

OceanMappingDataframe - Scalable Multi-Indexed Dataframe for Hydrography

by

Vishwa Barathy Gandhi Kalidasan

Bachelor of Computer Science and Engineering, SRM Institute of Science and
Technology, 2019

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

Master Of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s): Ian Church, PhD, Geodesy and Geomatics Engineering
Suprio Ray, PhD, Computer Science
Examining Board: Daniel Rea, PhD, Computer Science, Chair
Bradford Nickerson, PhD, Computer Science
Andrew Gerber PhD, Mechanical Engineering

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

February, 2023

© Vishwa Barathy Gandhi Kalidasan, 2023

Abstract

Ocean data constitutes one of the largest geospatial datasets. Due to developments in the field of multibeam sonar, the amount of data gathered from hydrographic surveys is growing, causing the data to fall into the category of massive spatial data. Dataframe is a popular data model used to represent the data and is widely used in data science applications. Due to the lack of a suitable dataframe that can load large volumes of multibeam sonar data and support advanced analytics libraries, in this thesis, a new multi-indexed dataframe, OceanMappingDataframe, is introduced that can be used to load, store, and analyze multibeam sonar data. The multi-indexed dataframe was implemented using the MODIN dataframe library. The multi-indexed dataframe can load the hydrography files in Generic Sensor Format (GSF) or CSV file format, and save the results in partitioned Parquet files. The multi-indexed dataframe can also support advanced AI libraries such as OpenAI. This has been demonstrated by applying the Reinforcement Learning (RL) algorithm to an outlier detection problem in hydrography.

Dedication

To all the oceans, lakes, rivers and the beautiful Marina Beach, Chennai.

I would like to first offer my respects to Emperumanar for guiding me and giving me all the knowledge. I want to thank my parents and grandparents for their love and support. I am always thankful to my mom, who sacrificed her professional career to give me the best childhood.

I would like to thank my uncles Parthasarathy Commandur and Ramesh Comondoor and their families for being a pillar of support for me and helping me through many difficult moments in my life.

My thanks to Vinodh Mama, Prabhu Mama, VS Mama and Ramaa Aunty for all their love and support.

I would like to thank all my teachers, especially Niveditha S, Indumathy M, Amudha S and Dr. T. Vijayakumar, for their encouragement and support.

I thank my friends Hareesh and Anirudh for being incredible friends. I want to thank Madumitha for all the support. My thanks to Austin and Katerina for being a great support.

I would like to thank the team of Zoi Meet B.V - Nick Yap, Kevin Oranje, Kaarmuhilan Kalaiyarasan and Barsha Deka for giving me opportunity and encouragement during the early stages of my career.

I thank Alex and Chippins Limited for giving me a peaceful place to live and study. I want to thank Rick, Stephen and Midnight for their love and support.

I want to express my deep gratitude to Dr. Brandyn Chase and all the doctors, nurses and staff of Dr. Everett Chalmers Regional Hospital.

Finally, I would like to thank the people of Fredericton for all their love, kindness and support.

Acknowledgement

I would like to thank Dr. Ian Church and Dr. Suprio Ray for giving me an amazing opportunity to conduct research under their supervision. I am always thankful to them for teaching me all the concepts and providing me with endless opportunities and connections for my professional growth. I am also thankful to them for going beyond the professional level and for being a pillar of support during my challenging times.

I sincerely thank the University of New Brunswick and the Faculty of Computer Science for giving me an opportunity to pursue my graduate program. My deepest gratitude to the University and my supervisors for funding my project and giving me assistantship opportunities.

I want to thank ArcticNet for providing me with the datasets. I also thank Tony Furey for patiently helping me select the datasets for the experiments. Thanks to Eric Younkin for introducing and teaching me the concepts of Kluster.

I thank Burns Foster and Teledyne Geospatial, for giving me the opportunity to work on the CARIS Cloud Technology Project.

I would like to thank Youji Cheng, Sheila Paterson, and Mitacs, for giving me a chance to participate in the Lab2Market Oceans 2021 program.

I would also like to thank my supervisors and the University for giving me the opportunity to present at the CHC 2022 conference.

Thanks to Graham Christie, Madeline Claire Vainionpää, Ahmadreza Alleosfour, Mary Oyuky Chian Leal, Elizabeth Christie and all my friends at the Ocean Mapping Group and University for their constant support and encouragement.

I also want to thank Hillary Nguyen and Jason MacFarlane, International Student Advisors, for their enormous support and encouragement during my travel to Canada and for helping me during the program.

Table of Contents

| | |
|--|------------|
| Abstract | ii |
| Dedication | iii |
| Acknowledgments | v |
| Table of Contents | vii |
| List of Tables | xi |
| List of Figures | xii |
| Abbreviations | xiv |
| 1 Introduction | 1 |
| 1.1 Problem statement | 3 |
| 1.2 Contribution | 4 |
| 1.3 Organization of the thesis | 5 |
| 2 Background | 6 |
| 2.1 Overview | 6 |
| 2.2 Multibeam Sonar | 6 |
| 2.2.1 Multibeam Sonar Data | 9 |
| 2.2.2 GSF Files | 9 |
| 2.3 Multi-Index Dataframe | 12 |

| | | |
|----------|--|-----------|
| 2.4 | Reinforcement Learning | 13 |
| 2.4.1 | Outline of the Reinforcement Learning Components | 14 |
| 2.4.2 | Workflow of the Reinforcement Learning | 14 |
| 2.5 | Concluding Remarks | 15 |
| 3 | Related Work | 16 |
| 3.1 | Overview | 16 |
| 3.2 | Pangeo | 16 |
| 3.2.1 | Xarray | 18 |
| 3.2.2 | Zarr | 21 |
| 3.2.3 | Kluster | 21 |
| 3.3 | Outlier Detection in multibeam | 23 |
| 3.4 | Concluding Remarks | 25 |
| 4 | Methodology | 26 |
| 4.1 | Overview | 26 |
| 4.2 | OceanMappingDataframe | 27 |
| 4.3 | Out-of-Core Dataframes | 28 |
| 4.3.1 | Comparison between out-of-core dataframes | 29 |
| 4.4 | Multi-Index | 31 |
| 4.5 | APIs for OceanMappingDataframe | 34 |
| 4.6 | Loading GSF Files | 39 |
| 4.7 | Saving to Parquet Files | 41 |
| 4.7.1 | Evaluation Process to Select Parquet Format for OceanMappingDataframe | 42 |
| 4.8 | Reading Parquet Files | 48 |
| 4.9 | Reinforcement Learning | 50 |
| 4.9.1 | Outlier Identification Problem | 51 |

| | | |
|----------|--|------------|
| 4.9.2 | Swath Editor | 52 |
| 4.9.3 | Designing of the Reinforcement Learning | 53 |
| 4.10 | Concluding Remarks | 61 |
| 5 | Experimental Evaluation | 62 |
| 5.1 | Overview | 62 |
| 5.2 | Experiment Setup | 62 |
| 5.3 | Dataset Description | 63 |
| 5.4 | Evaluation | 64 |
| 5.4.1 | OceanMappingDataframe Vs Pandas | 64 |
| 5.4.2 | Comparison between Multi-Index and Single Index OceanMap- pingDataframe | 66 |
| 5.4.3 | Saving to Parquet Format | 67 |
| 5.4.4 | Loading Parquet Files | 68 |
| 5.4.5 | Select Function | 69 |
| 5.4.6 | Pivot Function | 71 |
| 5.4.7 | Reinforcement Learning | 72 |
| 5.4.7.1 | Performance of RL on Dataset 1 | 74 |
| 5.4.7.2 | Performance of RL on Dataset 2 | 78 |
| 5.4.7.3 | Performance of RL on Dataset 3 | 81 |
| 5.4.7.4 | Performance of RL on Dataset 4 | 84 |
| 5.4.7.5 | Performance of RL on Dataset 5 | 87 |
| 5.4.8 | Concluding Remarks | 91 |
| 6 | Conclusion and Future Work | 92 |
| 6.1 | Conclusion | 92 |
| 6.2 | Future Work | 93 |
| | References | 101 |

