A COMPARATIVE STUDY OF FEATURE DETECTION METHODS FOR AUV
LOCALIZATION

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Andrew Kim

June 2018

© 2018
Andrew Kim
ALL RIGHTS RESERVED

COMMITTEE MEMBERSHIP

TITLE: A Comparative Study of Feature Detection
Methods for AUV Localization

AUTHOR: Andrew Kim

DATE SUBMITTED: June 2018

COMMITTEE CHAIR: Lynne Slivovsky, Ph.D.
Professor of Computer Engineering

COMMITTEE MEMBER: John Seng, Ph.D.
Professor of Computer Engineering

COMMITTEE MEMBER: Jane Zhang, Ph.D.
Professor of Electrical Engineering

ABSTRACT

A Comparative Study of Feature Detection Methods for AUV Localization

Andrew Kim

Underwater localization is a difficult task when it comes to making the system autonomous due to the unpredictable environment. The fact that radio signals such as GPS cannot be transmitted through water makes autonomous movement and localization underwater even more challenging. One specific method that is widely used for autonomous underwater navigation applications is Simultaneous Localization and Mapping (SLAM), a technique in which a map is created and updated while localizing the vehicle within the map. In SLAM, feature detection is used in landmark extraction and data association by examining each pixel and differentiating landmarks pixels from those of the background. Previous research on the performance of different feature detection methods have been done in environments such as cisterns and caverns where the effects of the ocean are reduced. Our objective, however, is to achieve robust localization in the open ocean environment of the Cal Poly pier. This thesis performs a comparative study between different feature detection methods including Scale Invariant Feature Transform (SIFT), Speeded-Up Robust Features (SURF), and Oriented FAST and Rotated BRIEF (ORB) on different sensors. We evaluate the feature detection and matching performance of these algorithms in a simulated open-ocean environment.

ACKNOWLEDGMENTS

Thanks to:

- My parents, Steve and Sook, for all the love and support they provide in everything I do.
- My brother, Alex, for his encouragement and always being by my side.
- My friends, Joey Angeja and Jonathan Tan, for making this journey with me and for getting me through tough times.

TABLE OF CONTENTS

	Page
LIST OF TABLES	viii
LIST OF FIGURES	ix
CHAPTER	
1 Introduction	1
2 Related Work	3
2.1 Image Processing Techniques	4
2.1.1 Histogram Equalization	5
2.1.2 Otsu Threshold	5
2.1.3 Regions of Interest	6
2.2 Feature Detection Algorithms	6
2.3 Feature-Based and Texture-Based Approaches	7
2.4 SIFT (Scale-Invariant Feature Transform)	7
2.5 SURF (Speeded-Up Robust Features)	10
2.6 ORB (Oriented FAST and Rotated BRIEF)	12
3 Experimental Design	14
3.1 Target Environment	14
3.2 Environment Simulation	15
3.2.1 UWSim	15
3.3 Experimental Procedure	19
3.3.1 Data Capture	20
3.3.2 Preprocessing	21
3.3.3 Feature Detection and Matching	28
4 Evaluation Results	30
4.1 Feature Detection Performance	30
4.2 Feature Matching Performance	34
4.2.1 Single-Frame Rotation Results	35
4.2.2 Multi-Frame Sequence Results	40
5 Conclusions and Future Work	48

5.1 Future Work	49
BIBLIOGRAPHY	51
APPENDICES	

LIST OF TABLES

Table		Page
2.1	Summary of Evaluated Feature Detection Algorithms	13
4.1	Feature Detection Results for Range Camera (Averaged over 600 images)	34
4.2	Feature Detection Results for Standard Camera (Averaged over 600 images)	34
4.3	Single Frame Matching Results for Range Camera	38
4.4	Single Frame Matching Results for Standard Camera	39
4.5	Frame Sequence Matching Results for Range Camera	45
4.6	Frame Sequence Matching Results for Standard Camera	46

LIST OF FIGURES

Figure	Page
2.1 Outline of Underwater Navigation Methods [14]	4
2.2 SIFT Process [12]	8
2.3 SURF Process [13]	11
3.1 UWSim Shipwreck Scene With Girona500	17
3.2 Simulation Vehicle Path for Data Capture	21
3.3 Camera and Range Camera Sensor Data	22
3.4 Standard Camera Preprocessing Pipeline	23
3.5 Grayscale Original Image	24
3.6 CLAHE Results	24
3.7 Median Blur	25
3.8 Otsu Threshold	25
3.9 Canny Edge Detection	26
3.10 Final Image after Preprocessing	26
3.11 Effects of CLAHE on Image Histogram	27
4.1 SIFT, SURF, and ORB Features for the Range Camera Sensor . . .	31
4.2 SIFT, SURF, and ORB Features for the Standard Camera Sensor .	32
4.3 Range Camera SIFT, SURF, and ORB Single-Frame Rotation Matches at 60 Degrees	36
4.4 Standard Camera (CLAHE) SIFT, SURF, and ORB Single-Frame Rotation Matches at 60 Degrees	37
4.5 Range Camera SIFT, SURF, and ORB Mean and Deviation for Rotation Frame-by-Frame	40
4.6 Standard Camera SIFT, SURF, and ORB Mean and Deviation for Rotation Frame-by-Frame	41
4.7 Range Camera SIFT, SURF, and ORB Frame Sequence Matching	42
4.8 Standard Camera (CLAHE) SIFT, SURF, and ORB Frame Sequence Matching	43

4.9	Range Camera SIFT, SURF, and ORB Mean and Deviation for Sequential Frame-by-Frame	46
4.10	Standard Camera SIFT, SURF, and ORB Mean and Deviation for Sequential Frame-by-Frame	47

Chapter 1

INTRODUCTION

Since the inception of autonomous vehicles in the 1970s, technology has seen advancements in speed, efficiency, and power, which allow for more useful and practical applications. As a result, many tasks that previously involved manual control by an operator are now being replaced with these autonomous vehicles. A great amount of groundbreaking research has been done on the topic of autonomous navigation in the air and on the ground; however, the same level of research has not been done for underwater applications due to environment specific challenges. The primary challenge of underwater navigation and localization is due to the rapid attenuation of higher frequency signals as well as the unstructured nature of the underwater environment [14]. Above water, most autonomous systems rely on radio and global positioning signals to localize themselves on a map. These signals can only travel short distances underwater, which is not ideal for applications that operate beneath the waters surface. For this reason, autonomous underwater vehicles (AUV) are equipped with acoustic or visual sensors as they perform significantly better than radio signals. Unfortunately, these sensors are susceptible to other environmental challenges such as poor water visibility, waves, and wildlife.

Research regarding autonomous underwater navigation and localization has significantly increased within the past decade. The exponential increase in computational power and smaller form factor of computers allowed for a shift from older technologies involving predeployed beacons to localize the AUV, to more efficient methods such as simultaneous localization and mapping (SLAM). SLAM is the process in which a robot autonomously creates a map of its environment, and at the same time localizes itself in the generated map [14]. A feature-based SLAM algorithm consists of two

steps: the feature detection step in which robust features are extracted from the sensor data, and the state estimation step in which the robot estimates its pose according to the extracted features. There are various methods of performing each phase of the SLAM process, resulting in large variability in its implementation. Different sensors can be utilized based on the target environment of the robot; furthermore, a range of different state estimation algorithms can be used including the extended Kalman filter (EKF) and particle filter (PF). Some incredible work has been done using AUVs in [Clark et al.] and [Dobke et al.] where SLAM was used to generate a map of underwater cisterns and caverns. Environments such as cisterns and caverns are ideal for AUVs and SLAM as the underwater structures reduce and eliminate much of the dynamic challenges of the ocean such as waves and current. However, not much work has been done in the open ocean as the environment is unpredictable and extremely difficult to model and compensate for.

