and Heading Reference System
AUV   Autonomous Underwater Vehicle
AUVAF Autonomous Underwater Vehicles for Algae Farming
BRIEF Binary Robust Independent Elementary Features
CNN   Convolutional Neural Network
CPS   Cyber-physical System
DGPS  Differential GPS
DOF   Degrees of Freedom
DSO   Direct Sparse Odometry
DVL   Doppler Velocity Log
EKF   Extended Kalman Filter
FAST  Features from Accelerated Segment Test
FLANN Fast Library for Approximate Nearest Neighbors
FREAK Fast Retina Keypoint
GPS   Global Positioning System
ICP   Iterative Closest Point
IMU   Inertial Measurement Unit
INS   Inertial Navigation System
KLT   Kanade-Lucas-Tomasi
LSD-SLAM Large-Scale Direct Simultaneous Localization and Mapping
LSTM  Long Short-Term Memory
MARVIE-A Maritime Autonomous Rovers for Visual Exploration - Archeology
MLESC Maximum Likelihood Estimation Sample Consensus
ORB   Oriented FAST and Rotated BRIEF
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ORB-SLAM</td>
<td>ORB-based Simultaneous Localization and Mapping</td>
</tr>
<tr>
<td>PnP</td>
<td>Perspective-n-Point</td>
</tr>
<tr>
<td>RSC</td>
<td>Random Sample Consensus</td>
</tr>
<tr>
<td>RGB-D</td>
<td>Red Green Blue-Depth</td>
</tr>
<tr>
<td>ResNet</td>
<td>Residual Network</td>
</tr>
<tr>
<td>RNN</td>
<td>Recurrent Neural Network</td>
</tr>
<tr>
<td>ROS</td>
<td>Robot Operating System</td>
</tr>
<tr>
<td>SAD</td>
<td>Sum of Absolute Differences</td>
</tr>
<tr>
<td>SAM</td>
<td>Small and Affordable Maritime Robot</td>
</tr>
<tr>
<td>SIFT</td>
<td>Scale-Invariant Feature Transform</td>
</tr>
<tr>
<td>SLAM</td>
<td>Simultaneous Localization and Mapping</td>
</tr>
<tr>
<td>SMaRC</td>
<td>Swedish Maritime Robotics Centre</td>
</tr>
<tr>
<td>SVO</td>
<td>Semi-direct Visual Odometry</td>
</tr>
<tr>
<td>SURF</td>
<td>Speeded Up Robust Features</td>
</tr>
<tr>
<td>UW-VO</td>
<td>Underwater Visual Odometry</td>
</tr>
<tr>
<td>VO</td>
<td>Visual Odometry</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

This chapter gives a brief description by introducing the problem addressed in this research project with a complete overview of the topics that the work will follow. Section 1.1 gives a general background about hydrobatics, Autonomous Underwater Vehicles and Visual Odometry in underwater scenarios. Section 1.2 explains the key problem that is addressed and states the research question considered. Section 1.3 and Section 1.4 highlight the purpose and goals of the research respectively followed by Section 1.5 to briefly discuss the research methodology adopted. Section 1.6 outlines the limitations of the project’s scope and finally, Section 1.7 outlines the structure of the rest of the project work.

This work investigates the viability of visual odometry frameworks particularly feature-based methods in hydrobatic Autonomous Underwater Vehicles (AUVs) and proposes an approach to assess its applicability for manipulating and navigating the AUV in challenging underwater environments.

1.1 Background

Hydrobatics is a concept that describes the agile movement of underwater vehicles, analogous to how aerobatics describes the agile movement of aerial vehicles. To be precise, hydrobatics is a term that refers to efficient and agile underwater robots [1]. Mobile robots with hydrobatic capabilities can perform complex tasks such as exploration, inspection, manipulation, and maintenance like conventional robots but with more maneuverability in difficult underwater environments. When infused with autonomy, the performance can increase multi-folds and overcome the challenges involved with regards to efficiency and compatibility.

Autonomous Mobile Robots (AMRs) are robots that have the capability to travel autonomously in any given environment without the need for human intervention or interaction. Autonomous Underwater Vehicles (AUVs) are a subdivision of AMRs that can operate independently underwater from crystal-clear shallow waters to the deep seas. It’s well-known that more than 80% of our Earth’s oceans are still unmapped, unobserved, and unexplored and currently, less than 10% of the global ocean is properly mapped [2]. With deep sea being the least observed part of Earth, AUVs serve a larger purpose than ever before. AUVs use a wide range of sensors and cameras to perform exploration and navigation tasks underwater. While performing these
tasks, it’s very crucial to know the position and orientation of AUVs without which there is a high possibility for losing track of the AUVs as it is becomes extremely difficult to track objects underwaters due to a plethora of factors.

A hydrobatic AUV can perform both hydrodynamic and acrobatic maneuvers, such as flips, rolls, and spins. Such maneuvers require high accuracy and robustness in the estimation of the AUV’s motion and pose, which are essential for controlling its position and orientation.

Visual Odometry (VO) is one of the key techniques used by AUVs to overcome this difficulty. VO is a technique that uses visual information from one camera or multiple cameras to estimate the pose and motion of a vehicle. When it comes to underwater environments, where GPS signals are very weak or literally unavailable, it becomes very crucial to accurately estimate the position and orientation of the vehicle. VO can provide reliable motion estimates for underwater robots by analysing the changes in the position of visual features captured by the camera [3]. However, there are some key challenges that VO faces when it comes to underwater scenarios such as low visibility, turbidity, water currents, hardware limitations and constantly changing lighting conditions.

1.2 Problem

VO faces many challenges in dynamic and complex environments, especially for aquatic applications. One of the main challenges is the presence of water currents, reflections, refractions, and turbidity that affect the image quality and feature extraction. Another challenge is the degradation of image quality due to light attenuation and scattering in underwater environments. Since VO estimates the motion and pose of a vehicle by analysing the images captured by a camera, the lack of distinctive and persistent features in underwater scenes makes it harder to establish reliable correspondences between consecutive images. Moreover, VO is prone to drift over time due to the accumulation of errors in the estimation process. This creates a need for a robust and accurate VO framework that can cope with these challenges and enable the possibility for manipulating hydrobatic AUVs in general.

In this thesis, we investigate the viability of VO for localizing hydrobatic AUVs using two state-of-the-art feature-based VO algorithms, ORB-SLAM2 and VISO2. Both these algorithms have been successfully applied to terrestrial and aerial robots. We conduct experiments based on these two methods on a simulated underwater dataset collected by a hydrobatic AUV called Small and Affordable Maritime (SAM) AUV equipped with monocular and stereo cameras. This study proposes some modifications in terms of hardware such as camera calibration and timestamp synchronization required in hydrobatic AUVs with SAM AUV as a reference for effective implementation of feature-based VOs in the future. The main research question is:
1. What are some potential feature-based VO algorithms suitable for underwater scenarios, particularly for hydrobatic AUVs?

2. In a simulated underwater setting, how feasible are ORB-SLAM2 and VISO2 as algorithms to perform VO in hydrobatic AUVs?

1.3 Purpose

The purpose of this research work is to find the most optimal approach for localization of hydrobatic AUVs in general and to investigate the possible use-case of algorithms to find the most feasible ones for performing VO and SLAM in hydrobatic AUVs. The target scenario is Even though most of the VO methods are usually targeted for ground autonomous vehicles, there is a high likelihood for the same methods to work in underwater scenarios and hydrobatic AUVs if the challenges mentioned previously such as turbidity, poor illumination, water currents et al are addressed properly. High precision is required as the dynamics and unpredictability of events are much worse in underwater environments due to a plethora of known and unknown factors involved as it is still constantly evolving and yet to be fully understood. The main purpose is to find how promising a VO framework can be so that it could be employed after continuous future iterations of research in various underwater environments.

This research was initially started as a research work in SMaRC and later became part of a future project work carried out by Codot Studios*, a cross-disciplinary software company also focused on autonomous solutions for maritime domain. The future project worked is focused on two key areas - namely Maritime Autonomous Rovers for Visual Exploration – Archaeology (MARVIE-A) and Autonomous Underwater Vehicles for Algae Farming (AUVAF). MARVIE-A is a project in progress focused on building highly maneuverable hydrobatic drones to explore the Indian Ocean to uncover hidden history by recovering archeological samples recovered underwater while AUVAF utilizes the same type of drones of efficient oceanic algae farming to sustainably harvest algae and seaweed in the ocean, contributing to responsible resource utilization. The overall purpose is focused on four disciplines – autonomy, endurance, perception, and communication – to develop next-generation maritime robots that can be useful oceanic production, safeguarding society, environmental sensing, and archaeological excavation in the ocean. If there is enough potential for a VO framework in the experimental testcases evaluated in this work, it can be extremely useful for the progress of the projects and disciplines mentioned above.

*SMaRC – Swedish Maritime Robotics Centre | Codot.io
1.4 Goals

The primary goal of this research is to pick two suitable state-of-the-art VO approaches for hydrobatic AUVs and check their feasibility by testing the algorithms in a simulated underwater dataset. In this regard, the chosen feature-based VO algorithms ORB-SLAM2 and VISO2 are evaluated to find out which one is more viable in the long run as a precursor for localization and manipulation of hydrobatic AUVs in challenging underwater environments. To bridge the knowledge gap that exists in the inability to integrating VO in AUV software components, the goal of this research is to also propose a prototype for a novel software architecture that can be used for the integration as a test-case scenario for running a single hydrobatic AUV test to quickly check the status of all software components involved.

1.5 Research Methodology

The research methodology adopted in this paper is based on the experimental approach, which aims to test the feasibility of two chosen feature-based VO frameworks called ORB-SLAM2 and VISO2 for targeted localization of hydrobatic AUVs for future use-cases. Both ORB-SLAM2 and VISO2 are implemented using Robot Operating System (ROS) Wrappers and are evaluated directly on an underwater dataset synthetically recorded in Stonefish simulator as a ROS bag. Ground truth values are used in comparison for evaluating both VO algorithms taken into consideration. The analysis and evaluation of the algorithms are based on both quantitative and qualitative metrics relevant for consideration in ROS environment. The estimated trajectories along with tracked features and pose estimation are visualized in ROS tools such as RViz and RQT for both algorithms.