Vita

List of Tables

| | | |
|------|---|----|
| 2.1 | Variables of <code>gsfSwathBathyPing</code> | 11 |
| 4.1 | Comparisons between Parquet and Feather | 47 |
| 5.1 | Dataset description for <code>OceanMappingDataframe</code> | 63 |
| 5.2 | Number of Records Considered for each Dataset | 64 |
| 5.3 | Comparison of times (in seconds) to load hydrographic multibeam datasets into a Python dataframe | 65 |
| 5.4 | Time comparison (in seconds) for reading Single-Index and Multi- Index <code>OceanMappingDataframe</code> | 67 |
| 5.5 | Time to save (in seconds) to Parquet format using Single-Index and Multi-Index | 68 |
| 5.6 | Results of Loading Parquet Files | 69 |
| 5.7 | Comparison of the time (in seconds) to select a subset of half a dataframe using a single index and a multi-index. | 71 |
| 5.8 | Time (in seconds) to perform pivot operations on three different Ocean- MappingDataframe sizes. | 72 |
| 5.9 | Performance of RL on Dataset 1 Results | 75 |
| 5.10 | Performance of RL on Dataset 2 Results | 79 |
| 5.11 | Performance of RL on Dataset 3 Results | 82 |
| 5.12 | Performance of RL on Dataset 4 Results | 85 |
| 5.13 | Performance of RL on Dataset 5 Results | 88 |

List of Figures

| | | |
|------|---|----|
| 2.1 | Sample Representation of Multibeam Sonar Process [35] | 7 |
| 2.2 | Sample Representation of Multibeam Sonar Workflow [40] | 8 |
| 2.3 | Workflow of <code>gsflib</code> [25] | 10 |
| 2.4 | Single-Index and Multi-Index Dataframe | 13 |
| 2.5 | Workflow of Reinforcement Learning | 15 |
| 3.1 | Workflow of Pangeo System [19] | 17 |
| 3.2 | Sample of a Multidimensional Array | 19 |
| 3.3 | Naming Dimension using Xarrays | 19 |
| 3.4 | Xarray Coords | 20 |
| 3.5 | Structure of Kluster | 23 |
| 4.1 | Architecture of MODIN [3] | 30 |
| 4.2 | MODIN Vs Pandas Working Comparison [1] | 30 |
| 4.3 | Setting Multi-Index To OceanMappingDataframe | 32 |
| 4.4 | Data structure of OceanMappingDataframe | 33 |
| 4.5 | Workflow of <code>read_gsf</code> | 40 |
| 4.6 | An example illustrating use of <code>read_gsf</code> | 41 |
| 4.7 | Time taken to store and retrieve five different storage formats | 43 |
| 4.8 | Memory consumption comparison for different formats | 44 |
| 4.9 | Storage comparison for different formats | 45 |
| 4.10 | Sample example of <code>read_parquet</code> method | 49 |
| 4.11 | QPS Swath Editor top view and behind view [21] | 52 |

| | | |
|------|---|----|
| 4.12 | An example of depth table | 56 |
| 4.13 | View of the Depth Table | 56 |
| 4.14 | Status Flag Table | 58 |
| 4.15 | Control Flow of Reinforcement Learning | 60 |
| | | |
| 5.1 | Time to load GSF Files in OceanMappingDataframe vs Pandas . . . | 66 |
| 5.2 | Dataset 1 Results | 78 |
| 5.3 | Dataset 2 Results | 81 |
| 5.4 | Dataset 3 Results | 84 |
| 5.5 | Dataset 4 Results | 87 |
| 5.6 | Dataset 5 Results | 90 |

Abbreviations

| | |
|--------------|---|
| <i>A2C</i> | Advantage-Actor Critic |
| <i>AI</i> | Artificial Intelligence |
| <i>CNN</i> | Convolutional Neural Network |
| <i>CSV</i> | Comma Separated Values |
| <i>CUBE</i> | Combined Uncertainty and Bathymetry Estimator |
| <i>GSF</i> | Generic Sensor Format |
| <i>I/O</i> | Input/Output |
| <i>MBES</i> | Multibeam Sonar |
| <i>NOAA</i> | National Oceanographic and Atmospheric Administration |
| <i>OMD</i> | OceanMappingDataframe |
| <i>PPO</i> | Proximal Policy Optimization |
| <i>RL</i> | Reinforcement Learning |
| <i>SONAR</i> | Sound Navigation and Ranging |

Chapter 1

Introduction

Hydrography is the field of mapping the oceans to understand and describe seabed features [17]. Some of the main applications of seabed mapping are nautical hydrography, offshore engineering, fisheries habitat monitoring and environmental monitoring. With advancements in the seabed mapping domain, such as developments of newer survey methods, sensors and software, the amount of data collected from hydrography surveys is increasing rapidly [18].

There are several processes involved in seabed mapping operations. The primary process is to carry out the surveying operation with the use of a sensor such as sonar or LiDAR along with other GNSS sensors. After acquiring the data using sensors, the data is georeferenced with the water levels of the ocean and the position of the survey vehicle. The georeferenced data is later passed through the process of identifying noise and for quality checks before the final product is generated.

As the entire seabed mapping operation requires multiple processes, sensors, and software, it is both time-consuming and expensive. In recent years there has been an effort to reduce the time and expenses involved in the operation by developing autonomous vehicles to survey the oceans, streamlining the processing software to process larger amounts of data, and developing new filters to automate the identification of outliers [40]. With the introduction of newer standards like IHO S-44 [13], the final quality of the data is standardized, making it easier to understand the ocean compared to before these automation techniques were deployed.

A widely used data structure for representing and analyzing data is dataframes. The dataframes provided in the Python Pandas library [42] and R [46] are the most popular dataframes used by the data science community to build machine learning, deep learning and AI models. As there is a large quantity of quality hydrographic survey data available now, there is a need to analyze the data using the latest scalable data analytics techniques and store the large datasets in lighter formats.

Since the hydrography community lacks scalable dataframes that can support multi-beam sonar data formats and modern data science libraries, this thesis proposes an OceanMappingDataframe for loading, storing, and analyzing large volumes of multi-beam sonar data. The proposed dataframe uses concepts of multi-indexing for the users to conveniently locate the required part of the data from a large data repository, store the large data in partitions and work along with data science libraries.