This paper examines the feature detection portion for autonomous underwater localization in an open ocean environment. The goals of this thesis include comparing different feature detection methods with various sensors and identifying the optimal feature detection pipeline for monitoring and maneuvering around the obstacles of a pier in open ocean. The rest of this paper is outlined as follows: Chapter 2 reviews related works that have been done in this area of study as well as provide background information, Chapter 3 describes the experimental design and the feature detection pipeline, Chapter 4 reviews the performance of the feature detection methods and provides a comparison of them, and Chapter 5 explains the conclusions that are drawn and possible future work.

Chapter 2

RELATED WORK

Localization is the act of determining the location of an autonomous agent given sensory data of the environment. Any aim to achieve autonomy in a robot usually requires some sort of localization method [18]. The distinction should also be made between localization and navigation. Navigational accuracy is the measurement of precision with which the agent guides itself from one point to another. On the other hand, localization accuracy can be described as the error in how well the agent localizes itself within a map. A taxonomy of localization and navigation methods are shown in Figure 2.1. In general, AUV localization and navigation methods can be placed into one of three categories [14]:

- **Inertial/dead-reckoning:** the use of accelerometers and gyroscopes for increased accuracy of the current state, but suffers from position error growth
- **Acoustic transponders and receivers:** based on measuring the time-of-flight (TOF) of signals from acoustic beacons for navigation
- **Geophysical:** the use of external environment information as references for navigation with the use of sensors and processing features

Feature-based SLAM utilizes sensors such as sonars and cameras along with positional sensors in order to detect features in the environment around the AUV and to localize itself. Therefore, SLAM can be identified within the inertial and geophysical categories of navigation. In recent years, the SLAM method has been developed to be utilized in a vast number of applications and environment, both on land and underwater. For example, in [Clark et al.] and [Dobke et al.], a small underwater Remotely

Figure 2.1: Outline of Underwater Navigation Methods [14]

Operated Vehicle (ROV) called the VideoRay Pro III was used to map cisterns in Malta and Gonzo using an implementation of SLAM.

2.1 Image Processing Techniques

The sensors used to gather the data is equally important as the means of detecting features. Although they are not the most reliable methods of obtaining robust data, optical imaging using video and camera film are still used for oceanographic operations. Some of the obstacles that may impede the performance of these underwater imaging platforms include rapid attenuation and the backscatter of light in the ocean [21]. High dynamic range cameras as well as image preprocessing techniques can be used to combat the lack of consistent illumination underwater.

2.1.1 Histogram Equalization

There are various image processing techniques that can be used to assist in detecting features in images that are affected by inconsistent lighting. Contrast Limited Adaptive Histogram Equalization (CLAHE) is an improved version of the Adaptive Histogram Equation (AHE) algorithm [20]. AHE is an image processing technique used to redistribute the lightness values of the image by computing several local histograms, each pertaining to a separate section of the image. The issues of AHE come from the limiting contrast enhancements that occur in homogeneous areas, resulting in more pixels in the same range of grayscale. CLAHE is used to enhance the contrast of images by changing the intensity values. The process is done with small regions called tiles using bilinear interpolation to smooth neighboring areas. Experiments performed in [Olvera et al.] show that by using CLAHE prior to feature detection, features that were previously ignored are detected and improves overall object detection performance.

2.1.2 Otsu Threshold

In addition to using preprocessing techniques to improve feature detection performance, threshold segmentation techniques such as the Otsu method can separate objects from noise and background information. An improved Otsu threshold segmentation method (TSM) proposed in [Yuan et al.] paired with the Canny edge detection was able to separate objects of interest, such as a shipwreck from a high-resolution image, more accurately compared to the iterative, local, and maximum entropy TSMs.

2.1.3 Regions of Interest

A variety of computer vision techniques can be used in conjunction to detect a more robust set of features. [Aulinas et al.] provides a method of detecting robust salient features from images under poor lighting conditions. The process begins with an image preprocessing stage which consists of single channel selection (gray-scale), followed by non-uniform light correction and normalization. The second stage of the process is focused on detecting the regions of interest (RoIs) in order to compute and detect keypoints of smaller regions where objects are likely to exist, rather than the entire image. Two parallel segmentation processes are computed: the first starting with edge detection followed by erosion/dilation operations, the second using a threshold on the hue channel based on the mean of the hue values. After the regions are computed in each segmentation process, the results are fused and the intersecting RoIs are used in the feature detection and matching algorithms. With the combination of the proposed RoI computation and SURF feature extraction methods, there was a significant improvement on the final SLAM estimate in the conducted experiments.

2.2 Feature Detection Algorithms

The localization of an AUV is highly dependent on the features that are extracted from the incoming sensor data. A feature detection algorithm computes points of interest, called keypoints, in an image which typically includes corner-like features that have high repeatability. A feature description algorithm then computes a distinct signature of a keypoint with the goal of identifying the same keypoint across multiple images [7]. These feature signatures are aimed to be invariant to changes in lighting, translation, and scale, making it an ideal solution for 3D mapping. For example, in a visual SLAM application, feature signatures can be compared between different frames to compute the trajectory of a camera or to identify previously visited parts of the environment.

This paper looks at three different feature detection methods: Scale-Invariant Feature Transform (SIFT), Speeded-Up Robust Features, and Oriented FAST and Rotated BRIEF (ORB).

2.3 Feature-Based and Texture-Based Approaches

Two basic approaches of feature detection include feature-based and texture-based detection methods. Feature-based detection focuses on identifying context-rich, although sparse, information throughout an image. Some of the first attempts in solving this detection problem include edge and corner detectors, as corners and edges correspond to areas of high information content. In other words, they are highly reliable points for matching between different images. However, feature-based detection methods do not perform as well as expected for changes in scale, illumination, rotation, or affine transformation. Texture-based detection methods were developed for situations where context-rich information is not available in an image. In other words, they are capable of extracting and matching feature between images despite the presence of textured backgrounds and lack of well-defined edges [6]. When compared to those of the feature-based methods, features detected by texture-based methods are simpler and tend to be discovered in dense areas. In recent years, more advanced algorithms for feature detection have been developed including the SIFT, SURF, and ORB algorithms. Table 2.1 provides a summary of the strengths and weaknesses for each algorithm.

2.4 SIFT (Scale-Invariant Feature Transform)

One of the first pioneers of texture-based feature detection and matching is David Lowe. In [Lowe], Lowe describes his method of extracting keypoints from an image and computing its descriptors called Scale-Invariant Feature Transform (SIFT).

Figure 2.2: SIFT Process [12]

Lowe explains that the SIFT process can be subdivided into four steps: scale-space extrema detection, keypoint localization, orientation assignment, and feature descriptor assignment.