1.6 Delimitations

There is a considerable amount of VO methods that are constantly under development with very limited open-source resources, the major limitation is conducting the experiments only on a chosen subset of VO algorithms, which are shortlisted based on the background research, purpose, target goals and availability of data. Due to this, the research has been carried out as an experimental approach. Analyzing more frameworks is not feasible as there is a scope creep for the project in terms of the target scenario (i.e) the focus is primarily on hydrobatic AUVs and algorithms which already have public access with support for ROS and Stonefish. The second crucial limitation is that the algorithms are tested exclusively on one dataset, which was manually recorded in a publicly accessible Stonefish environment. It is noteworthy to mention that it was literally impossible to find a dataset that was suited for the target scenario mentioned except for a few, which had more limitations than the one we manually recorded.
1.7 Structure of the thesis

Chapter 2 introduces the relevant background theory, focusing on underwater visual odometry, hydrobatic AUVs and Stonefish simulator. Chapter 3 discusses state-of-the-art VO methods present in the literature and demonstrates the detailed methodology of how they work along with their evaluation metrics. Chapter 4 describes the dataset used along with software architecture stack and experimental setup required for integrating VO and testing hydrobatic AUVs in a sample setup respectively. Chapter 5 presents the results of chosen VO methods. Finally, Chapter 6 draws the conclusions and opens new challenges for future work.
Chapter 2

Background

Due to the growing interest in AUVs, hydrobatic and interconnected AUVs, underwater navigation has become an active and challenging research topic. This is primarily because only a few of the land-based navigation methods used for land robots are applicable underwater. Depending on the purpose, underwater manipulation also becomes increasingly active and important so manipulation tasks can be performed underwater with good efficacy. Good underwater navigation is what paves way for effective underwater manipulation. Though manipulation can also be performed solely, without navigating properly to the required object or place, underwater manipulation serves only a smaller purpose. A robot uses its odometry to perform good navigation and effective solutions for odometry have been achieved using camera sensors. Such methods where we use a camera for odometry are referred to as Visual Odometry (VO) methods.

VO is the process of estimating the relative motion of a vehicle or a robot by analysing the images captured from the camera attached to it. Based on the type or whether the image sequences come from one camera or multiple cameras, VO can be classified into monocular, stereo, or omnidirectional VO. Monocular VO uses a single camera (as shown in Figure 2.1) and relies on the assumption of a known or constant scene depth or camera motion. Stereo VO uses two cameras with a known baseline and can estimate the depth from the disparity between the images. Omnidirectional VO uses a camera with a wide or panoramic field of view and can capture more information from the environment.

In underwater environments, VO becomes extremely challenging but valuable since ordinary sensors for on-road localization are usually not reliable in these hostile environments. For instance, GPS is applicable only when the AUV is on the water surface. In general, GPS can be accurate to within a few meters in open areas with a clear view of the sky. As the AUV moves beneath the water surface, the signals are attenuated by factors which create a lot of interference for GPS to be accurate and can lead to failure in tracking the AUV.
There are two critical aspects that can impact GPS accuracy underwater:

- The depth of the water: The greater the depth of the water, the lower the GPS accuracy. At deeper depths, GPS signals are reduced considerably by ocean currents, pressure and various confounding factors.

- The turbidity of the water: Turbid water (water containing suspended particles) can further weaken GPS signals, reducing GPS accuracy underwater even more.

The presence of metal objects and other e-waste in water can also block GPS signals, which can further reduce accuracy. Besides GPS, there are few other technologies that can be used to improve accuracy, such as Differential GPS (DGPS) and Inertial Navigation Systems (INS), but they are expensive when it comes to implementing them in scalable low-cost AUVs. Although wheel odometry is the easiest approach for position estimation, it suffers from position drift due to wheel slippage [10] hence VO methods that rely on visual information have the potential to provide accurate low-cost localization for AUVs and serve as a necessary precursor for most of the Simultaneous Localization and Mapping (SLAM) approaches.

### 2.1 Visual Odometry in Underwater Scenarios

One of the challenges in underwater robotics is to estimate the motion of a vehicle using only visual information and due to the inherent difficulties of underwater imaging, such as low contrast, blurring, refraction, and illumination changes, conventional VO methods designed for terrestrial or aerial applications may not work well in this domain. Therefore, several researchers have proposed custom solutions for underwater VO that exploit different feature extraction and matching techniques.
A common approach is to use the Scale-Invariant Feature Transform (SIFT) [11] to detect and describe salient key points in underwater images. SIFT is a popular feature descriptor that is invariant to scale, rotation, and affine transformations. Botelho et al. [12] used SIFT to compute the ego-motion of an underwater vehicle by finding correspondences between key points in consecutive images, rejecting outliers using Random Sample Consensus (RANSAC) Outlier Detection [13], and applying the homography matrix under the assumption of planar motion and a flat seabed. They compared their method with the Kanade-Lucas-Tomasi (KLT) [14] feature tracker, which tracks key points over time using optical flow. They found that SIFT was more robust than KLT to illumination and perspective changes, which are common in underwater scenarios. They also tested their VO system under various challenging conditions such as turbidity, snow marine, non-linear illumination, and noise. Burguera et al. [15] also adopted SIFT descriptors for an Extended Kalman Filter (EKF) SLAM framework with similar assumptions. In contrast to Botelho et al., they used a loop closure detection mechanism based on bag-of-words to correct the drift of the VO estimates. In [16], the authors relaxed the flat seabed assumption by using a depth gauge on the vehicle and estimating the motion only on the xy-plane. However, this method still required a constant depth during navigation.

Most of the existing methods for underwater VO rely on SIFT descriptors, which are known to be robust to various transformations. For example, Dabove et al. [17] proposed a monocular method that used SIFT features, RANSAC, and Kalman Filter to estimate the motion of an underwater vehicle. A similar approach was used in SSLAM [18], a stereo VO system that incorporated a loop chain matching scheme to improve the robustness of feature matching.

Ferrera et al. [19] developed UW-VO, a monocular keyframe-based method that was specifically designed for challenging underwater conditions, such as turbidity and temporary occlusions caused by marine life and algae. They used optical flow with KLT to track features between images, as they found that it performed better than standard descriptor-based methods, which often extracted ambiguous features from low-texture images. They compared their method with three state-of-the-art VO algorithms, ORB-SLAM [20], LSD-SLAM [21], and SVO [22], and showed that it achieved better results in 500 m deep underwater archaeological sites. UW-SLAM [23] extended UW-VO by adding loop closure detection.

Another approach for underwater VO is to use direct methods that minimize the photometric error between images instead of relying on feature extraction and matching. Direct methods can exploit all the pixels in the image and avoid losing information due to feature extraction. However, they are more sensitive to illumination changes and require subpixel accuracy. In [19], the authors proposed a direct VO method for underwater stereo images based on minimizing the Sum of Absolute Differences (SAD) between image patches. They used a rectification method that corrected both lens distortion and water-air refraction distortion. They also incorporated inertial measurements from an IMU to improve the accuracy and robustness of their method. They evaluated their method on both synthetic and real underwater datasets and showed that it outperformed SIFT-based methods.

The main difficulty for underwater visual odometry (VO) in deep waters is the high turbidity, while in shallow waters it is the scattering of sunlight. Moreover, the vehicle motion is less
stable in shallow waters due to the wave-induced oscillations. To cope with these challenges, Zhang et al. [24] developed a stereo VO system based on a modified version of ORB-SLAM2 [25]. Unlike ORB-SLAM, which relies on the previous motion model to reduce the search space for feature matching, they proposed a Quad matching method to handle the irregular motion of underwater navigation. They also used Speeded Up Robust Features (SURF), which performed better than SIFT, FAST [26], and ORB [27] in terms of average inlier numbers for underwater feature tracking. The VO system was evaluated in both shallow and deep waters. The results showed that the system could accurately reconstruct the entire trajectory in shallow waters, but it suffered from significant drift in deep waters due to the low quality of feature tracking caused by the high turbidity. They attempted to improve the image quality by applying three different enhancement techniques, but none of them improved the VO performance and instead introduced more noise to the images.

To enhance the camera motion estimation and reduce drift, many systems fuse VO with data from other sensors. For instance, Warren et al. [28] used a magnetometer to correct the yaw drift of a low-overlap stereo VO system. Weinder et al. [29] developed a method that initially combined a stereo camera and an IMU, and then extended it to include sonar data [30] and a depth sensor [31], resulting in a full SLAM system. In the case of monocular VO, data from other sensors are especially helpful to resolve the scale ambiguity problem. Crueze et al. [32] integrated a monocular camera with an IMU and a depth sensor. Xu et al. [33] proposed a system that employed two IMUs to measure acceleration and altitude, a sonar to estimate the distance to the seafloor, and a monocular VO module that is based on optical flow.

Deep learning techniques that learn feature representations or motion models from data have become popular in current VO research. In a variety of computer vision applications, such as object identification, segmentation, and recognition, deep learning techniques have demonstrated outstanding results. However, there are significant obstacles to using deep learning techniques to underwater voice over, including a lack of large-scale annotated datasets, a high processing cost, and the capacity to generalize across various conditions. In [24], the authors proposed a deep learning method for monocular underwater VO based on Convolutional Neural Networks (CNNs). They used CNN to extract features from underwater images and another CNN to regress the relative pose between two images. They trained their network on synthetic underwater images generated from a 3D simulator and fine-tuned it on real underwater images collected from different sites. They compared their method with SIFT-based methods and showed that it achieved better accuracy and robustness. In addition to these, other VO approaches that could be suitable underwater are also discussed in [61].

### 2.1.1 Traditional VO Approach

Besides the classification of VO based on camera setup, some VO methods are considered traditional approaches based on their focus on the quality and consistency of the image data, matching and extracting features across frames, etc. Feature-based methods and direct methods are usually considered as traditional VO methods.
There are two main approaches to VO: feature-based and direct methods. Feature-based methods extract and match salient features from the images, such as corners, edges, or blobs, and use them to estimate the camera motion and pose. Direct methods use the pixel intensity values directly from the images and minimize the photometric error between them. There are also hybrid methods that combine both feature-based and direct methods to exploit their advantages and overcome their limitations.

Feature-based methods are the most common and widely used approach to VO. They consist of four main steps: feature detection, feature description, feature matching, and motion estimation.

- Feature detection is the process of finding distinctive points in the images that can be reliably detected and tracked across multiple frames. Examples of feature detectors are Harris corner detector [34], FAST [35], SIFT [36], SURF [37], ORB [27] and AKAZE [38].

- Feature description is the process of computing a vector that represents the local appearance of each feature point. Examples of feature descriptors are SIFT, SURF, ORB, BRIEF [39], FREAK [40], and BRISK [41].