1.1 Problem statement

In the field of hydrography, the data collected from multibeam sonar is increasing rapidly [40]. As a result, hydrographic offices worldwide receive large volumes of multibeam sonar data. Therefore, it has created a need in the hydrography community to consider storage options for large data. Also, as multiple processes are involved between the collection and production of the hydrographic products, it requires a unique system for the users to conveniently load multibeam sonar files, perform transformations, and run analytical queries on large multibeam data.

In addition to the requirements mentioned above, the hydrography community usually spends much time manually identifying outliers in the acquired data. As large volumes of data are being acquired, outlier identification becomes a bottleneck in the workflow. Therefore the hydrography community is exploring how to incorporate AI algorithms in the workflow to automatically identify the outliers in the data.

Many approaches have been proposed to load large volumes of Earth Science data using array-based structures, such as Xarray, Kluster [53], and to load and represent multibeam sonar data using Xarray. However, the data science community largely relies on structures of dataframes, rather than array-based structures to develop machine learning, deep learning and AI models.

There is no tight integration between array-based structures, such as Xarray, and popular data science and machine learning libraries like scikit-learn [44]. As dataframes such as Pandas [42] are popular among the data science community and have tight integration with all the popular data science and machine learning libraries, they cannot load and transform large volumes of multi-dimensional geospatial data, and in particular, they do not have the required loaders to read multibeam sonar formats. Therefore there is a need in the hydrography community to develop a dataframe structure that can support multibeam sonar readers to read the native multibeam sonar files, broadcast the multi-dimensional data and have integrations with data science libraries.

In this thesis, the following are the objectives:

1. Develop a scalable dataframe structure that is well integrated with multibeam sonar readers to load large volumes of multibeam sonar data.
2. Maintain multi-dimensional data in a multi-indexed structure.
3. Demonstrate integration with data science and AI libraries.

1.2 Contribution

This thesis proposes OceanMappingDataframe, a MODIN-based [45] multi-indexed dataframe, to support transformations and operations on large amounts of hydrography data. The proposed dataframe is one of the first approaches to use a MODIN based dataframe structure to load and analyze hydrography data. Along with the proposed OceanMappingDataframe, this thesis demonstrates that the proposed dataframe can be used with advanced AI libraries by developing a Reinforcement Learning algorithm to identify outliers in hydrography data.

1.3 Organization of the thesis

The rest of the thesis is organized as follows. First, the background is introduced in Chapter 2. Later, the related work is discussed in Chapter 3 and the problem statement is defined in Chapter 4. Next, the proposed approach is described in Chapter 5 and the experiment and evaluation of the proposed approach is explained in Chapter 6. Finally, in Chapter 7, the conclusion and future works are discussed.

Chapter 2

Background

2.1 Overview

This chapter discusses the fundamental concepts related to the current research work. It discusses in detail the background information regarding multibeam sonar, multibeam sonar data, and multi-index dataframes. Also, the principles of Reinforcement Learning and the workflow of Reinforcement Learning is discussed.

2.2 Multibeam Sonar

The predominant way to find the depth of the ocean is to use Sound Navigation and Ranging (SONAR) [35]. The process involves the sonar sending acoustic signals to the seabed, and the sonar calculating the time taken to receive the echo of the acoustic signals. The time taken between sending and receiving the acoustic signal can determine the depth of the water. Different types of sonar devices are used for different applications, but the predominant sonar used for seabed mapping is multibeam sonar (MBES). This thesis involves data observed using multibeam sonar.

The fundamental concept behind multibeam sonar is that a single transmission of the acoustic signal is transmitted across the seabed. The sonar listens for various angles of sound coming from a single transmission. The variety of sound coming from different angles form the beams of the sound.

A unique range can be measured for each beam. Since each beam has a known direction and range, a depth is calculated for each beam [35]. Figure 2.1 gives a sample representation of the multibeam sonar process.

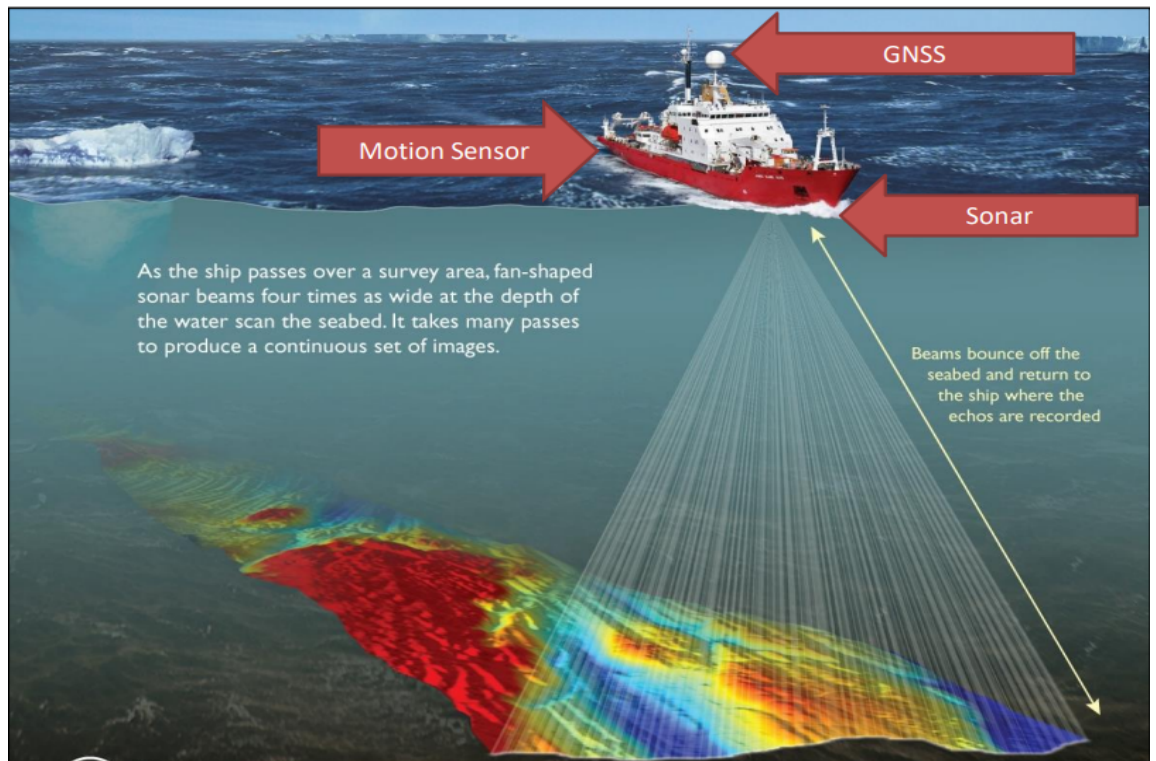


Figure 2.1: Sample Representation of Multibeam Sonar Process [35]

The raw depth data received from the sonar is later georeferenced to plot the received depths to a geographical position using the GNSS and motion sensor. The georeferenced data is later applied with sound velocity corrections and passed to identify errors in the data. As the sound returns from various directions, not all

the soundings return after hitting the seabed; some could return without hitting the seabed due to hitting a bubble, fish, seagrass or other environmental reasons causing errors in the data. Therefore, during the outlier detection process, the data operator needs to differentiate the soundings returned from the seabed and the error soundings. After cleaning, the data is passed to the quality check process, where the operators check if each area has a sufficient amount of valid sounding based on The International Hydrographic Organization to be later published for the final product. Figure 2.2 gives a sample representation of the multibeam sonar workflow [40].