The first step of scale-space extrema detection is performed to detect local points of interest within both scale and space. In order to find potential extremas, the Laplacian of Gaussian (LoG) filter is applied to an image with various values of σ . The σ value acts as a scaling parameter as the change in value allows for the LoG algorithm to detect blobs of different sizes. For example, when applying a gaussian kernel with a lower σ value on an image, higher values are given for small corners. On the other hand, a gaussian kernel with a higher value for σ results in higher values for larger corners. However, due to the fact that the LoG operation is quite computationally expensive, SIFT uses the Difference of Gaussians (DoG) instead. The DoG is an approximation of LoG obtained by taking the difference of Gaussian blurring an image with two different values of σ . This process is repeated for different octaves in the Gaussian Pyramid as shown in Figure 2.2a. Once the DoG are found, the images are searched for local extrema, also known as potential keypoints, in both scale and space.

The second step of the SIFT process is to refine the results by filtering through

the potential keypoints. This is done by applying a Taylor series expansion of scale space to get a more accurate location of the local extrema. The process of searching the image for local extrema over scale and space is illustrated in Figure 2.2b. If the intensity value of the extrema in question is less than a defined threshold value, it is rejected. After sorting through the potential keypoints, the edges in the image are removed as DoG have a higher response with edges. This is done by using a Hessian Matrix to compute curvature and applying an edge threshold, eliminating low-contrast keypoints and edge keypoints. All that remain after this step are strong points of interest.

These strong points of interest are considered our set of keypoints. The next step is to assign an orientation to each keypoint. This step plays an essential role in feature matching and achieving invariance in image translation and scale. The orientation state is assigned by looking at a neighborhood of pixels around the location of the keypoint and calculating the gradient direction and magnitude. An orientation histogram of 36 bins to cover 360° is then created to contribute to the stability of feature matching.

The final step of the SIFT process is to create the keypoint descriptor. A 16×16 neighborhood block is taken around the keypoint and is divided into 16 4×4 sub-blocks. For each sub-block, an 8-bin orientation histogram is generated, resulting in a total of 128 bin values. These values are stored as a vector to form the keypoint descriptor. These keypoint descriptors then allow for feature matching between different images by identifying the nearest neighbors and applying a ratio test to eliminate false matches.

2.5 SURF (Speeded-Up Robust Features)

Although Lowe’s SIFT algorithm contributed greatly in the area of robust texture-based feature detection and matching, one issue with the algorithm is speed. It is comparatively slower than other methods due to the complex computations that are performed in each step of the process. In 2006, [Bay et al.] introduced a faster version of the SIFT algorithm called Speeded-Up Robust Features (SURF)

SURF consists of the same four steps that are involved in the SIFT algorithm, but the methods of performing each step differ in a way to speed up the process. In the first step of finding the scale-space, Lowe used the Difference of Gaussian in order to approximate the Laplacian of Gaussian. The SURF algorithm goes a step further by approximating the LoG by using a box filter. The reasoning behind this decision is that the convolution of the box filter can be easily calculated by using integral images. This can be done in parallel for different scales and locations calculated using the determinant of the Hessian matrix. Examples of the approximation performed using box filters can be seen in Figure 2.3a. Another important improvement is the use of the sign of Laplacian (trace of Hessian Matrix) in order to distinguish light blobs on dark backgrounds from the reverse situation. In the matching phase, this data is used to only allow for matches that are of the same contrast type shown in Figure 2.3c. This extra information increases the speed at which the algorithm is executed without sacrificing accuracy.

In order to speed-up the orientation calculation and assignment for each keypoint, the SURF algorithm uses wavelet responses in both the vertical and horizontal directions with adequate Gaussian weights applied. The dominant orientation is then determined by calculating the sum of all responses in a sliding orientation window of 60° . Figure 2.3b illustrates plotted wavelet responses with a shaded sliding orientation window. SURF also provides the option to turn off orientation calculation for

(a) Box Filter Approximation of LoG

(b) Wavelet Response for Orientation

(c) Contrast Matching Between Images

Figure 2.3: SURF Process [13]

images that are not affected by rotation, further speeding up the process.

When it comes to creating the keypoint descriptor, the SURF algorithm once again calculates the wavelet responses in the vertical and horizontal directions. For each keypoint, a neighborhood around it is taken and divided into 4x4 subregions. The wavelet response is then calculated for each subregion and a vector with 64 dimensions is formed as the keypoint descriptor.

These alterations make the SURF algorithm almost three times as fast as SIFT without sacrificing a significant amount of accuracy. However, while the SURF algorithm works well in situations including blurred images and rotation, its performance

degrades under changes in viewpoint or illumination.

2.6 ORB (Oriented FAST and Rotated BRIEF)

A computationally efficient alternative to the SIFT and SURF feature detection algorithms is the Oriented Fast and Rotated BRIEF (ORB) descriptor brought up by Ethan Rublee, Vincent Rabaud, Kurt Konolige and Gary R. Bradski in [Rublee et al.]. In short, ORB utilizes the (Features from Accelerated Segment Test) FAST corner detection algorithm to extract keypoints and the (Binary Robust Independent Elementary Features) BRIEF descriptor with many modifications and enhancements for better performance. The paper states that ORB is much faster than both SIFT and SURF, and that the ORB descriptor performs much better than the SURF.

The FAST corner detection algorithm is the method of choice in finding keypoints due to its efficiency and ability to find robust corner features. FAST's speed and efficiency makes it a good candidate algorithm for real-time applications such as SLAM in which other feature detection algorithms are too slow. When used as a keypoint detector for ORB, however, it must be augmented with a pyramid scheme for scale and a Harris corner filter to reject edges and provide a reasonable score. The issue with the FAST algorithm is the lack of an orientation operator for mapping features between different images that other algorithms such as SIFT and SURF have. There are various ways to determine and describe orientation including using histograms of gradient computations (SIFT), or approximating block patterns (SURF). These methods are either computationally expensive, or yield poor results. As a result, the authors of [Rublee et al.] utilize a centroid technique that gives a single dominant result for each keypoint, rather than multiple values. More specifically, intensity weighted centroid is calculated for each patch with the corner detected by the Harris operator is in the center. The direction of the vector between the centroid and the

corner yields the orientation of the keypoint [11].

Once the keypoints are extracted and the orientation is determined, the ORB algorithm utilizes a modified version of the BRIEF feature descriptors. The original BRIEF algorithm performs poorly in situations that involve rotation. As a result, ORB includes a learning step to find less correlated binary tests which lead to a better descriptor rBRIEF, or a rotation-aware BRIEF [16]. This new descriptor is achieved by steering the original BRIEF according to the orientation of the features. For a feature set of binary tests, a matrix S is defined which contains the coordinates of these pixels. Then, by using the orientation of the patch, a rotation matrix is found and is used to create a rotated version of S . The set of points in the orientation-influenced matrix are used to compute the descriptor.