- Feature matching is the process of finding correspondence between feature points in different images based on their descriptors. Examples of feature matching methods are brute-force matching, FLANN [42], RANSAC [13], and MLESAC [43].

- Motion estimation is the process of computing the relative pose and motion of the camera from the matched feature points using geometric methods, such as perspective-n-point (PnP) [44] and essential matrix [45].

Feature-based methods have several advantages over direct methods. They are robust to illumination changes, occlusions, and dynamic objects, as they only use a sparse set of features that are invariant to these factors. They are also computationally efficient, as they reduce the dimensionality of the problem by discarding most of the image information and they can handle pure rotations, which are problematic for direct methods due to the lack of parallax. But the drawback is that they solely depend on the quality and quantity of the features detected and matched in the images. If there are not enough features or if they are poorly distributed, the motion estimation may be inaccurate or fail. Furthermore, they may lose information by discarding most of the image pixels that do not contain features. This may result in suboptimal solutions or drift over time. The following are some of the feature-based methods suitable for the underwater environments:

- ORB-SLAM2 [25]: A state-of-the-art method that uses ORB features for motion estimation and loop closure detection. It also incorporates direct methods for stereo initialization and pose optimization. It can handle large-scale and challenging environments with high accuracy and robustness. The upgraded third version is just an improved version with less camera configuration.
• **SVO [22]:** A fast and lightweight method that uses FAST features for image alignment and pixel intensity values for motion estimation. It can perform semi-direct monocular visual odometry at high frame rates on low-power devices.

• **VISO2 [46]:** A library and feature-driven method for stereo and monocular visual odometry that uses SURF features for feature matching and RANSAC for outlier rejection. It can handle urban and off-road scenarios with moderate accuracy and efficiency.

Direct methods are a relatively new approach to VO that emerged in recent years. They consist of two main steps: image alignment and motion estimation.

- **Image alignment** is the process of finding a transformation that minimizes the photometric error between two images. The photometric error is defined as the difference between the pixel intensity values in one image and their corresponding values in another image after applying the transformation.

- **Motion estimation** is the process of computing the relative pose and motion of the camera from the transformation parameters using nonlinear optimization methods, such as Gauss-Newton or Levenberg-Marquardt.

Direct methods have some advantages over feature-based methods. They use all or most of the image pixels for motion estimation, which makes it a bit more accurate and slightly less prone to drift. They also do not require feature extraction and matching, which simplifies the pipeline and reduces computational costs. They can also handle low-textured or repetitive scenes, which are challenging for feature-based methods due to the lack of distinctive features. But they are sensitive to illumination changes, occlusions, and dynamic objects, as they assume that the pixel intensity values are constant across different images. The cache is that they require an initial guess for the motion estimation, which may be difficult to obtain particularly in underwater scenarios and cannot handle pure rotations well, as they rely on parallax to estimate depth. The following are some of the potential direct methods that can be used for underwater drones:

- **LSD-SLAM [21]:** A novel method that uses pixel intensity values for image alignment and motion estimation, but only on a semi-dense set of pixels that have high gradient magnitude. It can also perform large-scale semi-dense mapping of the environment.

- **DSO [47]:** An extension of LSD-SLAM that uses pixel intensity values for image alignment and motion estimation, but only on a sparse set of pixels that are selected based on their gradient magnitude and variance. It can also perform full bundle adjustment on all active frames.

- **DVO [48]:** A simple method that uses pixel intensity values for image alignment and motion estimation, but also incorporates RGB-D data to estimate the depth from the
2.1.2 Learning-based VO Approach

Learning-based approaches are a new and promising approach to visual odometry, which is the act of calculating the motion and posture of a camera or a robot using pictures collected by the camera. Machine learning techniques, such as deep neural networks, are used in learning-based methods to learn the mapping between pictures and camera movements from data, rather than hand-crafted features or geometric models. There are two types of learning-based approaches: supervised and unsupervised methods. To train the network, supervised approaches require ground-truth labels for camera movements, such as postures or optical flow. Unsupervised approaches do not require ground-truth labels and instead train the network using self-supervision signals such as photometric consistency or depth consistency. Unsupervised methods do not require ground-truth labels, but instead use self-supervision signals, such as photometric consistency or depth consistency, to train the network. This literature review aims to survey some of the recent works on learning-based methods for visual odometry, focusing on their architectures, datasets, and performance.

The first type of learning-based approach taken into consideration are the supervised methods, which generally employ ground-truth labels as supervision to recover camera motion from a pair or sequence of photos using a deep neural network. End-to-end training of the network is possible using a loss function that evaluates the difference between anticipated and ground-truth motion. Here are some examples of supervised methods:

- DeepVO [49]: This is one of the first methods that uses a Convolutional Neural Network (CNN) to estimate the 6-DOF camera pose from a sequence of monocular images. The network consists of two parts: a feature extraction part that uses a FlowNet [50] architecture to extract features from each image, and a recurrent neural network (RNN) part that uses long short-term memory (LSTM) cells to model the temporal dependencies between features and output the pose increment. The network is trained end-to-end using a loss function that combines Euclidean distance and cosine similarity.

- VINet [51]: An interesting method that uses a CNN-LSTM network to estimate the 6-DOF camera pose from a sequence of monocular images, but also incorporates inertial measurements to improve the accuracy and robustness. The network consists of three parts: a CNN part that uses a ResNet [52] architecture to extract features from each image, an LSTM part that models the temporal dependencies between features and outputs the visual odometry, and an extended Kalman filter (EKF) part that fuses the visual odometry with the inertial measurements to output the final pose. The network is trained end-to-end using a loss function that measures the difference between the predicted and ground-truth pose.
• ESP-VO [53]: This uses an event-based camera, which is a type of camera that only records changes in pixel intensity, to estimate the 6-DOF camera pose from a sequence of event frames. The network consists of two parts: an event feature extraction part that uses an event-specific CNN to extract features from each event frame, and an RNN part that uses Gated Recurrent Units (GRUs) to model the temporal dependencies between features and output the pose increment. The network is trained end-to-end using a loss function that combines Euclidean distance and cosine similarity.

Supervised methods have several advantages over traditional methods for visual odometry. They can learn complex and nonlinear mappings between images and camera motion without relying on hand-crafted features or geometric models. They can also handle challenging scenarios, such as low-textured or dynamic scenes, where traditional methods may fail. Additionally, by modifying the input data or network architecture, they may be easily adaptable to other types of cameras or sensors. At the same time, huge volumes of labelled data are required for training, which may be difficult or expensive to gather, or may not cover all conceivable instances. They also have generalization concerns since they may not perform well on previously unknown data or in situations that differ from the training data, and they may lack interpretability because they do not present any intermediate outcomes or confidence levels for their predictions.

Unsupervised methods are the second type of learning-based methods for visual odometry. They use a deep neural network to estimate the camera motion from a pair or a sequence of images, without using ground-truth labels as supervision. Instead, they use self-supervision signals, such as photometric consistency or depth consistency, to train the network. Photometric consistency assumes that pixel intensity values are constant across different images after applying the camera motion. Depth consistency considers that depth values are inversely proportional to disparity values across different images after applying the camera motion. Some examples of unsupervised methods are:

• UnDeepVO [54]: A CNN-based algorithm that learns to estimate the camera pose from consecutive frames without any pre-requisites required of the involved scenario. First extracts features from the images and then uses the deep learning architecture of CNN to perform regression on the camera pose.

• SfMLearner [55]: Another CNN-based that learns to estimate the camera pose and depth from a sequence of images. It does this by first registering the images to a common reference frame using SfM techniques and then uses CNN to predict the camera pose and depth for each image in the sequence.

• GeoNet [56]: More of a semi-supervised algorithm that learns to estimate the camera pose from a sequence of camera images with sparse ground-truth poses. It performs this by first extracting features from the images and then also uses a CNN to regress the camera pose so can be considered unsupervised as well.
Unsupervised methods have several advantages over supervised methods for visual odometry. They do not require labelled data for training, which makes them more scalable and adaptable to different scenarios. They also provide intermediate results, such as depth maps or explainability masks, which can be useful for other tasks or applications. They can achieve comparable or even better performance than supervised methods on some benchmarks. The issue with unsupervised methods is that they rely on strong assumptions, such as constant brightness or small motion, which may not be practical in underwater scenarios. They also suffer from occlusions, dynamic objects, and illumination changes, which may violate the photometric or depth consistency. Hence, they may lack robustness, as they can produce noisy or inaccurate predictions due to the inherent ambiguity of the problem that exists underwater.

### 2.1.3 Traditional VO vs Learning-based VO

Traditional and learning-based approaches for underwater VO have significant advantages and limitations. Traditional approaches are more mature and dependable, but they rely on handcrafted characteristics and assumptions that may not be valid in underwater circumstances. Learning-based approaches are more versatile and flexible, but they require a big quantity of data and computer resources, and they may be difficult to comprehend and robust.

Among the traditional methods, feature-based methods are more common and widely used for underwater VO, as they can handle large displacements and illumination changes better than direct or semi-direct methods. It is obvious that feature-based methods may fail in texture-less or dynamic scenes, where feature extraction and matching become difficult. But the focus of most underwater missions is texture or scene-driven, so that can be negated by making sure to implement careful tuning of parameters and thresholds to achieve optimal performance.

Among the feature-based methods, VISO2 is a great option for underwater purposes as it is fast, accurate, and robust. A fast and cross-platform C++ library with MATLAB wrappers for computing the 6 DOF motion of a moving mono/stereo camera. It is based on minimizing the reprojection error of sparse feature matches, which are extracted by searching for blobs and corners in the current image. This makes it more robust to critical lighting conditions and low-texture seabed than other methods that rely on dense or semi-dense matching. Since in underwater environments, the visibility of light underwater can gradually decrease as the depth increases, VISO2 can be considered optimal. It also supports monocular egomotion estimation, which is still very experimental and uses the 8-point algorithm for fundamental matrix estimation and assumes a known and fixed camera height over ground for scale estimation and it does not depend on external libraries because it has a higher feature density (up to 15,000 feature matches), and it features a structure-from-motion pipeline for 3D reconstruction. These features make VISO2 a suitable choice for underwater scenarios, where the visual information is often sparse and noisy, and where 3D mapping is desirable. Its fundamental limitation is that it requires well corrected pictures and great precision in known calibration parameters, as well as being susceptible to drift over long distances. The reason why this happens is because VISO2 does not use inertial measurements and it can enhance the accuracy and durability of VO in
underwater conditions greatly. As a result, several recent studies have recommended combining LIBVISO2 coupled with an IMU sensor to improve performance [57]. This greatly indicates the capability of VIS02 for its performance in underwater scenario, particularly for hydrobatic AUVs as most of them do have an Inertial Measurement Unit (IMU) sensor embedded.