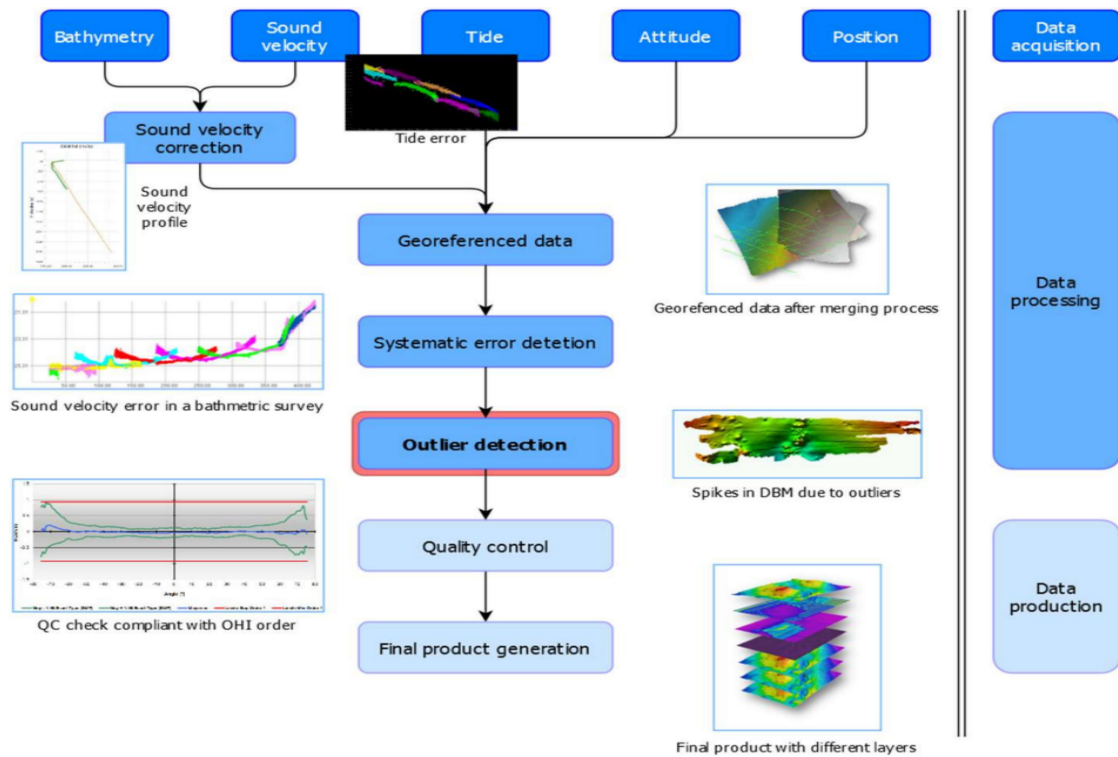


Figure 2.2: Sample Representation of Multibeam Sonar Workflow [40]

2.2.1 Multibeam Sonar Data

A multibeam sonar data is a record of millions of depths along with their georeferences (e.g., latitude, longitude), survey vehicle’s motion sensor references (e.g., roll, pitch, heave, heading), sound velocity profiles of the water and tidal information. All the different types of data are captured using different sensors and stored in specific formats. To read all the sensor data and process the data, specialized software is required, such as Teledyne Caris HIPS and SIPS [5] and QPS Qimera [20]. Users can collect, process, and publish multibeam sonar data using this specialized software.

2.2.2 GSF Files

One of the popular multibeam data formats is the Generic Sensor Format (GSF). The GSF format was developed by U.S. Naval Oceanographic Office in 1994 and is currently managed by Leidos. The GSF format is a binary format to store multibeam sonar data. To read GSF files, `gsflib` [25] is used. `gsflib` is a C-based library that uses functions such as `gsfopen` and `gsfread` to read GSF files. Figure 2.3 shows the workflow of the `gsflib` and the data structure of the GSF file. From Figure 2.3, the GSF data file is accessed using the `gsfOpen`, `gsfRead`, `gsfWrite`, and `gsfClose` functions.

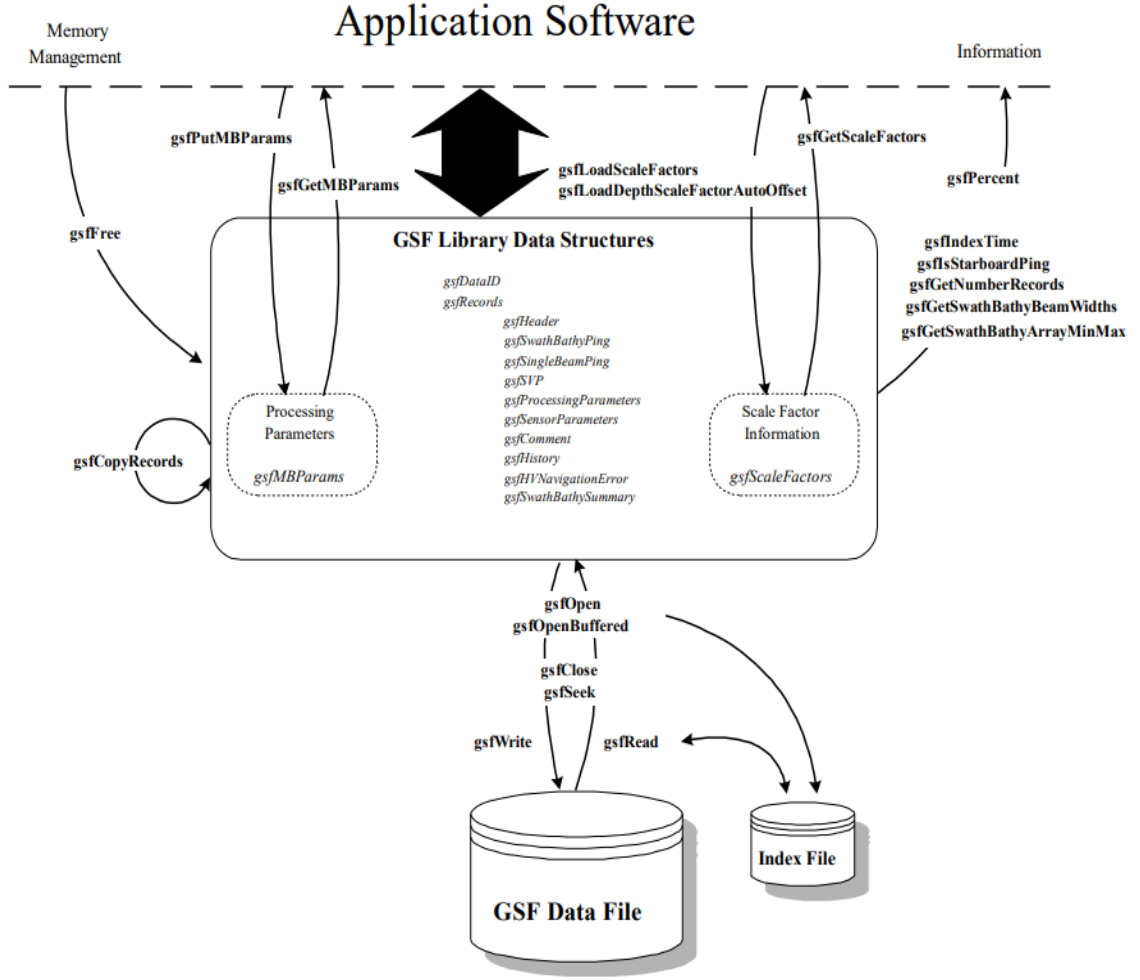


Figure 2.3: Workflow of `gsflib` [25]

After accessing the data file, `gsflib` creates a specific data structure to organize the data into datagrams/records. The `gsfSwathBathyPing` record holds the information of a ping, such as the depth and location values of all the beams in the ping and motion sensor values of that particular ping. Table 2.1 lists all the variables found inside the `gsfSwathBathyPing` record. By accessing the `gsfSwathBathyPing` record, the user can retrieve the information of a ping and its associated variables listed in Table 2.1.