Descriptor	Speed (Relative to SIFT)	Performs Well For	Degrades Under
SIFT	1	Blob Detection, Scale, Illumination, Rotation	-
SURF	3 times faster than SIFT	Blurred images, Rotation	Viewpoint, Illumination
ORB	Faster than both SIFT and SURF	Corner Detection, Rotation	Blob Detection

Table 2.1: Summary of Evaluated Feature Detection Algorithms

Chapter 3

EXPERIMENTAL DESIGN

In our experimentation, we evaluate the performance of the three different feature detection algorithms (SIFT, SURF, and ORB) discussed in the previous section. Our evaluation of algorithm performance consists of how feature detection is handled in a single frame as well as the feature matching performance between different frames. In addition to comparing different algorithms, we also compare how different sensors affect the results of the feature detection algorithms. The sensors that we compare include a standard high-definition camera and a range camera. In comparing the performance of feature detection on these two sensors, we examine how the detected features and the matching performance between the sensors differ with and without noise.

3.1 Target Environment

One of the major factors that must be accounted for in designing an underwater navigation system is the target environment. Different underwater environments and mission types affect the methodology in which an AUV is localized. For example, in applications involving caverns where the water is clear, using a single camera may be sufficient in detecting all structures necessary to successfully localize the AUV. The mission of our AUV, on the other hand, is to autonomously localize and navigate through the Cal Poly pier to examine and record growth on the pier pilings.

Since the pier is in the open ocean, the quality of the attainable sensor data is fully dependent on the constantly changing visibility of the water. This is more so for the standard camera as murky water will deem the sensor unviable to use, while the

range camera will still be able to produce high resolution data due to its reliance on ultrasonic waves rather than light. It is also important to note the forward orientation of the sensors as we are focused on examining what is in front of the AUV.

3.2 Environment Simulation

For development purposes, we utilize a robot simulator to emulate the environment and to capture sensor data. By using an environment simulator, we can adjust different variables in the water to help recreate different scenarios and objects. The prerequisites we have set for a development simulator include an accurate model of the underwater environment, an AUV with camera and sonar sensor support, and scenes that contain similar objects as the pier. With these requirements in mind, we use a simulator called UWSim [19], an open-source project designed for underwater vehicles.

3.2.1 UWSim

UnderWater SIMulator (UWSim) is an underwater simulator that is primarily used for marine robotics research and development. This simulator visualizes an underwater virtual scenario that can be configured using standard modeling software, allowing for the user to create 3D models and objects to build scenes that represent the target environment. Based on a survey [Cook et al.], when compared to other underwater simulators such as MORSE and Gazebo, UWSim is a more realistic simulation of an underwater environment due to its high visual fidelity. UWSim also supports various underwater vehicles including surface vessels and robotic manipulators along with simulated sensors for each vehicle. The robot and the sensors can be accessed externally through network interfaces which allow for easy integration between the visualization tool and previously implemented control architectures. These supported

features, which will be discussed in further detail, make UWSim a viable initial development platform for our research.

Configurable environment

One feature that UWSim is known to have is an accurate visual representation of an underwater environment. The simulator allows the user to configure 3D scenes by the use of an XML file. These scenes can be created with standard modeling software with file types supported by the OpenSceneGraph (OSG) programming interface [19]. Once the objects are created, they can be imported in the XML code. New nodes are automatically added to the defined environment model to create underwater visualization effects such as the ocean surface, underwater visibility, water color, silt particles, god rays, etc. UWSim also includes pre-installed scenes that may be used out-of-the-box such as the shipwreck scene shown in Figure 3.1. Once the XML is configured and the visualizer application is launched, the effects can be toggled or changed by using the specified key bindings which can be viewed in the help menu of the simulator.

Multiple Robot Support

The UWSim software design also allows for the support of multiple different underwater vehicles and manipulators. This is possible by the specialization of the included abstract classes within the simulator. Vehicles and manipulators are composed of a 3D model (which can be designed by a third-party application) and is described using an XML file. This file includes the visual model of the vehicle as well as the kinematics and dynamics. Some of the default robots that are defined in UWSim include the Girona500 vehicle and the ARM5E manipulator. Within a single simulation, different robots can be loaded and simultaneously manipulated for missions

Figure 3.1: UWSim Shipwreck Scene With Girona500

that require multiple vehicles.

Simulated Sensors

In addition to the support of various robots, UWSim also provides twelve different sensors that may be used in conjunction with these vehicles and manipulator joints. These sensors are defined in the main XML file for the environment, specifically in the definition of the robot the sensor is being mounted on. The positioning and movement of these sensors operate in six degrees of freedom (6DOF). In our experiments, we utilize the camera and the range camera sensor. The other supported sensors are listed and described as follows [19]:

- **Camera:** Provides virtual images
- **Range Camera:** Depth image of the camera
- **Range Sensor:** Measures the distance to the nearest obstacle along predefined

directions

- **Object Picker:** Fakes object grasping when the object is closer to a predefined distance
- **Pressure:** Provides a pressure measure
- **Doppler Velocity Log (DVL):** Estimates the linear speed at which the vehicle is traveling
- **Inertial Measurement Unit (IMU):** Estimates the vehicle orientation with respect to the world frame
- **GPS:** Provides the vehicle position with respect to the world, but only works when the vehicle is near the surface of the water
- **Multibeam:** Simulates an array of range sensors, providing distances to nearest obstacles in a plane at constant angle increments
- **Force:** Estimates the force and torque applied to the vehicle part
- **Structured Light Projector:** Projects a laser or regular light on scene
- **Dredge:** Dredges mud for buried objects

Network Interface

The method through which these sensors and the robot are controlled is by means of external software through a network interface. The developers of UWSim integrated the simulator into an open-source meta-operating system for robots called the Robot Operating System (ROS). ROS provides the core functionality of interfacing with the robot and sensors in a simulation as it provides many facilities for communication and distributed computing. By sending messages through the network interfaces, the user

is able to access and update properties including velocity, acceleration, direction, etc. This allows for creation of pre-routed navigation routes as well as manual override and control of the vehicles and manipulator joints. Sensor data such as a video feed from a camera is also transmitted through this network interface.

The network interface that ROS operates on is composed of three components: nodes, topics, and messages. A node is simply a process that is doing computation. This general definition can be applied to the simulation of sensors, the processing of sensor data, and even moving the robot. These nodes are combined together within a graph where they communicate with one another using streaming topics and services. Topics are the buses in which nodes communicate with one another and are intended for unidirectional streaming. Nodes generally are unaware of who they are communicating with. Instead, they interface through a publisher/subscriber relationship. Nodes will subscribe to topics that are of interest; nodes can also publish generated data to a relevant topic. Finally, messages are the data that is being published within the topics. A message is a simple data structure that comprises of typed fields.

3.3 Experimental Procedure

Our experimental procedure consists of two parts: data capture and data processing. As described in the previous section, we utilize the UWSim simulator in order to capture data from a realistic and representative model of our target environment. The sensor data from the simulator is then passed to a Python script which performs the feature detection and matching algorithms.

3.3.1 Data Capture

Our experimentation setup consists of an AMD Ryzen 5 1600x 6-core processor, 16 GB of memory, and an Nvidia GTX 1080 graphics card. The higher core and thread count supported by the processor allows for more responsiveness from the simulator as multiple processes must be spawned and controlled simultaneously, and since UWSim primarily relies on the graphics performance of a system. For example, individual processes are required for the robot, camera, and range camera sensors. The graphics card handles all of the visualization and graphics rendering, and with the GTX 1080, we were able to achieve an average frame rate of 15 frames per second in our test scene.