Among the learning-based methods, unsupervised or self-supervised methods are more suitable for underwater VO, as they do not require ground-truth labels that are hard to obtain in underwater scenarios. Unsupervised or self-supervised methods can learn from large amounts of unlabeled data and adapt to different environments. However, unsupervised, or self-supervised methods may suffer from scale ambiguity and occlusion issues, as they rely on image reconstruction or view synthesis as a proxy task. The research is very limited when it comes to applying learning-based methods to underwater VO, as most of the research in this field focuses on urban or natural scenes. One of the few works that addresses this problem is [58], which evaluates the performance and accuracy of different learning-based methods in the underwater context and compares them to classical methods. The work also proposes an extension of a learning-based method by using a visual-inertial sensor fusion network to correct the visual odometry estimate drift. The results show that learning-based methods can achieve better results than classical methods in some cases, especially in extreme lighting conditions where feature extraction and matching become impossible. But the research is very limited when it comes to applying them real-time or in scenario-driven underwater datasets and predominantly the results are based on assumptions and constantly evolving.

2.2 Hydrobatic AUVs

The field of hydrobatic AUVs is a relatively new area with very few published research. Hydrobatic AUVs are a type of AUV that are designed to be more maneuverable than traditional AUVs, both in terms of design and efficiency. They can hover, rotate, and change direction quickly, which makes them well-suited for tasks that require agility.

As shown in Figure 2.2, they are a new class of underwater robots that can perform agile and complex maneuvers in the water and can have high efficiency both in terms of speed and range, making them very useful for various applications, such as ocean production, algae farming, environmental sensing, and security. They can also enable new capabilities such as docking, inspection, or under-ice operations. However, designing and controlling hydrobatic AUVs is a challenging problem, as it involves nonlinear dynamics, fluid forces, and limited sensing and communication.

Hydrobatic AUVs are usually classified based on their mechanism of action:

1. Thrust vectoring AUVs: These AUVs use thrusters to control their direction and orientation. The thrusters are mounted on the AUV's body in such a way that they can
be used to create a force in any direction. This allows the AUV to hover, rotate, and change direction quickly.

2. Tilting AUVs: These AUVs use a tilting mechanism to change their orientation in the water. The tilting mechanism is typically mounted on the AUV's body, and it allows the AUV to rotate around its vertical axis. This allows the AUV to hover and change direction quickly.

![AUV Performance Trade-offs](image)

Figure 2.2: AUV Performance Trade-offs [3] (AUVs Courtesy: SAAB AB)

### 2.2.1 SAM AUV

One of the first examples of a hydrobatic AUV is SAM, which stands for Small and Affordable Maritime Robot, developed by Swedish Maritime Robotics Centre (SMaRC). SAM (shown in Figure 2.3) is a torpedo shaped AUV that weighs 15kg and measures 1.5m in length. It has six actuators: a thrust-vectoring mechanism for steering, counter-rotating propellers for propulsion, a buoyancy system for depth control, and trim control for adjusting the center of gravity in longitudinal and transverse axes [3]. SAM can perform hydrobatic maneuvers by using its unique actuator configuration. With regards to SAM, the focus is on robots that are characterized by small size (~20kg dry weight) so the handling becomes easy so only 1-2 persons are enough to operate it for testing. The goal is to develop a swarm of AUVs like SAM AUV to make highly cost-efficient AUVs. The SAM AUV has been created as a first step to initiate their experiments without the infrastructure needed for missions with larger craft.
SAM is capable of complicated maneuvers such as static hovering, tilting up to 90 degrees, and turbo turning with a 1.5 turning radius [7]. A Doppler Velocity Log (DVL) sensor for estimating velocity, an Inertial measurement unit (IMU) to calculate the AUV's angular rate and orientation, an Attitude and Heading Reference System (AHRS) to provide attitude information, three cameras, and other underwater sensors like a fuel level sensor, multibeam sonar sensor, pressure sensor, and others are all present in SAM's hardware, which is described in [7] and shown in Figure 2.4. All these electrical parts help SAM function properly and offer important details about the robot’s underwater information such as pose and status.

Figure 2.3: SAM AUV in real-time and simulation [7]
To further expand the actuator system and increase the attributes of the AUV, a manipulator from Blue Robotics will be integrated [59]. The manipulator has 1 degree of freedom, meaning it can only open and close in one direction but not rotate. It is driven by a linear actuator with a brushed motor and has a built-in controller/driver for PWM signal. The manipulator can be easily mounted on the AUV without heavily affecting its center of mass and maneuverability. The robot’s sensors provide feedback and publish to a ROS topic, that the manipulator can subscribe to, so that it will be actuated based on the cameras’ readings. The manipulator and AUV will then be controlled by the developed Linear Quadratic Regulator (LQR) controller and compared with PID and MPC as well, as shown in Figure 2.5 below.
### 2.3 Stonefish Simulator

Stonefish Simulator (shown in Figure 2.6) is a state-of-the-art open-source software tool developed for marine robotics research. It combines a physics engine and a lightweight rendering pipeline to deliver realistic simulation of underwater robots and environments. The physics engine is based on the core functionality of the Bullet Physics library, extended to include advanced hydrodynamics computation based on actual geometry of bodies, as well as simulation of underwater sensors and actuators [60]. The rendering pipeline, developed from the ground up, provides realistic rendering of atmosphere, ocean, and underwater environment, considering the effects of wavelength-dependent light absorption and scattering. Stonefish Simulator can be used for various purposes, such as testing control and vision algorithms in realistic underwater intervention tasks, designing, and prototyping new underwater robots, or creating virtual environments for training and education.

Stonefish Simulator consists of a C++ library and a Robot Operating System (ROS) package called stonefish_ros, which implements a standard simulator node that loads the simulation scenario from an XML file. The stonefish_ros package also delivers an interface for ROS interaction through messages and services, enabling convenient integration with ROS-oriented applications. Technically, it is a software package that extends the functionality of Stonefish Simulator and delivers an implementation of a standard simulator node, which can be launched from a launch file or a terminal command. Since the simulator node loads the description of the simulation scenario from an XML file, the physical and graphical properties of the simulated entities, such as robots, sensors, actuators, materials, looks, lights, and environment can be specified based on the requirements of the target maritime domain.

The XML file can also include ROS specific definitions, such as parameters, publishers, subscribers, and services. The simulator node parses the XML file and constructs the simulation scenario using the Stonefish library. The simulation automatically starts and runs at a specified frequency. With each simulation step, the simulator node publishes the readings of the selected sensors using appropriate ROS topics and updates the setpoints of the robots’ actuators based on incoming ROS messages. The sensor publishing and actuator updating are defined through a set of functions, constituting a Stonefish-ROS interface. Moreover, some functions of the Stonefish library are exposed through ROS services, such as resetting or pausing the simulation. Stonefish ROS enables convenient integration of Stonefish Simulator with other ROS packages and tools, such as Rviz, RQT and Gazebo.

Figure 2.6 shows a basic scene with a hydrobatic AUV and a small coffee cup simulated using the XML file in the Stonefish simulator and Figure 2.7 shows how objects in the Stonefish simulator can be integrated with other modules such as a control panel (which can be used for navigating and manipulating the AUV). In this case, the control panel, also called as the simulated real-time web GUI has been integrated with Stonefish simulator based on the open-source resources available for SAM AUV in Github*.

* [github.com/nilsbore/sam_stonefish_sim](https://github.com/nilsbore/sam_stonefish_sim)
  [github.com/nilsbore/sam_webgui](https://github.com/nilsbore/sam_webgui)
Figure 2.6: Simulated World in Stonefish (with inbuilt control panel for checking sensor information, controlling turbidity and external factors)

Figure 2.7: SAM AUV simulated in Stonefish, along with the SAM Web GUI
2.4 Summary

To summarize, underwater VO is a difficult topic for hydrobatic AUVs, particularly in terms of achieving agility, efficiency, and cost-optimization for future underwater missions. Several approaches based on diverse feature extraction and matching techniques, such as SIFT, KLT, direct methods, and deep learning methods, have been presented. As previously noted, each technique has advantages and disadvantages based on the use-case situation, sensor availability, hardware requirements, and other external considerations.

Among these methods, VISO2 stands out as a robust and efficient VO algorithm that can work with both monocular and stereo cameras. It uses a sparse feature-based approach that matches features across frames using a fast descriptor and a bucketing technique. It also employs a robust motion estimation method that minimizes the reprojection error using RANSAC and bundle adjustment. VISO2 has been previously tested for underwater scenarios [46] as it provides a ROS Wrapper that can potentially be synchronized with Stonefish simulator and seems the most straightforward approach to check the feasibility of VO for hydrobatic AUVs with SAM AUV as the testing ground. But the resources related to its implementation in underwater AUVs are predominantly assumptions. With regards to that, ORB-SLAM2 seems to be the next viable algorithm as it has been previously researched in underwater AUVs [25] and is suitable for stereo and RGB-D cameras. Hence it can be considered as the baseline method because the results seem more feasible for ORB-SLAM2 compared to VISO2 based on the literature review.

SAM AUV meets most of the requirements sufficient for testing hydrobatic capabilities in AUVs and has already been tested in real-time and simulation [3] [7]. Based on the background research done so far, in terms of the approach, only VISO2 and ORB-SLAM2 are compatible with the hardware capabilities of SAM AUV as they meet most of the requirements, including the requirement of an IMU. Hence, this work will focus on ORB-SLAM2 as the baseline method, as it is a combination of VO and Simultaneous Localization and Mapping (SLAM) approach together with more available resources and VISO2 as the experimental VO method considered for checking the feasibility as it is solely focused on VO – both monocular and stereo approaches.
Chapter 3

Methods

This chapter demonstrates the research methodology and methods considered in this research work. Section 3.1 focuses on the baseline chosen, a state-of-the-art feature-based SLAM approach. For the means of bridging the research gap between the chosen algorithms, Section 3.2 and Section 3.3 present a detailed explanation of LIBviso2, the core algorithm behind VISO2 and ROS Wrapper of LIBviso2 respectively. Section 3.4 and Section 3.5 present the monocular and stereo approaches of the chosen feature-based VO framework evaluated, describing them. Finally, the evaluation metrics are described in Section 3.6. The methodology adopted in this work is to evaluate the viability of VISO2 in algae farm dataset recorded using SAM AUV in Stonefish Simulator in comparison with results obtained by ORB-SLAM2 and ground truth on the same dataset, as VISO2 requires more focus on the underwater domain comparatively.