Table 2.1: Variables of `gsfSwathBathyPing`

| | | | |
|-------------------|--------------------|--------------------|-------------------|
| ping_time | scaleFactors | latitude | speed |
| height | sep | number_beams | center_beam |
| reserved | tide_corrector | gps_tide_corrector | depth_corrector |
| heading | pitch | heave | roll |
| scaleFactors | depth | nominal_depth | across_track |
| travel_time | beam_angle | mc_amplitude | mr_amplitude |
| echo_width | quality_factor | receive_heave | depth_error |
| along_track_error | quality_flags | beam_flags | signal_to_noise |
| horizontal_error | sector_number | detection_info | system_cleaning |
| doppler_corr | sonar_vert_uncert | sonar_horz_uncert | detection_window |
| sensor_id | sensor_data | berb_en | incident_beam_adj |
| longitude | ping_flags | speed | along_track |
| vertical_error | across_track_error | beam_angle_forward | course |
| mean_abs_coeff | - | - | - |

2.3 Multi-Index Dataframe

Dataframe is the popular data structure used to represent and analyze data [45]. A dataframe is a tabular organization of the data having a certain number of cells, and each cell can store information. The information stored in the cell could be a number, string, date, boolean or geolocations. The way to access the table is to query the table using commands and retrieve the information. The key in the dataframe is the index of the dataframe. All the data in the dataframe are referenced to the index of the dataframe. A dataframe typically has the row number as the index. However, users can also create a multi-index dataframe by transforming the dataframe, setting specific columns as indexes and referencing all the data to indexes that are set. An example of a comparison between a single-index and multi-index dataframe is given in Figure 2.4. As all the soundings in the multibeam sonar data are referenced to a particular record and a beam in the particular record, one possible way to efficiently store the multibeam sonar data is by utilizing the multi-indexed dataframe with the record and beam number as the indexes.

| | col1 | col2 | col3 | | | col3 |
|---|------|------|------|------|------|------|
| | | | | col1 | col2 | |
| 0 | a | a | 1 | a | a | 1 |
| 1 | a | b | 2 | | b | 2 |
| 2 | a | c | 3 | | c | 3 |
| 3 | b | a | 4 | b | a | 4 |
| 4 | b | b | 5 | | b | 5 |
| 5 | b | c | 6 | | c | 6 |

Figure 2.4: Single-Index and Multi-Index Dataframe

2.4 Reinforcement Learning

Reinforcement learning is the branch of Artificial Intelligence (AI) where the AI Agent tries to solve a problem on its own by learning the problem and the solution from the rules of the problem. Reinforcement Learning has four components - Agent, Policies, Reward and Environment. A short description of the four components is given below.

- **Agent:** It is the AI-Agent, which will be solving the problem.
- **Action Space:** These are the possible actions an Agent can take at a particular given time.
- **Reward:** Based on an action the Agent takes, a reward is given based on how well that action leads to solving the problem.

- **Environment:** The Agent navigates to find a solution within the problem boundary.

2.4.1 Outline of the Reinforcement Learning Components

The environment is the problem domain in which the AI agent will work. The environment decides the action space and observation space for the AI agent. For example, in a game of chess, the environment is the chess game in which the AI agent will be working. The observation space is the arrangement of the chess board, and the action space is the possible set of moves that the AI agent can take for the particular arrangement. The user generates a reward depending on how well the AI agent took action to respond to the observation. In the game of chess, the reward could be the points the AI agent earned for taking a move.

2.4.2 Workflow of the Reinforcement Learning

The user first creates an environment based on the problem statement; thereby, the user develops the mechanism of generating the observation space and action space for the agent to take action for the observation. Finally, the user develops the mechanism to reward the AI-agent based on the action it performed automatically. As shown in Figure 2.5, the agent receives the observation space state S_t and, based on the observation space, takes an action A_t . Then, the environment generates the reward R_t and to the agent based on the action A_t . After which the next observation space S_{t+1} is given to the agent, and the agent takes an action A_{t+1} for which it receives reward R_{t+1} . The agent tries to find the patterns of actions that gives more rewards and thereby solve the problem.

As an example of Reinforcement Learning, in the game of chess, an AI agent starts by observing the current state of the chessboard, S_t . The agent then takes an action, A_t , resulting in a reward, R_t . The process repeats with the agent observing the updated state, S_{t+1} , and taking the next action, A_{t+1} , which yields the reward R_{t+1} .

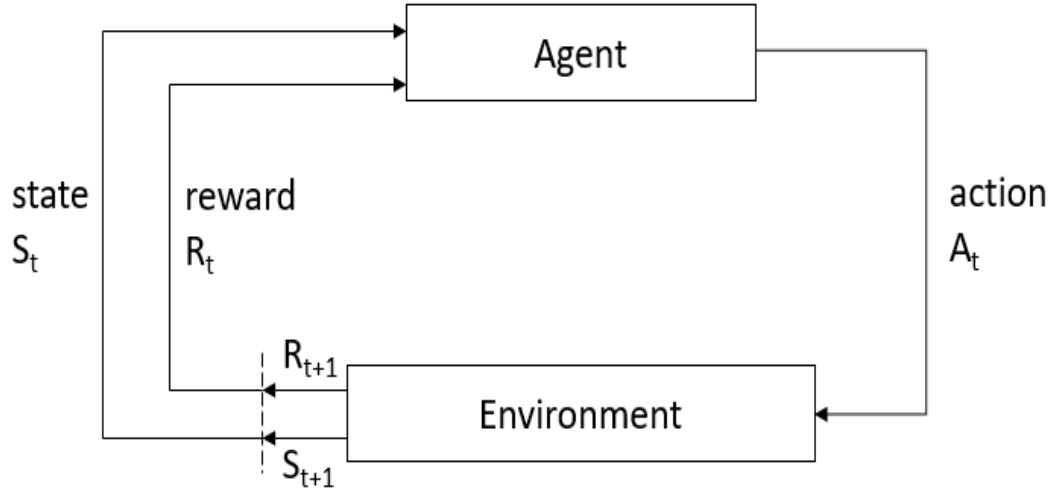


Figure 2.5: Workflow of Reinforcement Learning

2.5 Concluding Remarks

This chapter discussed the necessary background for the thesis, such as multibeam sonar data, multi-index dataframes and components of Reinforcement Learning. The next chapter discusses the related work in areas of loading large geospatial data and detecting outliers in hydrography data.