The scene we use for data collection is a modified version of the included shipwreck scene. In order to use the desired robot and sensor configuration, we alter the XML file to include the Girona500 vehicle mounted with the camera and range sensors. The shipwreck scene was chosen as the base environment for our experimentation due to the similar features it contains when compared to the features of the pier. For instance, the wood mast of the ship is representative of the pilings of the pier. Another motivation is that the ropes and platforms on the ship can be seen as other obstacles that may be unexpected within the pier environment, such as growth or fishing lines.

The data captured includes video feeds from both the camera and range camera sensors. While the Girona500 moves along a predefined path around one of the ship masts shown in Figure 3.2, both sensors save image data at 15 frames per second at a resolution of 1920×1080 . This data is then passed to a Python program to perform feature detection and analysis.

Figure 3.2: Simulation Vehicle Path for Data Capture

3.3.2 Preprocessing

In the process of performing and evaluating the feature detection methods, we examine how the detection and matching performance compare for the different algorithms on different sensor data. The image processing and computer vision techniques are implemented using the OpenCV 3.3.0 library for Python. The videos captured from the simulator are broken into frames and stored as a list of image objects. Each image goes through a series of preprocessing steps prior to feature detection and matching. Two different preprocessing techniques are implemented, one for each sensor. Figure 3.3 shows image samples from the standard camera and range camera.

The preprocessing algorithm for the range camera data is minimal due to the fact that the sensor outputs a grayscale image in which objects closer to the camera have lower single-channel values than further objects. This provides a distinct contrast between the foreground objects and background. Also, as a result of the use of

(a) Range Camera Data

(b) Standard Camera Data

Figure 3.3: Camera and Range Camera Sensor Data

proximity for data instead of light, we eliminate the possibility of noise caused by non-uniform lighting. However, particulates and underwater organisms may be picked up as noise by the range camera. In order to eliminate most of the noise, we simply apply a 3×3 Gaussian filter with a convolution kernel.

The camera data requires more complex preprocessing algorithms in order to accurately detect features. The poor lighting in the environment as well as the large amount of diffusion calls for a methodology in which the foreground objects can be separated from the background. In order to achieve this goal, we use computer vision techniques to find the regions of interest (ROI). An overview of the preprocessing

algorithm for the camera data is shown in Figure 3.4. The resulting images from each step in the preprocessing pipeline are displayed in Figures 3.5 - 3.10.

Figure 3.4: Standard Camera Preprocessing Pipeline

We begin by converting the original image into grayscale, allowing us to work with a single color channel as shown in Figure 3.5. However, due to the poor lighting, the resulting image is still relatively dark. As a result, we used the previously discussed method of Contrast-Limited Adaptive Histogram Equalization. With this method, we are able to adjust the histogram of the original grayscale image to brighten up potential objects in the image. Figure 3.11 displays how the CLAHE function adjusts the histogram to improve contrast within the image. We see a slight separation from the lower values and the higher values. This allows for a greater distinction between darker and brighter pixels in the resulting image shown in Figure 3.6.

After the histogram equalization process, a median filter is applied to reduce the amount of noise due to non-uniform lighting, bubbles, and particulates. This noise can cause false-positives during feature extraction. As a result, the median filter is used to attenuate the effects of random floating particles.

Although the median filter will reduce the impact of particulate noise, it will not completely eliminate it as the particles vary in size. In order to further reduce noise and find our region of interest, we apply an Otsu threshold on the filtered image. As discussed in the previous chapter, the Otsu method can retain the information of

Figure 3.5: Grayscale Original Image

Figure 3.6: CLAHE Results

Figure 3.7: Median Blur

Figure 3.8: Otsu Threshold

Figure 3.9: Canny Edge Detection

Figure 3.10: Final Image after Preprocessing

(a) Original Grayscale Histogram

(b) Histogram After CLAHE

Figure 3.11: Effects of CLAHE on Image Histogram

objects while ignoring background and noise information. From the resulting binary image in Figure 3.8, we can see that both masts of the ship are extracted from the image with minimal amount of noise.

The binary image that results from the application of the Otsu method may be used as a mask to only expose the ROI within the original image. However, we further refine our ROI by performing a combination of edge detection and morphological filtering. We first determine all of the edges with the use of the Canny edge detector

as shown in Figure 3.9. With the newly discovered edges, we are able to perform a series of morphological operations on the edges in order to reduce the remaining noise. We first erode the edges in order to eliminate insignificant spots on the image. We then use dilation in order to join regions and fill the objects, resulting in the final mask containing our ROI. The mask is applied to the image that resulted from the CLAHE operation with the bitwise *AND* operation, exposing only the parts of the image in our defined ROI. The final image is shown in Figure 3.10.

3.3.3 Feature Detection and Matching

In our experimentation, we examine 10 second video sequences of both sensors. The frames are initially converted OpenCV matrices which are stored in a Python list. Each image then goes through the sensor-specific preprocessing step as described in the previous subsection. After the preprocessing is complete, we evaluate the SIFT, SURF, and ORB feature detection algorithms on the resulting image [12][13][11]. Both the preprocessing and feature detection parameter values are calibrated with the first frame of the video feed. Once the parameter values are calibrated, the three algorithms are run for each frame and the resulting feature descriptors are stored.

We also examine the feature matching performance of each algorithm in two parts. First, we look at how feature matching performs in response to changes of orientation of the same frame. One image is rotated in intervals of 60° with a max of 180° , and the features are matched and examined at each interval. The second test is to examine matching performance in continuous frames of a video stream. This is done by iterating through the resulting feature descriptors for each frame and matching them with a base image. The base image is defined to be the first image in a set of 15 frames. The set of frames ensures that the same objects are found in the same set, but at different locations and orientations.

The performance and efficiency of each feature detection algorithm is evaluated by the execution time, feature count, and distance between matches. In comparing the distance between matches, we calculate the mean and deviation of the distance arrays for each pair of matching images using Equations 3.1 and 3.2, where N is the number of descriptors and d is the distance. Here the mean and deviation values can be seen as a measure of accuracy. When an algorithm is trying to determine a potential match for a feature between two images, the distance is calculated between the descriptor of the original image and every descriptor in the second image. The lowest distance is a potential match as it is the feature descriptor that is most like the original descriptor. The mean is the average of all the matched distances that occur in a single image. The lower the mean value, the more likely the match is of the same object. The deviation value represents the average distance between the descriptor values and the calculated mean for the image. These calculations are performed for both the single-frame and multi-frame tests.

$$mean(\mu) = \frac{\sum_{i=1}^N d_i}{N} \quad (3.1)$$

$$deviation(\sigma) = \sqrt{\frac{1}{N} \sum_{i=1}^N (d_i - \mu)^2} \quad (3.2)$$

Chapter 4

EVALUATION RESULTS

The objective of our experiments is to evaluate the performance and efficiency of the SIFT, SURF, and ORB feature descriptors in an open-ocean environment for localization purposes. We examine how the three algorithms compare on single-frame and multi-frame tests using different sensor data. For each sensor, the range camera and standard camera, a single frame is rotated at 60° , 120° , and 180° to test the matching performance for controlled rotation invariance. To evaluate performance of a real-time application, we use a data stream of 150 frames from the UWSim simulator. The frames are divided into 10 bins of 15 frames, where the first frame is the base for matching the following 14 frames. In quantifying performance and efficiency, we take into account the quality of the features, execution time, and matching distances.