3.1 Baseline Method: ORB-SLAM2

ORB-SLAM2 is a real-time SLAM system for monocular, stereo, and RGB-D cameras, which can also be used for performing VO [25]. Interestingly it not only estimates the pose of the camera based on the images but can also build a map of the environment concurrently. It works based on ORB features, which we already discussed (i.e.) fast and robust keypoints that can be detected perform matching across subsequent images. As discussed, it works primarily based on two threads that are tracking and mapping along with the closure of loops, each of which communicates with each other through an inbuilt map database to store the points and keyframes [25].

ORB-SLAM2 uses a two-stage approach to SLAM [25]:

1. Tracking: In the tracking stage, ORB-SLAM2 tracks the current camera pose using the features that have been observed in the past.
2. Mapping: In the mapping stage, ORB-SLAM2 builds a map of the environment using the features that have been observed in the past.
The tracking stage is performed using a tracking-by-detection approach. This means that ORB-SLAM2 first detects features in the current image, and then matches these features to features that have been observed in the past. The matches are used to estimate the current camera pose. The tracking stage of ORB-SLAM2 is responsible for estimating the current camera pose. This is done by detecting features in the current image, matching these features to features that have been observed in the past, and then using the matches to estimate the camera pose. The feature detection step in ORB-SLAM2 is performed using the ORB feature detector.

The ORB feature detector is a fast and efficient feature detector that is based on the oriented FAST detector [25]. The ORB feature detector detects features by looking for corners in the image. The feature matching step in ORB-SLAM2 is performed using the Brute-Force matcher. The Brute-Force matcher is a simple but effective matcher that compares each feature in the current image to all of the features that have been observed in the past. The camera pose estimation step in ORB-SLAM2 is performed using the PnP solver. The PnP solver is a standard algorithm for estimating the pose of a camera from a set of point correspondences.

The mapping stage is performed using a back-end approach. This means that ORB-SLAM2 first builds a local map of the environment around the current camera pose. The local map is then used to build a global map of the environment. The mapping stage of ORB-SLAM2 is responsible for building a map of the environment. This is done by adding new features to the map, and by updating the poses of the features in the map. The new features are added to the map using the FAST feature detector. The FAST feature detector is a fast and efficient feature detector that is based on the difference of Gaussians. The poses of the features in the map are updated using the Iterative Closest Point (ICP) algorithm [35]. The ICP algorithm is a standard algorithm for aligning two point clouds.

ORB-SLAM2 also supports map reuse, which means that the system can load a previously saved map and use it for localization or SLAM. Map reuse can be useful for long-term operation or for transferring maps across different devices. ORB-SLAM2 also supports relocalization, which means that the system can recover from tracking failures by finding the best match between the current frame and the map.

The following pseudocode summarizes the requirements for using ORB-SLAM2 as a library:

```python
def ORB_SLAM2(current_image):
    # Detect features in the current image.
    features = detect_features(current_image)

    # Match the features in the current image to features that have been observed in the past.
    matches = match_features(features)

    # Estimate the camera pose using the matches.
    camera_pose = estimate_pose(matches)
```

```python
# Detect features in the current image.
features = detect_features(current_image)

# Match the features in the current image to features that have been observed in the past.
matches = match_features(features)

# Estimate the camera pose using the matches.
camera_pose = estimate_pose(matches)
```
# If the camera pose is not accurate, re-initialize the tracking.
if not is_accurate(camera_pose):
    reinitialize_tracking()

# Add new features to the map.
add_new_features(features)

# Update the poses of the features in the map.
update_feature_poses(features)

return camera_pose

## 3.2 LIBVISO2

LIBVISO2, also called VISO2, is a state-of-the-art library for performing monocular and stereo VO that relies on feature-based motion estimation and robust outlier rejection. It extracts features from the current image by searching for blobs and corners using a fast non-maximum suppression algorithm. It matches features across frames using a SAD metric and a left-right consistency check to ensure uniqueness and correctness of matches. It estimates the motion between frames by computing the essential matrix that encodes the rotation and translation using the five-point algorithm and a Gauss-Newton optimization [46]. It rejects outliers by using RANSAC to find the largest subset of matches that agree with the estimated motion. It decomposes the essential matrix into rotation and translation components using Singular Value Decomposition (SVD). It resolves the scale ambiguity by assuming constant velocity or constant acceleration models.

It performs the following steps for each pair of consecutive frames [46]:

- **Feature extraction:** It searches for blobs and corners in the current image using a fast non-maximum suppression algorithm. It computes a 16x16 pixel patch around each feature and normalizes it to zero mean and unit variance. It stores the features in a kd-tree for fast nearest neighbor search.

- **Feature matching:** It finds correspondence between features in consecutive frames using a SAD metric and a left-right consistency check. It searches for the nearest neighbor of each feature in the previous frame using the kd-tree and computes the SAD score. It repeats the same process for the reverse direction and checks if the matches are consistent. It discards matches that have a SAD score above a threshold or are not unique or consistent.

- **Motion estimation:** It computes the essential matrix that encodes the rotation and translation between frames using the five-point algorithm and a Gauss-Newton optimization. It uses the normalized eight-point algorithm to obtain an initial guess of
the essential matrix and then refines it using the five-point algorithm, which solves a system of polynomial equations using Groebner basis methods. It optimizes the essential matrix using a Gauss-Newton algorithm that minimizes the reprojection error between matched features.

- Outlier rejection: It uses RANSAC to discard matches that do not agree with the estimated motion. It randomly samples a minimal set of five matches and computes the essential matrix using the five-point algorithm. It counts the number of inliers that have a reprojection error below a threshold and repeats this process for a fixed number of iterations. It selects the essential matrix that has the largest number of inliers and recomputes it using all the inliers.

- Motion decomposition: It decomposes the essential matrix into rotation and translation components using SVD. It enforces the rank-two constraint on the essential matrix by setting its smallest singular value to zero. It obtains four possible solutions for the rotation and translation and selects the one that has the most points in front of both cameras. It resolves the scale ambiguity by assuming constant velocity or constant acceleration models and computing the scale factor that minimizes the change in velocity or acceleration.

### 3.3 VIS02 – ROS Wrapper

VIS02 ROS Wrapper*, aka viso2_ros*, is a ROS package that provides an interface for using LIBVIS02, a library for visual odometry, in an ROS-oriented environment. It implements two nodes: mono_odometer and stereo_odometer, which estimate the camera motion based on incoming rectified images from calibrated cameras. It publishes the odometry information as nav_msgs/Odometry and geometry_msgs/PoseStamped messages as well as through a transforms named tf, for easy visualization through RViz.

Since VIS02 ROS Wrapper is a software package that extends the functionality of LIBVIS02, it contains wrappers in ROS for both monocular and stereo visual odometry. It provides two nodes: mono_odometer and stereo_odometer, which perform the following steps for each pair of consecutive frames:

- Feature extraction: They use the feature extraction module of LIBVIS02 to search for blobs and corners in the current image using a fast non-maximum suppression algorithm. They compute a 16x16 pixel patch around each feature and normalize it to zero mean and unit variance. They store the features in a kd-tree for fast nearest neighbor search.

* [wiki.ros.org/viso2_ros?distro=indigo](http://wiki.ros.org/viso2_ros?distro=indigo)
• Feature matching: They use the feature matching module of LIBVISO2 to find correspondences between features in consecutive frames using the SAD metric and a left-right consistency check. They search for the nearest neighbor of each feature in the previous frame using the kd-tree and compute the SAD score. They repeat the same process for the reverse direction and check if the matches are consistent. They discard matches that have a SAD score above a threshold or are not unique or consistent.

• Motion estimation: They use the motion estimation module of LIBVISO2 to compute the essential matrix (for monocular) or the fundamental matrix (for stereo) that encodes the rotation and translation between frames using the five-point algorithm and a Gauss-Newton optimization. They use the normalized eight-point algorithm to obtain an initial guess of the matrix and then refine it using the five-point algorithm, which solves a system of polynomial equations using Groebner basis methods. They optimize the matrix using a Gauss-Newton algorithm that minimizes the reprojection error between matched features.

• Outlier rejection: They use the outlier rejection module of LIBVISO2 to discard matches that do not agree with the estimated motion. They randomly sample a minimal set of five matches and compute the essential or fundamental matrix using the five-point algorithm. They count the number of inliers that have a reprojection error below a threshold and repeat this process for a fixed number of iterations. They select the matrix that has the largest number of inliers and recomputes it using all the inliers.

• Motion decomposition: For monocular odometry, they decompose the essential matrix into rotation and translation components using singular value decomposition (SVD). They enforce the rank-two constraint on the essential matrix by setting its smallest singular value to zero. They obtain four possible solutions for the rotation and translation and select the one that has the most points in front of both cameras. They resolve the scale ambiguity by assuming constant velocity or constant acceleration models and computing the scale factor that minimizes the change in velocity or acceleration. For stereo odometry, they directly obtain the rotation and translation from the fundamental matrix using SVD. They do not need to resolve scale ambiguity since they have stereo information.

• Odometry publishing: They publish the odometry information as nav_msgs/Odometry and geometry_msgs/PoseStamped messages using appropriate ROS topics. They also publish a tf transform from odom frame to base_link frame, where odom is an arbitrary fixed world frame and base_link is an arbitrary fixed robot frame with REP 105 frame ids.

The following pseudocode summarizes the requirements for implementing VISO2:

```python
def VISO2(current_image, previous_image):
```
# Estimate the camera pose using the current and previous images.
camera_pose = estimate_pose(current_image, previous_image)

# If the camera pose is not accurate, re-initialize the tracking.
if not is_accurate(camera_pose):
    reinitialize_tracking()

# Add new features to the map.
add_new_features(current_image)

# Update the poses of the features in the map.
update_feature_poses(current_image, previous_image)

return camera_pose

## 3.4 Method 1: Monocular VISO2

Monocular VISO2 is a monocular visual odometry method that estimates the camera motion based on incoming rectified images from calibrated cameras. It runs in the background as a ROS node that subscribes to image topics and publishes odometry topics. It requires information about the camera’s z-coordinate and its pitch to estimate the scale of the motion. It also requires a tf_transform from the base_link frame to the camera frame to calculate the robot motion based on the camera motion.

Monocular VISO2 estimates the camera pose using a technique called PnP (Perspective-n-Point) pose estimation. PnP pose estimation works by finding the pose of a camera that best matches a set of known points in the scene. In the case of monocular VISO2, the known points are the features that have been detected in the current image.