Chapter 3

Related Work

3.1 Overview

In the previous chapters, the thesis problem statement was defined and the fundamental concepts of the thesis were described. In this chapter, the existing systems used in the hydrography research community will be described.

3.2 Pangeo

The goal is to be able to work with a large volume of hydrography data. An ecosystem that is involved in the development of systems to accommodate a large amount of geoscience data is Pangeo. Pangeo is a community of geoscientists working collaboratively to develop open-source software and infrastructure to aid big data geoscience research [36].

The products developed by this community are software packages that can connect to the cloud and high-performance computing environments. The software packages that can aid large geoscience analytics in the cloud and high-performing computing using the Pangeo architecture are called Pangeo-based software.

The Pangeo environment was created when the amount of geo data, such as the ocean and atmospheric data, exceeded the storage of a single computer and required cloud and high-performance computing environments to store the data. This created a need for an infrastructure to easily access the data in the storage, perform high-speed transformations on the data and be user-friendly and interactive.

With these principles, the Pangeo system was developed by the community of geoscientists. Pangeo uses high-level data models such as Xarrays [26] and Pandas [42] to store the values of the data. To do faster data transformations, it uses distributed parallel computing with Dask. For the users to work interactively, it uses the Jupyter [39] environment. The workflow of the Pangeo system is given in Figure 3.1

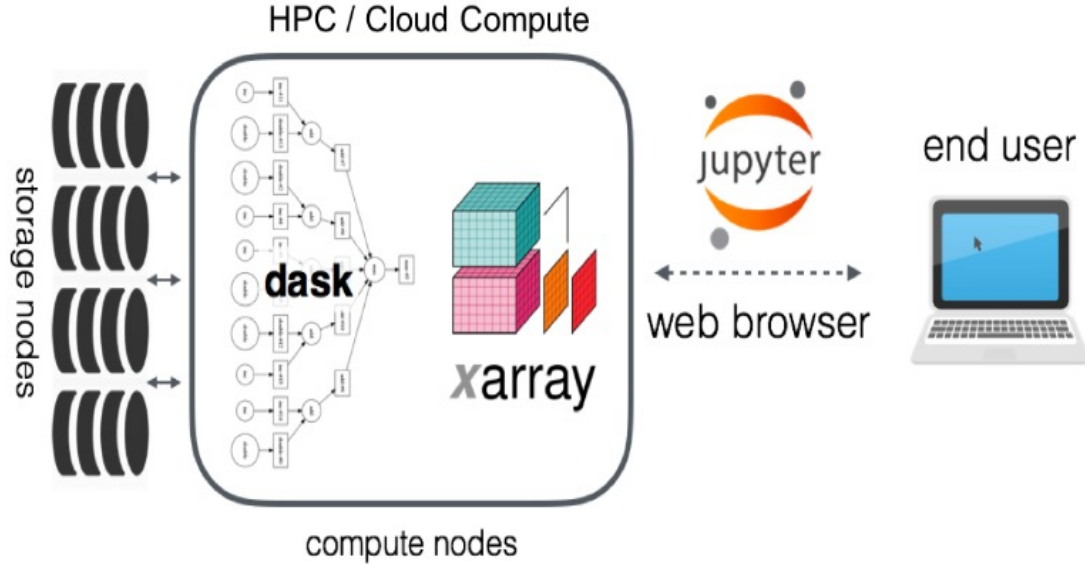


Figure 3.1: Workflow of Pangeo System [19]

Pangeo uses Python as its primary programming language to build its packages. The developers of Pangeo take geoscience data, modify the data to the data models such as Xarrays and use NumPy arrays alongside Dask to perform distributed parallel computing of the data. The packages in the environment store the data in popular big data formats such as HDF [11] and Zarr [27]. The next sections describes the Xarray and Zarr formats in detail. Xarray does not have tight integration with modern data science libraries. Although a very few attempts like sklearn-xarray [22] have been developed, it is difficult to develop machine learning and deep learning models using Xarray. Therefore Xarray is a popular data structure to load geoscience data but a less favourable data structure to be used along with data science libraries. Due to this, the thesis focuses on maintaining data in the form of a dataframe rather than the Xarray.

3.2.1 Xarray

Among the most popular data structures in Python language are the NumPy [37] arrays. The NumPy arrays are single-dimensional data structures to hold values of the data. The NumPy arrays can also be stacked upon each other to create multidimensional arrays. An example of a multidimensional array is shown in Figure 3.2. The sample of a multidimensional array has two dimensions to refer to and access a value.


```
1 a = np.random.random(size=16).reshape(4,4)
```

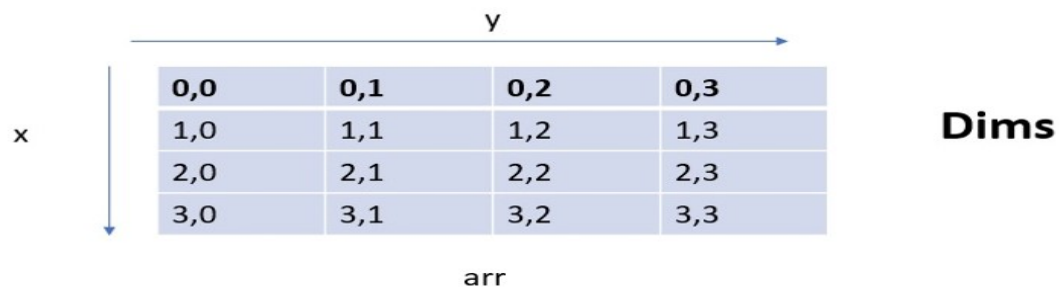
| 0,0 | 0,1 | 0,2 | 0,3 |
|-----|-----|-----|-----|
| 1,0 | 1,1 | 1,2 | 1,3 |
| 2,0 | 2,1 | 2,2 | 2,3 |
| 3,0 | 3,1 | 3,2 | 3,3 |

a

```
array([[0.23785177, 0.52362102, 0.19724696, 0.3938349 ],
       [0.93810275, 0.17367743, 0.29867663, 0.36085567],
       [0.45248046, 0.57387797, 0.26044881, 0.64713948],
       [0.36503703, 0.39224858, 0.27730985, 0.10025423]])
```

Figure 3.2: Sample of a Multidimensional Array

```
import xarray as xr
arr = xr.DataArray(a,dims=["x","y"])
```



```
1 arr
```

xarray.DataArray (x: 4, y: 4)

```
array([[0.23785177, 0.52362102, 0.19724696, 0.3938349 ],
       [0.93810275, 0.17367743, 0.29867663, 0.36085567],
       [0.45248046, 0.57387797, 0.26044881, 0.64713948],
       [0.36503703, 0.39224858, 0.27730985, 0.10025423]])
```

► Coordinates: (0)

► Attributes: (0)

Figure 3.3: Naming Dimension using Xarrays

However, it becomes difficult to remember each value's position and access it. To aid this process of accessing the data easily in large multidimensional arrays, the users can use the Xarray package [26] to name the dimensions of the multidimensional arrays. This is represented in Figure 3.3. Also, the users can change the default style of referencing the arrays to their custom needs using the Xarrays coords parameter. It is shown in Figure 3.4.