4.1 Feature Detection Performance

One primary function of feature descriptors is to detect and extract features from a given set of data. In order to test feature detection performance, we extract SIFT, SURF, and ORB features for each frame in our data set, including the rotated frames. We then examine the quality of the features in the output image as well as the efficiency of each algorithm. Figures 4.1 and 4.2 include the resulting SIFT, SURF, and ORB feature descriptors for the range camera and standard camera respectively.

From a visual perspective, we can see how these algorithms perform differently on the same image. The SIFT algorithm on the range camera shows detected features along edges of contrast with minimal amount of false-positives. The SURF features are not as precise and exist in the general area of the objects with a higher false-

(a) SIFT Features

(b) SURF Features

(c) ORB Features

Figure 4.1: SIFT, SURF, and ORB Features for the Range Camera Sensor

(a) SIFT Features

(b) SURF Features

(c) ORB Features

Figure 4.2: SIFT, SURF, and ORB Features for the Standard Camera Sensor

positive rate. This is expected as the SURF algorithm uses a faster, but less accurate, approximation of the LoG than the SIFT algorithm in order to improve the execution time. The ORB algorithm, on the other hand, does not detect quite as many features as the other two algorithms. However, it excels in detecting corners and points of intersection with the least amount of false-positives among the three algorithms.

The algorithms perform much differently on the standard camera as the data does not contain the same levels of contrast as the range sensor. Instead, the camera data depends on light from above the ocean surface to illuminate the objects. This issue is addressed by preprocessing the image, but it still does not achieve the level of clarity that the range camera provides. The three algorithms perform similarly to each other on the camera sensor. However, the SIFT algorithm detects more robust features along the poles of the mast where the SURF and ORB algorithms are lacking.

In order to test the time efficiency of the feature detection algorithms, we record the average keypoints and execution time for each. The results of running the feature detection algorithms on 150 frames of a video stream are shown in tables 4.1 and 4.2. The results show the average number of keypoints in an image, the execution time for an image in seconds, and the execution time per keypoint in nanoseconds. For the range camera, we see that the SIFT algorithm detected the most amount of keypoints while the ORB algorithm detected the least. The same rankings apply for the execution time for each image. However, the results for the standard camera do not yield the same results in average keypoint count. Instead, we see that the ORB algorithm detects more keypoints on average than both the SIFT and SURF. While the average execution times are comparable to those of the range camera, the drop in average keypoints for all descriptor types results in higher execution times per keypoint for the standard camera. The average execution time per keypoint can be seen as a measure of efficiency for the feature detection algorithm. We see the greatest increase in execution time per keypoint for the SIFT algorithm due to the

greatest drop in the average detected keypoints, representing a drop in efficiency. The SURF algorithm also sees a significant rise in time per keypoint. However, between the range camera and standard camera datasets, the ORB algorithms drops less than 100 keypoints. As a result, the average execution time per keypoint stays relatively the same.

Descriptor	Keypoints	Time (s)	Time per Keypoint (ns)
SIFT	2335	0.696	2.982
SURF	1711	0.220	1.283
ORB	500	0.0813	1.627

Table 4.1: Feature Detection Results for Range Camera (Averaged over 600 images)

Descriptor	Keypoints	Time (s)	Time per Keypoint (ns)
SIFT	342	0.683	19.954
SURF	192	0.177	9.244
ORB	409	0.0813	1.986

Table 4.2: Feature Detection Results for Standard Camera (Averaged over 600 images)

4.2 Feature Matching Performance

The other primary function of feature detection algorithms is to identify and match feature descriptors within different sets of data, invariant to image transformation. In order to evaluate the matching performance of each algorithm, we use two different test data sets: single-frame and multi-frame sequence performance.

4.2.1 Single-Frame Rotation Results

The first tests the rotation invariance for a single frame. For each of the 150 frames in the original dataset, the image is rotated in intervals of 60° to a max of 180° . Figures 4.3 and 4.4 show the matching results for a single-frame rotation test at 60° . The features are detected in the rotated image and matched with the original. We utilize the brute-force matching algorithm along with the Euclidean distance measure to match features between different frames. In brute-force matching, the descriptor of one feature in one set is matched with all the others in the second set. A distance measure, in our case Euclidean distance, is used to return the closest feature. In other words, the smaller the distance between the features of the two data sets, the more similar the two features are. For each descriptor, we record the average number of matches and average execution times, similar to the tests we performed for the detection process. In addition, we record the mean and deviation of the distance arrays for each match, both of which are measures of accuracy. A lower value for the mean of distances shows greater accuracy between matches as the average distance between features are lower. The deviation of the distance array is defined as the difference between the descriptor values and the mean of distances in the image. Tables 4.3 and 4.4 show the range camera and standard camera results for the single-frame rotation tests.

It is important to note that the number of matches is dependent on the number of keypoints detected. In the results, we measure the ratio between the matches found and the total number of keypoints detected for each algorithm. This measure can be found by dividing the number of matches by the total number of features detected. As we are using the same image and just rotating it, this is a test of rotation invariance. The higher the match percentage, the more rotation invariant the algorithm is. From the results, we can conclude that the SIFT algorithm is the most rotation invariant,

(a) SIFT Matches

(b) SURF Matches

(c) ORB Matches

Figure 4.3: Range Camera SIFT, SURF, and ORB Single-Frame Rotation Matches at 60 Degrees

(a) SIFT Matches

(b) SURF Matches

(c) ORB Matches

Figure 4.4: Standard Camera (CLAHE) SIFT, SURF, and ORB Single-Frame Rotation Matches at 60 Degrees

followed by ORB, then SURF from the match percentages of both sensors.

Unlike the measure of execution time in the feature detection process, the execution time for the matching process varies based on the number of matches made. The fact that every feature in one data set is compared to every feature in the second is the cause of this difference. A greater number of features detected result in a much greater number of comparisons that have to be made, effectively increasing the execution time. This can be seen when comparing the times between the two sensors. We see the greatest drop for the SIFT feature descriptor when comparing the range camera to the standard camera. The SURF algorithm also sees a drop in execution time, while the ORB algorithm stays relatively the same. In response, the time per match changes. We observe that the SIFT and SURF algorithms are much more efficient using the range camera while the ORB algorithm maintains a similar efficiency between the two sensors.

Descriptor	Matches	Match Ratio (%)	Time (s)	Time per Match (ns)	Mean	Deviation
SIFT	1415	60.60	0.274	1.939	5.380 - 44.350	17.641 - 41.653
SURF	592	30.91	0.174	2.937	0.00514 - 0.119	0.0162 - 0.0789
ORB	281	56.20	0.128	4.539	0.00905	0.0267

Table 4.3: Single Frame Matching Results for Range Camera

In measuring accuracy, we measure the mean and the deviation of the distance arrays for each matching pair of images. Both of these measurements may display a great range in values. Recall that the lower mean and deviation values correlate to greater accuracy in descriptor matching. From the mean and deviation ranges calculated, the ORB and SURF algorithms both contain the lowest values inferring higher accuracy. However, the ORB descriptor is the more consistent as it maintains

Descriptor	Matches	Match Ratio (%)	Time (s)	Time per Match (ns)	Mean	Deviation
SIFT	254	74.27	0.128	5.0552	14.717 - 84.204	24.919 - 87.999
SURF	103	53.65	0.111	10.743	0.0138 - 0.199	0.0165 - 0.145
ORB	259	63.33	0.123	4.746	0.0198	0.0422

Table 4.4: Single Frame Matching Results for Standard Camera

the same mean and deviation values throughout all tests. There is much variation within the SIFT and SURF algorithms with SURF being the more accurate one out of the two. The results also show that overall accuracy in matching is much better using the range camera than the standard camera. The graphs in Figures 4.5 and 4.6 display the distance means and deviations for the rotation intervals of a single frame for each algorithm. We can see the same reaction to rotation for both sensors. The interesting part is how the values suddenly drop when matching the image rotated at 180° using the SIFT and SURF methods. This drop signifies more accurate matches for the flipped image when compared with the other rotation angles. For the ORB descriptor, however, the mean and deviation values stayed constant throughout all rotation angles, exemplifying the consistent matching performance regardless of the rotation angle.