In general, monocular odometry and SLAM systems cannot estimate motion or position on a metric scale. All estimates are relative to some unknown scaling factor. It overcomes this by assuming a fixed transformation from the ground plane to the camera. The required parameters are the height and pitch of the camera. To introduce these values, in each iteration the ground plane must be estimated. That is why features on the ground as well as features above the ground are mandatory for the mono odometer to work.

The steps are roughly as follows:

1. RANSAC and the 8-point technique are used to find the F matrix from point correspondences.
2. Utilizing the camera calibration, calculate the E matrix.
3. Calculate the scaled R|t and 3D points.
4. Calculate the ground plane in three dimensions.
5. Scale points and R|t using camera_height and camera_pitch.
The following table summarizes the crucial requirements for Mono VISO2 to work optimally:

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>Rectified images from a single properly calibrated camera</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Odometry information as ROS messages and tf transform</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>Camera’s z-coordinate, pitch, focal length, principal point</td>
</tr>
<tr>
<td><strong>Dependencies</strong></td>
<td>LIBVISO2 library, ROS framework</td>
</tr>
</tbody>
</table>

Table 3.1: Requirements for Monocular VISO2

### 3.5 Method 2: Stereo VISO2

Stereo VISO2 is a stereo visual odometry method that estimates the camera motion based on incoming rectified images from calibrated cameras. It runs in the background as a ROS node that subscribes to image topics and publishes odometry topics. It does not require any additional information to estimate the scale of the motion since it has stereo information. It also requires a tf transform from the base_link frame to the camera frame to calculate the robot motion based on the camera motion.

In a properly calibrated stereo system 3D points can be calculated from a single image pair. The linear system to calculate camera motion is therefore based on 3D-3D point correspondences. There are no limitations for the camera movement or the feature distribution.

Stereo VISO2 estimates the camera pose using a technique called Stereo PnP pose estimation. Stereo PnP pose estimation works by finding the pose of a camera that best matches a set of known points in the scene, using both the left and right images.

The following table summarizes the basic requirements for Stereo VISO2 to work properly:

<table>
<thead>
<tr>
<th>Requirements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input</strong></td>
<td>Rectified images from two or more properly calibrated cameras</td>
</tr>
<tr>
<td><strong>Output</strong></td>
<td>Odometry information as ROS messages and tf transform</td>
</tr>
<tr>
<td><strong>Parameters</strong></td>
<td>Focal length, principal point</td>
</tr>
<tr>
<td><strong>Dependencies</strong></td>
<td>LIBVISO2 library, ROS framework</td>
</tr>
</tbody>
</table>

Table 3.2: Requirements for Stereo VISO2
3.6 Evaluation Metrics

The accuracy of VO methods can be determined by comparing the estimated trajectory with the ground truth. Visualizing these trajectories is probably the most straightforward way to have a rough idea about the accuracy. Besides this, the quantitative aspect of evaluating the results, we consider covariance, number of tracked features, runtime, and drift in accordance with the ROS tools usually considered in ROS for evaluating the performance such RQT, RViz and ROS terminal. The specific metrics and factors will change constantly in tracking the performance of AUVs, particularly hydrobatic AUVs based on the target scenario. The metrics considered in this research are mostly relevant for feature-based VO approaches in a simulated underwater scenario and can subjectively change for other VO approaches based on the setting.
Chapter 4

Experiments

This chapter goes through the experiments conducted in this work that are necessary for evaluating the results in the future. The focus of this chapter is on bridging the research gap that exists in integrating VO with other software node modules required for full functioning of hydrobatic AUVs. Section 4.1 demonstrates the software architecture stack required for the integration of VO. Section 4.2 discusses the experimental scenario that can be replicated for testing VO algorithms in the form of a sample pool setup that is effective enough to run a sample AUV test. Section 4.3 explains about the simulated dataset synthetically recorded through Stonefish simulator using the cameras of SAM AUV available online.

This research has predominantly carried out all the experiments on the ROS as the coding environment using the Stonefish simulator. The languages of choice used for implementing the algorithms as a part of this project are Python and C++. ROS is a framework for writing software for robots and consists of an array of tools, software libraries, and conventions. This project heavily relies on the ROS framework. It includes, for example, the feed of camera images, inertial data, and radio information to the proposed system. Additionally, the use of ROS enables a flexible use of the developed loosely coupled modules that interface to VISO2 and Stonefish, as well as the used hardware. This is further described in the following sections.

4.1 Software Architecture Stack

Though the goal of this thesis is to test the viability of VO, a software architecture stack needs to be setup to bridge the research gap that exists in integrating VO. There is requirement for a right path to exist in the form of a blueprint for integration of VO into other node modules in the form of ROS nodes. Each ROS node can handle a different algorithm targeted to perform a specific function. If a VO works, it can’t function solely as a node module in a ROS environment. VO needs to be integrated with object detection and other localization nodes such as path planner for it to be used for navigation of hydrobatic AUVs. As VO always requires pre and post-processing, Figure 4.1 demonstrates a node stack in the form of a ROS Publisher-Subscriber architecture created using the sensor modules from SAM AUV to demonstrate a blueprint for integration of VO into other ROS nodes. As evident, it has separate node modules for perception, localization and path planning, which are three crucial pillars for running a single AUV test both in real-time and simulator.
Each ROS node mentioned in the stack indicates ROS topics (boxes) and nodes (ellipses). Each can have its own algorithm running in the nodes and get published as topics respectively so AUVs can constantly evolve managing computational efficiency.

The following are the three key nodes from the stack:

- `/Object_Detection_Node` – detects objects and performs computer vision
- `/Visual_Odom_Node` – determines object’s pose and performs VO/SLAM
- `/Path_Planner_Node` – for reaching the target and defining a path using a planning algorithm to find the most optimal route

### 4.2 Experimental Scenario

Based on the applicability of hydrobatic AUVs, the primary motive can be broken down into a basic scenario as a future use-case:

\[ \text{Begin Survey} \rightarrow \text{Detect/Approach Object} \rightarrow \text{Grab Object} \rightarrow \text{Proceed to Goal Point} \]
Using this basic use-case as a reference point, a pool scenario has been demonstrated which can serve as a prototypical model for full functional demonstration of an hydrobatic AUV.

![Diagram](image)

**Figure 4.2**: Experimental Pool Scenario created for SAM AUV (AUV Courtesy: SMARC)

The whole experiment needs to check most of the scenarios mentioned below for an optimal end-result in the context of a complete hydrobatic AUV test, so that all node modules are evaluated at the same time being computationally and economically less intensive than a large scale AUV test.


**Figure 4.2** shows an experimental representation of the pool scenario in Stonefish simulator along with software architecture stack modules required for the single hydrobatic AUV test:

- Given a goal point and trajectory, start from one end of the pool and scan through water.
- Modules perform their tasks, descend to the required object, grab it and reach the goal.
4.3 Datasets

Public access to datasets related to underwater environments is very limited in terms of open-source access. Adding to that, the datasets that are open source don’t support file format in the format of ROS bag files required for testing in the ROS environment. Since the chosen feature-based VO approaches require camera configuration information and at least decently calibrated cameras, the available underwater datasets also were neglected. The only exception was the Underwater Caves* dataset, which does provide ROS integration and a ROS bag file, but it is only recorded through a single camera, indicating that we need information about the camera’s height and pitch as it is a monocular camera setup, which is unfortunately not publicly accessible.

To ease out all these complexities, the primary dataset used in this work is a dataset recorded manually using the open-source files available from the cameras of SAM AUV in a simulated algae farm scenario based on Stonefish. Figure 4.5 shows the algae farm scenario accessed from Stonefish environments created for SAM AUV available in Github**. The dataset was recorded in Stonefish using SAM Web GUI and the configured SAM cameras, which are available for public access in Github***. The camera feed from all three cameras of SAM AUV is recorded as a single ROS bag file, a file format in ROS for storing ROS message data. This is just a dataset recorded in ROS environment. This dataset is termed as “Algae Farm Dataset” and will be available for public access once this research work is published.

The Algae Farm Dataset is a ROS Bag file that spans a duration of 163 seconds in actual time, but the frames can be speeded up or slowed down based on the requirements. This dataset,

* cirs.udg.edu/caves-dataset
** smarc_stonefish_sims/smarc_stonefish_worlds
*** smarc-project-native-gui/sam_webgui_native

Figure 4.3: Simulated Pool Scenario in Stonefish (KTH beverage can created using Blender)
Experiments comprising 483,093 messages, captures intricate dynamics and responses of SAM AUV in a simulated algae farm as a Stonefish environment. The frames were captured from three cameras of SAM AUV, where each camera will have a separate stream but combined into a single bag file. This dataset was specifically recorded in this manner so it can be used for both monocular and stereo approaches of VO. Figure 4.4 shows the left camera stream of SAM AUV as visible in the bagfile.

The dataset comprises messages from a variety of topics, such as encompassing critical sensor readings, camera information and control commands. The following list shows the key message topics, size, rate, and message type of the key topics of the Algae Farm Dataset:

- **Size**: 13.2 GB
- **/sam/core/imu**: 3263 messages - sensor_msgs/Imu
- **/sam/core/thrust_vector_cmd**: 2037 messages - sam_msgs/ThrusterAngles
- **/sam/dr/altitude**: 1631 messages - std_msgs/Float64
- **/sam/dr/depth**: 1631 messages - std_msgs/Float64
- **/sam/dr/odom**: 16,308 messages - nav_msgs/Odometry
- **/sam/sim/dvl**: 816 messages - cola2_msgs/DVL

These diverse data streams holistically characterize SAM AUV's state, motion, and interactions, made compatible to be visible in Stonefish to test varying conditions. Figure 2.6 shows the control panel of Stonefish simulator where the conditions and external factors can be controlled such as turbidity, suspended particles, etc.

![Figure 4.4: Left Camera Stream of SAM AUV from the Algae Farm Dataset](image-url)
Figure 4.5: Simulated Algae Farm in Stonefish
Chapter 5

Results

In this chapter we present and analyze the results obtained after performing our experiments using the ROS bag file based on the simulated algae farm scenario in the Stonefish simulator. We start by evaluating the results achieved by the baseline method, ORB-SLAM2 (Section 5.1). The experiments conducted on the feature-based method, Monocular VISO2 and the obtained experimental results are discussed in Section 5.2. Section 5.3 reports the experiments conducted on the same feature-based method but in the context of stereo camera setup and analyzes the results achieved. There are different evaluation metrics for each method considered and hence the ground truth is assumed wherever required in the context. The evaluation metrics are predominantly laid out in the form of graphs, maps, tables, and message topics that are considered as quantitative and qualitative metrics while testing an algorithm in the ROS environment.