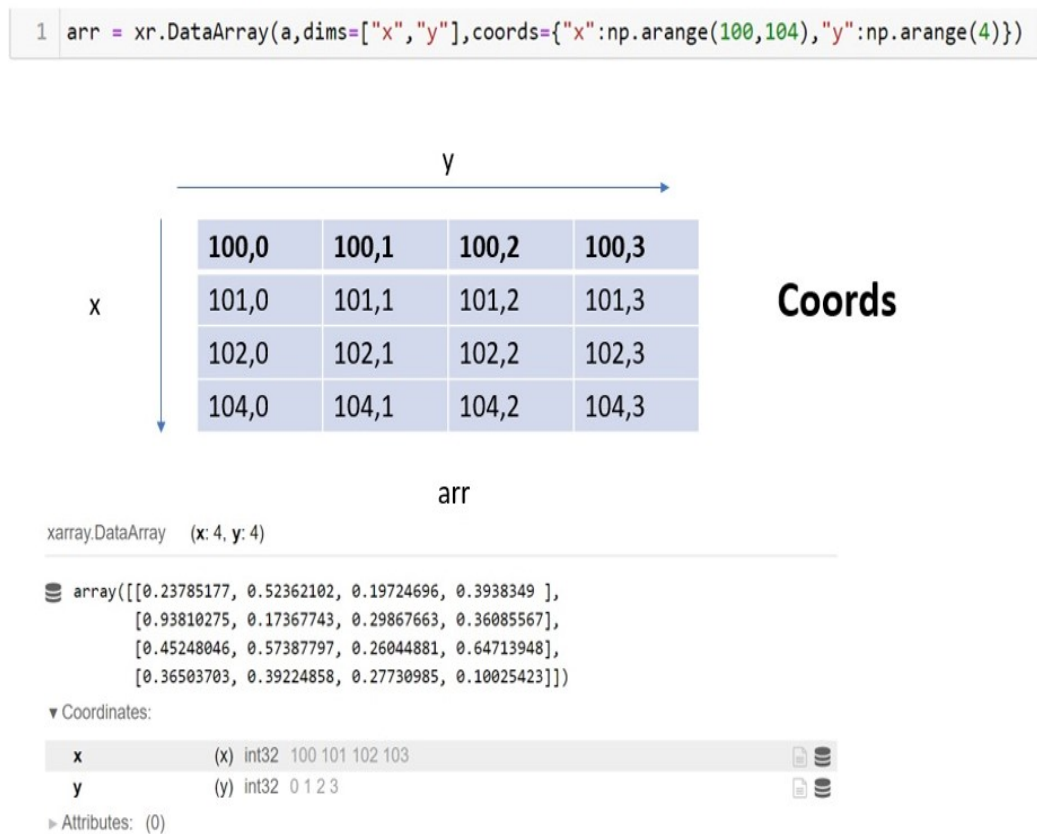


Figure 3.4: Xarray Coords

Once the user creates a multidimensional array, the user can stack the multidimensional array on top of each other and create an Xarray dataset [26]. Later, for accessing a value in the Xarray dataset, the user references the coords and dimen-

sions. Since the goal of the Xarray is to aid large geoscience data, the geoscience data can be easily created into a multidimensional array using the Xarray. For subsequent transformations, the chunks of the multidimensional array can be accessed and processed in distributed computing applications using Dask systems. Presently, Xarrays can load and store the data of netCDF [16], Zarr and NumPy binary format files.

3.2.2 Zarr

Zarr is a storage format developed to store large multidimensional arrays in cloud optimized format. Zarr [27] compresses the multidimensional arrays and stores the arrays as chunks. The user can retrieve the chunk of multidimensional arrays required and update the values in the chunk. The scalability to process large amounts of multidimensional arrays is provided by Dask. In a practical scenario, the users load the chunks of multidimensional arrays stored in Zarr format into the Xarrays dataset and use Dask for distributed computing.

3.2.3 Kluster

Kluster [53] is a multibeam hydrographic processing software developed by the National Oceanographic and Atmospheric Administration (NOAA) on the components of the Pangeo environment. Kluster is one of the first approaches to have used distributed computing principles in the hydrography domain [53].

Kluster reads the hydrography data stored in Kongsberg multibeam (.all) format and converts them into a multidimensional Xarray dataset. It uses the Zarr format to store and retrieve the data in a chunked manner. It uses the PROJ library for georeferencing, VisPy [24] for visualization and Jupyter for an interactive environment. The advantages of Kluster include referencing the data in the form of multidimen-

sional Xarrays and extending the processing to distributed computing principles.

Kluster reads the hydrography data and creates the coords for the Xarray dataset. It makes the **beam**, **time** and **xyz** as the coords for the Xarrays. Upon these coords, the other attributes are referenced. The attributes can be referenced to all coords or a few of the coords. The user performs transformations with the attributes using the Dask backend and stores the Xarray in the Zarr format. A sample structure of the Kluster is shown in Figure 3.5.

As shown in Figure 3.5, the Kluster reads multibeam sonar data and creates three coordinates **beam**, **time**, and **xyz**, upon which the other data variables are referenced. As Kluster extends from Xarray, the data variables can be broadcast to all the coordinates or a combination of the coordinates. For example, the data variable **acrosstrack** can be seen referenced to both beam and time coordinates. In contrast, the data variable **corr_altitude** can be seen referenced to only the time coordinate. Along with creating coordinates and broadcasting data variables to the coordinates, Kluster files can also contain remarks and information under the Attributes value.

Code: `print(dataset.multibeam.raw_ping)`

```
Out[4]: [<xarray.Dataset>
  Dimensions:                (beam: 432, time: 794, xyz: 3)
  Coordinates:
    * beam                    (beam) int32 0 1 2 3 4 5 6 ... 426 427 428 429 430 431
    * time                    (time) float64 1.376e+09 1.376e+09 ... 1.376e+09
    * xyz                     (xyz) <U1 'x' 'y' 'z'
  Data variables: (12/34)
    acrosstrack               (time, beam) float32 dask.array<chunksize=(794, 400), meta=np.ndarray>
    alongtrack                (time, beam) float32 dask.array<chunksize=(794, 400), meta=np.ndarray>
    altitude                  (time) float32 dask.array<chunksize=(794,), meta=np.ndarray>
    beampointingangle         (time, beam) float32 dask.array<chunksize=(794, 400), meta=np.ndarray>
    corr_altitude             (time) float32 dask.array<chunksize=(794,), meta=np.ndarray>
    corr_heave                (time) float32 dask.array<chunksize=(794,), meta=np.ndarray>
    ...
    tx                        (time, beam, xyz) float32 dask.array<chunksize=(794, 400, 3), meta=np.ndarray>
    txsector_beam             (time, beam) uint8 dask.array<chunksize=(794, 400), meta=np.ndarray>
    x                         (time, beam) float64 dask.array<chunksize=(794, 400), meta=np.ndarray>
    y                         (time, beam) float64 dask.array<chunksize=(794, 400), meta=np.ndarray>
    yawpitchstab              (time) <U2 dask.array<chunksize=(794,), meta=np.ndarray>
    z                         (time, beam) float32 dask.array<chunksize=(794, 400), meta=np.ndarray>
  Attributes: (12/42)
    _compute_beam_vectors_complete: Thu Feb 24 14:39:08 2022
    _compute_orientation_complete: Thu Feb 24 14:39:06 2022
    _conversion_complete: Thu Feb 24 14:38:56 2022
    _georeference_soundings_complete: Thu Feb 24 14:39:15 2022
    _sound_velocity_correct_complete: Thu Feb 24 14:39:12 2022
    _total_uncertainty_complete: Thu Feb 24 14:39:18 2022
    ...
    system_identifier: 103
    system_serial_number: [103]
    units: {'acrosstrack': 'meters (+ starboard)'}...
    vertical_crs: Unknown
    vertical_reference: waterline
    xyzrph: {'beam_opening_angle': {'1375591800': ...]}
```

Figure 3.5: Structure of Kluster

3.3 Outlier Detection in multibeam

There are many previous approaches proposed to identify outliers in the data. Based on [40], it is observed that the hydrography industry is motivated to create automatic tools to aid in the outlier identification process.