In summarizing the results for the single frame rotation tests, we see that the SIFT algorithm is the most rotation invariant for both sensors due to high match percentages. We also see that the SIFT algorithm is most efficient when taking into account the execution time per match on both sensors. However, the SIFT algorithm has the greatest overall execution time which may not be a viable solution to slower processing real-time systems. By looking at the mean and deviation of the distance arrays, we see that the ORB and SURF algorithms create more accurate matches, with

Figure 4.5: Range Camera SIFT, SURF, and ORB Mean and Deviation for Rotation Frame-by-Frame

ORB being the more consistent one. In comparing matching performance between the two sensors, we see higher match ratio and lower execution times on the standard camera, but at the cost of match accuracy.

4.2.2 Multi-Frame Sequence Results

Our second test set consists of a sequence of frames of a video stream where the AUV circles around a mast of the shipwreck while moving downwards. The goal of this test is to examine how the different feature descriptors handle matching in an uncontrolled environment. Unlike the single frame test where a single variable is changed, this multi-frame test simulates performance in a real-world application. Environmental factors such as changes in perspective and illumination are all present

Figure 4.6: Standard Camera SIFT, SURF, and ORB Mean and Deviation for Rotation Frame-by-Frame

within this test set. For each frame in a 150-frame video clip, we match the features discovered within a frame to a specified base frame. In order to match frames where the same objects exist between the base frame and the frame in question, we divide the video sequence into 10 bins of 15 frames. As the simulator ran on our system with an average framerate of 15 frames per second, each bin contains 1 second of the video. The base frame is specified as the first frame in each bin, which is then matched with the 14 following frames. Figures 4.7 and 4.8 show the matching results between a base frame and one of the following frames for the range and standard cameras.

The measurements we record in Tables 4.5 and 4.6 are identical to those of the single frame test. In comparison the the single frame test, we see the match ratio drop for both sensors, which is expected as we are no longer looking at the same

(a) SIFT Matches

(b) SURF Matches

(c) ORB Matches

Figure 4.7: Range Camera SIFT, SURF, and ORB Frame Sequence Matching

(a) SIFT Matches

(b) SURF Matches

(c) ORB Matches

Figure 4.8: Standard Camera (CLAHE) SIFT, SURF, and ORB Frame Sequence Matching

image, and some of the features are no longer visible at a different perspective. We see a greater drop in match ratio for the SIFT and SURF algorithms as they detect more features in a single image. Therefore, when the perspective is changed, we lose sight of more features for those two algorithms when compared to the ORB.

Just as we observed in the single frame test, we see that the execution time per frame is affected by the number of matches. The range sensor results exemplify this relationship as SIFT takes the longest amount of time with the most number of matches, and ORB taking the least amount of time with the least number of matches. These timings also affect the efficiency of the algorithm in terms of the time it takes to make each match. The three algorithms have similar execution time per match on the range sensor, but ORB algorithm takes much less time per match on the standard camera.

In measuring the accuracy of the algorithms, we look at the mean and deviation of the distance arrays for each image. The ranges for the mean and deviation values are much larger compared to the single frame test. This is due to the fact that adjacent frames have very slight changes in perspective, which result in smaller distances between matches. Frames that are further apart in time, however, experience a larger change in perspective which results in larger distances between matches. The standard camera yields larger values and larger ranges for both the mean and deviation because of the light-reliant property of the sensor. As the AUV moves, the illumination of the environment changes, causing the feature descriptor values for the same feature to vary more. Also, when the AUV descends deeper, we see a loss in illumination for the standard camera. The range camera, on the other hand, uses the distances to objects in order to generate high-contrast images that is consistent regardless of light.

We further investigate the accuracy of the algorithms in the graphs of the frame-

by-frame mean and deviation calculations shown in Figures 4.9 and 4.10. These graphs display the mean and deviation values for each frame matched with the specified base frame. The SIFT and SURF algorithms on the range sensor maintain fairly constant mean and deviation values as the AUV moves further away from the base frame position. For the ORB algorithm, the mean increases as the AUV moves further, and the deviation decreases in response. The differences between the algorithm responses may be due to the amount of matches that occur, where the SIFT and SURF algorithms create more feature matches than the ORB. The results for the standard camera show how the change in illumination can affect the matching performance. Both the SIFT and SURF results are much more sporadic and not nearly as consistent as the equivalent test with the range camera. The ORB results remain consistent throughout the frames, although it does not see the same pattern as the range camera results.

Descriptor	Matches	Match Ratio (%)	Time (s)	Time per Match (ns)	Mean	Deviation
SIFT	582	24.92	0.164	2.830	0.336 - 114.292	2.091 - 73.453
SURF	399	23.34	0.102	2.562	0.00053 - 0.129	0.0021 - 0.0843
ORB	266	53.15	0.083	3.141	0.091 - 50.70	0.654 - 18.987

Table 4.5: Frame Sequence Matching Results for Range Camera

Descriptor	Matches	Match Ratio (%)	Time (s)	Time per Match (ns)	Mean	Deviation
SIFT	68	19.99	0.0595	8.695	6.835 - 252.014	12.512 - 106.758
SURF	44	22.84	0.0545	12.450	0.00224 - 0.425	0.00193 - 0.110
ORB	188	45.88	0.0521	2.772	0.635 - 70.391	0.899 - 25.694

Table 4.6: Frame Sequence Matching Results for Standard Camera

Figure 4.9: Range Camera SIFT, SURF, and ORB Mean and Deviation for Sequential Frame-by-Frame

Figure 4.10: Standard Camera SIFT, SURF, and ORB Mean and Deviation for Sequential Frame-by-Frame

Chapter 5

CONCLUSIONS AND FUTURE WORK

Based on the results of our two test datasets, we are able to define the advantages and disadvantages of the different feature detection algorithms and sensors. Overall, the SIFT algorithm detects a large amount of robust features from an image while maintaining lower execution time per match in the matching process. However, the overall execution time per image is the worst out of the three images due its computational complexity, which may restrict performance on some real-time systems. The SIFT algorithm also yielded the highest distance values per match, meaning that the matching was not as accurate as the other two algorithms.