5.1 Baseline Method: ORB-SLAM2

As a preliminary step, the performance of ORB-SLAM2 was evaluated on the Algae Farm Dataset we recorded. As discussed before, this dataset is an ROS bag file of a hydrobatic AUV moving through an algae farm. The dataset consists of several image sequences of algae with different turbidity conditions, simulating realistic underwater environments.

We found that ORB-SLAM2 was able to initialize and track most of the sequences except for some cases where the features were less, or the turbidity was too high. It also generated a sparse map of the environment and detected some loop closures when revisiting previously seen areas. It was observed that ORB-SLAM2 suffered from a slight drift over time, especially in featureless sequences. This was because of the change in Jerlov water type that was initiated while recording the bag file to vary conditions to see how it affects performance in the algorithms. In the case of ORB-SLAM2, it affected the quality and repeatability of the ORB features.

We can also notice that ORB-SLAM2 sometimes fails to recover from tracking losses or relocalize after loop closures, resulting in inconsistent maps. Figure 5.2 shows one such case where it suffered a slight drift and halted too early as it did not recover after a loop closure on the same dataset. Figure 5.3 shows the case where it performed proper localization and built a proper map of fully traversed path of SAM AUV in Stonefish simulator.
Figure 5.1: ORB-SLAM Map Initiation

Figure 5.2: Tracked Features of Algae with slight drift and loss of tracking
Figure 5.3: Matched keyframes with ORB features and loop closures from the fully traversed map of ORB-SLAM2

The colors from the map generated indicate the following:
- **Green** - Initial Keyframe (camera frame)
- **Purple** - Intermediate keyframes (frames that are tracked initially as AUV starts moving)
- **Blue** - Loop closures (keyframes detected when AUV comes back to a previous location)
- **Red** - Final keyframes (final set of keyframes detected)
- **Dots (in red color)** - Matched keyframes

With regards to the evaluation metrics, one key evaluation metric used to assess the performance of the ORB-SLAM2 method was the measure of drift over time. Drift was quantified by calculating the positional errors that accumulated as the algorithm tracked the AUV's movement through various sequences. The magnitude of drift was analyzed in both short and extended featureless sequences. The resulting drift trajectories were visualized to understand the extent of position deviations from the ground truth.
Another evaluation metric employed was the detection of loop closures in the generated environment map. Loop closures are indicative of the algorithm's ability to recognize previously visited locations and reduce the impact of accumulated drift. The number and accuracy of loop closures detected by ORB-SLAM2 were assessed against the known ground truth loop closures. Table 5.1 shows all the evaluation metrics that were tracked with respect to the considered scenario.

The number of tracked features can be used to measure the accuracy of the SLAM algorithm. A higher number of tracked features indicates that the SLAM algorithm can extract more information from the images, which can lead to better accuracy. The drift can be used to measure the stability of the algorithm with respect to SAM AUV. A lower drift indicates that the SLAM algorithm can maintain its accuracy over time while runtime can be used to measure efficiency. A lower runtime indicates that the SLAM algorithm can process the images more quickly. The robustness to noise can be used to measure the reliability and a higher robustness to noise indicates that the algorithm can handle noise and still produce accurate results. The map accuracy can be used to measure the completeness of the map created by the SLAM algorithm. A higher map accuracy indicates that the SLAM algorithm can create a more accurate representation of the environment.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matched Keypoints</td>
<td>1037</td>
</tr>
<tr>
<td>Drift</td>
<td>0.26 meters</td>
</tr>
<tr>
<td>Runtime</td>
<td>8.15 seconds</td>
</tr>
</tbody>
</table>

Table 5.1: Numerical Evaluation Metrics considered for ORB-SLAM2
5.2 Method 1: Monocular VISO2

<table>
<thead>
<tr>
<th>Seq</th>
<th>Timestamp</th>
<th>Got</th>
<th>Lost</th>
<th>Change Ref. Frame</th>
<th>Motion Estimate Valid</th>
<th>Num Matches</th>
<th>Num Inliers</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>1637241199.976651657</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>932</td>
<td>418</td>
<td>0.100163416</td>
</tr>
<tr>
<td>36</td>
<td>1637241200.169477422</td>
<td>False</td>
<td>False</td>
<td>True</td>
<td>True</td>
<td>1058</td>
<td>498</td>
<td>0.101614135</td>
</tr>
<tr>
<td>37</td>
<td>1637241200.378200374</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>375</td>
<td>150</td>
<td>0.066359972</td>
</tr>
<tr>
<td>38</td>
<td>1637241200.571087149</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>1150</td>
<td>538</td>
<td>0.111133861</td>
</tr>
<tr>
<td>39</td>
<td>1637241200.763803102</td>
<td>True</td>
<td>False</td>
<td>False</td>
<td>False</td>
<td>984</td>
<td>467</td>
<td>0.104465572</td>
</tr>
</tbody>
</table>

Table 5.2: Echoed Data /mono_odometer/info - Monocular VISO2 on Algae Farm Dataset

Table 5.2 summarizes the key information obtained from the /mono_odometer/info topic. The motion_estimate_valid flag indicates whether the motion estimate provided by the odometry algorithm is considered valid. Except the first two cases, the motion_estimate_valid flag was False, indicating that the algorithm encountered difficulties from hardware with respect to time stamp.
The got_lost flag, which indicates if the camera got lost during the estimation process, was initially False, suggesting that the camera remained tracked but eventually lead to some kind of delay in the synchronization of timestamp from the camera. The camera stream considered in this case was the left camera stream of SAM AUV (shown in Figure 4.3) The number of feature matches (num_matches) and inliers (num_inliers) varied for each message (as shown in Figure 5.4). However, despite the variation in the number of matches and inliers, the motion estimate was not considered valid. But the motion estimate held its validity status until the camera got lost while estimating the features.

Consequently, the lack of motion while performing visualization in RViz aligns with the findings from the odometry estimation results, indicating that the Monocular VISO2 faced challenges or limitations in accurately estimating the camera's motion when there is a minor hardware delay but as long as the movement of AUV was stable with respect to its height and pitch, which may indicate that the algorithm might work, as evident from the covariance visualization in RViz (as visible from the Figure 5.5), where the pose is determined as a part of the odometry results but becomes static when there are unsynchronized time stamps, which is indicated as “got_lost”.

The monocular visual odometry algorithm was applied to estimate the camera's trajectory and motion. Ground truth values (shown in Figure 5.4 and Figure 5.7) were employed in RViz to evaluate how well the estimate is in comparison to the covariance and pose of Monocular VISO2. As visible from the comparison, Monocular VISO2 started well with the pose estimation and did track features initially as shown in Figure 5.5 and Figure 5.6 but lost track once the timestamp synchronization issue occurred. This issue could also arise due to network delays in estimating the position. Monocular VISO2 is highly sensitive to both these issues, which were primarily due to hardware configuration issues that might have occurred while recording the bag file. So, it is evident from the quantitative results that Monocular VISO2 requires highly synchronized timestamp and zero network delay while performing motion estimation.
Figure 5.4: Ground Truth – Odometry derived from SAM AUV sensors (yellow lines indicate the traversed path based on the sensor information in SAM AUV)

Figure 5.5: Monocular VISO2 – Covariance and Pose Estimation with Sparse Features (red line indicates the initial camera pose and green dots indicate the features)
Figure 5.6: Monocular VISO2 - Odometer Values (visualized in RQT) (x represents time, y represents the feature matches)

Figure 5.7: Ground Truth – SAM AUV inbuilt odometry information (visualized in RQT) (x presents time, y represents the features)
### 5.3 Method 2: Stereo VISO2

<table>
<thead>
<tr>
<th>Time</th>
<th>Warning</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>13:13:12.38</td>
<td>Detected jump back in time of 163.055s. Clearing TF buffer.</td>
<td>/tmp/binarydeb/ros-melodic-tf2-ros-0.6.5/src/transform_listener.cpp:TransformListener::subscription_callback_impl:106</td>
</tr>
<tr>
<td>13:15:41.96</td>
<td>[stereo_processor] Low number of synchronized left/right/left_info/right_info tuples received.</td>
<td>/home/sbsk/catkin_ws/src/viso2/viso2_ros/src/stereo_processor.h: StereoProcessor::checkInputsSynchronized:84</td>
</tr>
<tr>
<td>13:13:13.90</td>
<td>[stereo_processor] Low number of synchronized left/right/left_info/right_info tuples received.</td>
<td>/home/sbsk/catkin_ws/src/viso2/viso2_ros/src/stereo_processor.h: StereoProcessor::checkInputsSynchronized:84</td>
</tr>
<tr>
<td>Time</td>
<td>Event Description</td>
<td>Source Path</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>13:15:11.97</td>
<td>[stereo_processor] Low number of synchronized left/right/left_info/right_info tuples received.</td>
<td>/home/sbsk/catkin_ws/src/viso2/viso2_ros/src/stereo_processor.h: StereoProcessor::checkInputsSynchronized:84</td>
</tr>
<tr>
<td>13:13:28.90</td>
<td>[stereo_processor] Low number of synchronized left/right/left_info/right_info tuples received.</td>
<td>/home/sbsk/catkin_ws/src/viso2/viso2_ros/src/stereo_processor.h: StereoProcessor::checkInputsSynchronized:84</td>
</tr>
<tr>
<td>13:15:26.97</td>
<td>[stereo_processor] Low number of synchronized left/right/left_info/right_info tuples received.</td>
<td>/home/sbsk/catkin_ws/src/viso2/viso2_ros/src/stereo_processor.h: StereoProcessor::checkInputsSynchronized:84</td>
</tr>
</tbody>
</table>

Table 5.2: Echoed Data /stereo_odometer/info - Stereo VISO2 on Algae Farm Dataset

Table 5.2 shows the warnings observed during the execution of Stereo VISO2, as a /stereo_odometer node in the algae farm dataset. The warning messages indicate potential issues with the overall performance of Stereo VISO2. The first warning suggests a detected jump back in time, which resulted in clearing of the transformation (TF) buffer. This primarily indicates a disruption or inconsistency again in timing of the data received from all three cameras which potentially affected the accuracy of odometry estimation.