The SURF algorithm was designed to be a faster version of SIFT by using a faster, but less accurate, approximation of the LoG along with other optimizations. This is demonstrated in the results as the SURF execution times in both feature detection and matching are faster than that of the SIFT. The SURF algorithm also sees an improvement in matching accuracy. The disadvantage of SURF, however, is that the features extracted are not nearly as robust as the ones produced by the SIFT algorithm. The resulting features are scattered around the area of the object with higher false-positive rates.

The ORB algorithm was very consistent throughout the tests. It detects the least amount of features of the three algorithms, but it performs the best in the matching process. Although it yields a higher execution time per match, the ORB algorithm takes the least amount of time for total execution. The consistent performance of this algorithm is shown in the match ratio and distance accuracy measurements. The match ratio stays relatively consistent between the range and standard cameras. The mean and deviation of the match distances were both the lowest and most consistent

between all tests. However, the downside of the ORB method is lower feature count. As we observed, it performs excellently at detecting corners and intersection, but it fails to detect other essential features such as edges.

In comparing the two sensors tested, the range camera yielded the most consistent results as it is not affected by illumination or visibility conditions. From our tests, we found higher feature counts and match percentages with lower matching distance values on the range camera than the standard camera for all algorithms. We also verified the more consistent matching performance in real-time applications by measuring the mean and deviation of the match distances in a sequence of frames. For the targeted pier environment, the SIFT or SURF methods would both work well with this sensor as the pier pilings contain simple edges. The SURF algorithm, however, would perform much faster without sacrificing a great amount of feature robustness as the objects under the pier are quite simple. Due to the fact the the ORB algorithm fails to detect edges, it would not be a viable solution for this application. Although standard cameras are a viable solution for some underwater localization applications, our results show that the performance is highly dependent on the illumination. In the open-ocean pier environment where visibility conditions are constantly changing, we recommend the use of sensors that rely on range data instead of light data. These sensors include side-scan sonars, range cameras, or an array of single-point range sensors. These sensors yield higher feature counts and more accurate feature matches while eliminating visibility issues.

5.1 Future Work

Our work provides an overview of popular feature detection methods and how they compare when using different types of sensors. The following are future works that may extend this research and work towards a fully autonomous system:

Integration into SLAM

Feature detection is one part of the SLAM process. The motivation behind this research was to eventually apply the results into a SLAM algorithm. As we have evaluated the performance of different feature detection methods, it is important to see how well they improve the localization and mapping processes of SLAM. The first step in this direction may be to implement it in a simulated environment such as UWSim.

Testing with Real Sensors

As the sensor data in our research is simulated, our next step would be to conduct this research using real sensors with a more realistic computing platform. The non-simulated sensors may produce more noise as it is not predictable as the programmed data from the simulator. Also, all our processing was done using a desktop computer with a 6-core processor and a discrete graphics card, which an AUV is unlikely to have. This may make a difference in performance as some feature detection algorithms require more compute power than others.

AUV Integration

Our eventual goal is to have a SLAM system integrated within an AUV for autonomous localization and navigation. We are trying to create a SLAM algorithm robust enough to autonomously navigate through a pier environment without the intervention of a user. This requires a robust feature detection and matching algorithm as well as an accurate state estimation algorithm.

BIBLIOGRAPHY

- [1] J. Aulinas, M. Carreras, X. Llado, J. Salvi, R. Garcia, R. Prados, and Y. R. Petillot. Feature extraction for underwater visual slam. In *OCEANS 2011 IEEE - Spain*, pages 1–7, June 2011.
- [2] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In A. Leonardis, H. Bischof, and A. Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [3] C. M. Clark, C. S. Olstad, K. Buhagiar, and T. Gambin. Archaeology via underwater robots: Mapping and localization within maltese cistern systems. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 662–667, Dec 2008.
- [4] D. Cook, A. Vardy, and R. Lewis. A survey of auv and robot simulators for multi-vehicle operations. In *2014 IEEE/OES Autonomous Underwater Vehicles (AUV)*, pages 1–8, Oct 2014.
- [5] A. Dobke, J. Vasquez, L. Lieu, B. Chasnov, C. Clark, I. Dunn, Z. J. Wood, and T. Gambin. Towards three-dimensional underwater mapping without odometry. In *Unmanned Untethered Submersible Technology Conference 2013*, Aug 2013.
- [6] M. Guerrero. A Comparative Study of Three Image Matching Algorithms: Sift, Surf, and Fast. Master’s thesis, Utah State University, 2011.
- [7] J. Hartmann, J. H. Klssendorff, and E. Maehle. A comparison of feature descriptors for visual slam. In *2013 European Conference on Mobile Robots*, pages 56–61, Sept 2013.

- [8] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, Jan 2004.
- [9] P. Miller, J. Farrell, Y. Zhao, and V. Djapic. Autonomous underwater vehicle navigation. In *IEEE J. Ocean. Eng.*, volume 35, pages 663–678, Jul 2010.
- [10] R. D. P. Olvera, E. M. Zern, J. C. P. Ortega, J. M. R. Arreguin, and E. G. Hurtado. A feature extraction using sift with a preprocessing by adding clahe algorithm to enhance image histograms. In *2014 International Conference on Mechatronics, Electronics and Automotive Engineering*, pages 20–25, Nov 2014.
- [11] ORB (Oriented FAST and Rotated BRIEF).
https://docs.opencv.org/3.3.0/d1/d89/tutorial_py_orb.html, 2017.
- [12] Introduction to SIFT (Scale-Invariant Feature Transform).
https://docs.opencv.org/3.3.0/da/df5/tutorial_py_sift_intro.html, 2017.
- [13] Introduction to SURF (SpeededUp Robust Features).
https://docs.opencv.org/3.3.0/df/dd2/tutorial_py_surf_intro.html, 2017.
- [14] L. Paull, S. Saeedi, M. Seto, and H. Li. Auv navigation and localization: A review. *IEEE Journal of Oceanic Engineering*, 39(1):131–149, Jan 2014.
- [15] P. Ridao, M. Carreras, D. Ribas, and R. Garcia. Visual inspection of hydroelectric dams using an autonomous underwater vehicle. In *J. Field Robot.*, volume 27, pages 759–778, 2010.
- [16] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, Nov 2011.
- [17] I. T. Ruiz, S. de Raucourt, Y. Petillot, and D. Lane. Concurrent mapping and

- localization using sidescan sonar. In *IEEE J. Ocean. Eng.*, volume 29, pages 442–456, Apr 2004.
- [18] M. P. Schlachtman. Using monocular vision and image correlation to accomplish autonomous localization. Master’s thesis, California Polytechnic University, San Luis Obispo, June 2010.
- [19] UWSim Features. <http://www.irs.uji.es/uwsim/uwsim-features>.
- [20] Y. Wan. Performing contrast limited adaptive histogram equalization technique on combined color models for underwater image enhancement. In *International Journal of Interactive Digital Media*, volume 1, 2013.
- [21] L. Whitcomb, D. R. Yoerger, H. Singh, and J. Howland. Advances in underwater robot vehicles for deep ocean exploration: Navigation, control, and survey operations. In J. M. Hollerbach and D. E. Koditschek, editors, *Robotics Research*, pages 439–448, London, 2000. Springer London.
- [22] X. Yuan, J.-F. Martnez, M. Eckert, and L. Lpez-Santidrin. An improved otsu threshold segmentation method for underwater simultaneous localization and mapping-based navigation. *Sensors*, 16(7), 2016.