The subsequent warnings highlight a low number of synchronized tuples of left/right/left_info/right_info messages, indicating that the stereo odometry algorithm received
an insufficient number of input images and camera information for processing image sequences from left and right cameras at the same time due to hardware synchronization of the stereo camera setup, similar to the one detected while evaluating Monocular VISO2.

The number of received images and camera info varied for each warning. As displayed in Figure 5.7, yellow warning messages are displayed while running the stereo_odometer node of Stereo VISO2, which means there are no errors in running the algorithm on the dataset but there is an issue with the hardware synchronization such as low number of synchronized tuples between the cameras and network delay. This shows that Stereo VISO2 is also very sensitive to hardware errors.

Disparity image (shown in Figure 5.8) shows the tracked features collectively from all the cameras and gets generated only when the stereo_image_proc node of Stereo VISO2 algorithms runs, so feature extraction does work. Right and left camera view (shown in Figure 5.9 and 5.10 respectively) shows how there is a slight delay in both frames. In this scenario, right camera has a synchronization delay compared to the left camera in the form of rectified images. The images are grayscale because Stereo VISO2 generates a rectified monochrome image sequences as frames after stereo_image_proc starts running successfully.

Stereo VISO2 did not generate any meaningful results (as visible from Figure 5.11) in comparison with the ground truth (shown in Figure 5.12) because of the hardware synchronization issue that was prevalent in the timestamps received from the stereo camera setup, which highlights its importance and the algorithm’s adherence to sensitivity.

Figure 5.8: Warning Messages for Stereo VISO2 in Terminal
Figure 5.9: Disparity Image – Features Tracked from All Cameras

Figure 5.10: Right Camera View – Rectified Image Frame in Algae Farm Dataset
Figure 5.11: Left Camera View – Rectified Image Frame in Algae Farm Dataset

Figure 5.12: RQT Plot – Results of Stereo VISO2 (empty values indicate odometer not running properly for the received camera images)
Chapter 6

Conclusion

This final chapter is a retrospective of the work done (Sections 6.1 and 6.2), as well as a look forward into the future to what other ideas can be explored or how the research can be progressed (Section 6.3) and reflections (Section 6.4) with regards to ethics and sustainability of this research to conclude.

6.1 Conclusions

To conclude, this research work is an experiment towards testing the feasibility of two chosen VO approaches – ORB-SLAM2 and VISO2 in the hydrobatic AUV setup that was manually created from the open-source resources available.

Based on the experiments we performed, ORB-SLAM2 runs properly and creates a map with loop closures indicating its high efficiency to estimate the camera trajectory with relevance to the ground truth but in a more robust way. Monocular VISO2 executes, but it fails to create a map limiting itself to tracking only some sparse features and estimating the initial position of the camera frame while Stereo VISO2 fails to run completely. Both Monocular and Stereo VISO2 require highly synchronized camera images with properly calibrated cameras and with that satisfied, they still have high potential for fast and efficient performance in such simulated scenarios based on the literature review we did. Currently, ORB-SLAM2 is the front runner for hydrobatic AUVs based on our experimental results.

In addition, this research has also demonstrated a software architecture stack for carrying an autonomy pipeline in hydrobatic AUVs to integrate VO and other software components and also, a simulated test case scenario is demonstrated in the form of a sample pool scenario, which can serve as a prototype for performing a single AUV test to check the feasibility of algorithms in hydrobatic AUVs with significantly less computation.
6.2 Limitations

The application of both Monocular VISO2 and Stereo VISO2 encountered an unforeseen roadblock (i.e.) absence of synchronized time stamps between all three cameras. This as we discussed emerged from the hardware configuration of the camera itself. The inability to harness the method's capabilities underlines the extreme high sensitivity of these algorithms in achieving accurate motion estimation and depth perception. The findings underscore that even the most promising methods can falter in the absence of appropriate prerequisites. Monocular VISO2 shows promise until the synchronization issue from the left camera but Stereo VISO2 failed to produce any significant results but was able to detect and track the features.

Manual configuration and calibration of the cameras was a significant limitation for this research as it did make an impact in improving the results of the findings in this research. Moreover, the limited availability of datasets that suit the purpose of this research work is also a major limitation due to which, there was a necessity to manually record a dataset in the ROS environment. Testing algorithms in hydrobatic AUVs is a time-intensive and economically very expensive process so limited research with regards to maritime simulation and scenarios in Stonefish also was a major hurdle as these algorithms could not be tested in different simulated scenarios at the same time. Designing multiple scenarios in Stonefish and manually recording multiple datasets are beyond the scope of this project hence the chosen algorithms and dataset are in adherence to the scope creep and purely focused on validating the research question of this project.

6.3 Future Work

This research project lays out the basis for in-depth studies on the application of feature-based VO methods to the underwater domains, particularly targeting hydrobatic AUVs, and future work on this field can be carried forward in several directions as ocean is the least explored part of our planet and there are endless possibilities based on the target scenarios.

Firstly, as already mentioned, properly synchronized timestamps from the camera and publicly accessible camera configuration parameters with accurately calibrated cameras will significantly improve the performance of all three algorithms tested in this research. Even ORB-SLAM2 will perform much better with a better dataset recorded from better cameras with highly sophisticated hardware. VISO2 can provide exciting insights on to how it estimates visual odometry in comparison to the wheel odometers present in hydrobatic AUVs, if the future research is purely focused on more feature-rich and challenging underwater datasets.

Secondly, this work was limited to only one single underwater dataset recorded in simulation via Stonefish for the evaluation. We did make it as challenging as possible with the limited resources, it is indeed restricted to a single scenario with just some varying conditions. Hence, future work may also investigate replicating this various such scenarios as this covers only a drop of the ocean.
6.4 Reflections

This whole work is targeted towards the benefit of the ocean and the society by considering the fact that how AUVs and hydrobatic AUVs can impact and create a very positive outlook in terms of disrupting the following domains of research:

- Safety and efficiency: AUVs can reduce the risks and costs associated with human diving operations by operating at depths and conditions that are impossible or dangerous for humans, such as deep oceans, underwater caves, polar regions, or contaminated areas [4]. AUVs can also perform tasks that are tedious, repetitive, or time-consuming for human divers, such as pipeline inspection, seafloor mapping, or water quality sampling [5]. By using AUVs instead of human divers, we can protect human lives and health, as well as improve the efficiency and quality of underwater operations.

- Knowledge and welfare: AUVs can contribute to the advancement of scientific knowledge and social welfare by collecting data and images that are otherwise inaccessible or difficult to obtain by conventional means, such as surface ships or observatories [4]. AUVs can also provide real-time or near-real-time information that can help track and respond to dynamic phenomena, such as oil spills, algal blooms, or climate change [5]. By using AUVs for scientific purposes, we can enhance our understanding of the ocean and its ecosystems, as well as support decision-making and policy making for ocean management and conservation.

- Innovation and creativity: AUVs can foster innovation and creativity in underwater technology by constantly evolving and improving in terms of design, performance, functionality, and autonomy [4]. AUVs often draw inspiration from nature and mimic marine life forms, such as fish, jellyfish, or octopus [5]. AUVs also challenge engineers and researchers to develop novel solutions for navigation, communication, sensing, and manipulation in the underwater environment [6]. By using AUVs for technological development, we can stimulate scientific curiosity and discovery, as well as create new opportunities and applications for underwater exploration.

- Environmental impact: Compared to surface vessels or manned submersibles, AUVs have lower power and emission rates, which can lessen the environmental impact of underwater operations [4]. When AUVs adopt bio-inspired forms and motions, they also cause the least amount of disruption and harm to marine species and ecosystem [5]. Instead of employing traditional platforms, we can use AUVs to maintain the ocean's natural beauty and diversity while also saving energy and resources.

- Ocean health: AUVs can support the monitoring and protection of ocean health by collecting data and images that can help assess the status and trends of the ocean parameters and processes, such as temperature, salinity, pH, currents, waves, tides, etc. [4]. AUVs can also detect and measure the presence and effects of pollutants,
Conclusion

contaminants, invasive species, or other threats to the ocean environment [5]. By using AUVs for environmental monitoring, we can identify and mitigate the sources and impacts of ocean degradation, as well as promote the restoration and resilience of the ocean ecosystem.

- Ocean use: By aiding in the discovery and extraction of natural resources from the ocean floor or water column, such as minerals, metals, oil, gas, or biomass, AUVs can enable new kinds of sustainable ocean use and exploitation [4]. Also, tidal, and thermal energy from the water may be developed and maintained with the use of AUVs [5]. By utilizing AUVs for maritime resource management, we can diversify and safeguard our energy supply while maximizing the consumption and distribution of ocean resources.

Hydrobatic AUVs can be beneficial in novel use cases such as

- Agility and efficiency: Hydrobatic AUVs are very agile and can perform challenging maneuvers that encompass the full 0–360 flight envelope, such as loops, rolls, or spins [1]. Such AUVs can be beneficial in novel use cases in ocean production, environmental sensing, and security, by enabling new capabilities for docking, inspection, or under-ice operations [1] [7]. Hydrobatic AUVs can also be efficient in terms of range and speed, by optimizing their hydrodynamic performance and reducing their drag and power consumption [1] [7]. By using hydrobatic AUVs instead of conventional AUVs, we can improve the flexibility and versatility of underwater operations, as well as reduce the operational costs and time.

- Simulation and verification: Hydrobatic AUVs require accurate and efficient simulation models to generate and test new maneuvers and control strategies [1]. Such models need to capture the nonlinear effects and turbulence that occur at high angles of attack, which are not well described by traditional methods [1]. Hydrobatic AUVs also require rigorous verification methods to ensure their safety and reliability in real-world scenarios [1]. Such methods need to account for the uncertainties and disturbances that affect the underwater environment, such as currents, waves, or obstacles [1]. By using simulation and verification tools for hydrobatic AUVs, we can enhance the quality and robustness of underwater technology, as well as reduce the risks and errors of underwater operations.

- Cyber-physical systems: Hydrobatic AUVs can be integrated with cyber-physical systems (CPSs), which comprise a network of sensors and actuators that relate to a computing and communication core. Such systems can enable new functionalities and applications for hydrobatic AUVs, such as adaptive sampling, collaborative autonomy, or swarm intelligence [8]. CPSs can also facilitate the development and maintenance of hydrobatic AUVs, by providing virtual validation, real-time feedback, or remote control [8]. By using CPSs for hydrobatic AUVs, we can increase the scalability and adaptability of underwater technology, as well as improve the usability and accessibility of underwater operations.
References


Appendix A

Node and TF Tree for Stereo VISO2 (in RQT)