The authors in [40] have created a taxonomy to classify the outlier detection method. The paper points out that there are more algorithms developed using the unsupervised approach to detect the outliers and fewer algorithms developed using supervised learning algorithms.

The authors in [49] use a density-based method to identify outliers in multibeam data. They use an ensemble technique of local outlier factor algorithm and density-based spatial clustering of applications with noise (DBSCAN) algorithm to identify spatial outliers over a long period.

The authors in [52] use a binning technique along with an image processing technique to identify outliers. The authors first bin a region of multibeam data and classify the bin with high density to be the inlier. The other bins are classified using the image processing technique with respect to the inlier bin.

The authors in [30] use a distance-based approach to identify outliers. The authors attempt to use a triangulation technique to find the inliers and outliers. The approach is first to create edges between all the points using a triangulation method, and based on the specified threshold, the outliers are identified from the seabed.

In addition to the above-mentioned distance and density-based approaches to identify outliers, one of the most used techniques is a statistical-based approach. The most popular and commercially used algorithm is the Combined Uncertainty and Bathymetry Estimator, commonly called the CUBE algorithm [33].

The CUBE algorithm is used extensively in NOAA and the Canadian Hydrographic Services [40]. The CUBE algorithm estimates the potential seabed depth values based on an error model. The CUBE algorithm returns the confidence for its estimation along with the seabed depth values. The confidence of the estimation is based upon the three factors such as hypothesis count, hypothesis strength and hypothesis uncertainty [41]. This makes the user understand the different estimations of the seabed along with the confidence associated with them and makes a differentiation

between the inlier and outlier.

In recent years, a few deep learning-based approaches have been proposed in addition to the previous statistical approach. For example, Teledyne Caris developed a commercial software called Caris Mira AI [6] for identifying outliers using deep learning techniques. Caris Mira AI transforms the point cloud of the sonar data into voxels, and the voxels are sent to a Convolutional Neural Network (CNN). The CNN identifies and flags the voxels that are noisy and sends the flagged output to the user.

3.4 Concluding Remarks

This chapter discussed the related work currently developed in the thesis topics, such as Kluster, CUBE algorithm and Caris Mira AI. The next chapter discusses the proposed approach to develop a multi-index dataframe to support hydrography data operations and transformations. It also introduces the approach to identify outliers using a Reinforcement Learning algorithm.

Chapter 4

Methodology

4.1 Overview

This chapter discusses the methodology used to solve the problem statements mentioned in the previous chapter. In brief, the three objectives of the thesis are listed below.

The first two objectives are to develop a scalable dataframe system that can load large multibeam sonar files, perform transformation operations on the data and store the large data in convenient, lightweight formats. The third objective is to demonstrate that by using the dataframe, the users can conveniently build AI models for multibeam sonar data using popular AI frameworks and apply the models to identify outliers in the data.

4.2 OceanMappingDataframe

A popular data structure to load large geoscience data is the Xarray. An example of using the Xarray data structure to load multibeam sonar data is Kluster [53]. Although the Xarray data structure can be used to load and transform multidimensional geospatial data, the weak integration with data science libraries makes it less effective in building machine learning, deep learning and AI models. Popular dataframes such as Pandas support high integration with data science libraries, but they do not scale with large volumes of data and cannot be used to load large volumes of data [45].

To aid other data science and AI developments in the hydrography community, a novel dataframe structure is needed, which can handle large volumes of multibeam sonar data and have high integration with data science and AI libraries. This thesis introduces a dataframe exclusively for the needs of the hydrography community. The OceanMappingDataframe (OMD) is built on the principles of out-of-core data structures to handle large volumes of hydrography data and to process the data.

The proposed dataframe's objective is to be able to load multibeam sonar data directly into the dataframe structure, which is specifically indexed in accordance with the workflow to perform transformations on the loaded data. These concepts are described in upcoming sections.

Beyond the dataframe structure and ability to store the data in modern formats, the development of the dataframe was centered around the applications of the dataframe in the workflow. One of the observed issues in multibeam formats was that they were not compatible to work along with modern data science and AI libraries. Due to this, it is difficult to apply the libraries to the data and build modern data science models on hydrography data.

So, the OceanMappingDataframe is created to directly load multibeam sonar files in a dataframe structure and act as an interchangeable data structure to apply modern data science and AI libraries.

4.3 Out-of-Core Dataframes

As the first objective of OceanMappingDataframe is to be able to load the multibeam sonar data into a dataframe, it poses a challenge when the data is too large to fit into the available main memory of the computer. The out-of-core dataframes are used in the big data domain to accommodate larger datasets that do not fit into the main memory of the system. They use different techniques to accommodate the data into the main memory. A popular technique to accommodate large datasets in the main memory uses distributed computing or multi-threading principles, where the large data is broken down into smaller chunks, and the smaller chunks of data are distributed to all the cores or clusters of the CPU.

Many out-of-core dataframes are being developed. To develop the proposed OceanMappingDataframe, comparison between three popular out-of-core-dataframes, namely Vaex [31], Dask [48] and MODIN [45], were studied. In the following section, a comparison between the out-of-core dataframes is provided.

4.3.1 Comparison between out-of-core dataframes

The most popular out-of-core dataframe is the Dask dataframe. Dask uses distributed computing principles and distributes the data into small chunks into all cores of the main memory or cluster of CPUs. Vaex creates a memory map of the large dataset stored in the external hard disk or cloud storage and performs all the processing on the created memory map object. MODIN also uses distributed computing principles to distribute the data to all cores or clusters of CPUs.

As MODIN directly extends most of the APIs from Pandas [42], the popular data science dataframe library, and is compatible with popular machine learning libraries, the OceanMappingDataframe was extended from the MODIN dataframe structure. In addition, an advantage that the users get by using MODIN is the reusability of Pandas code. The user can change just one line of invoke statements of Pandas code (`import pandas as pd`) to invoke MODIN (`import modin.pandas as pd`) and use the existing code Pandas with MODIN. The developers of MODIN have kept the API syntax of MODIN the same as Pandas. Figure 4.1 represents the architecture of MODIN. MODIN developers have changed the working algebra of MODIN to accommodate larger datasets and optimize the dataframe. Figure 4.2 illustrates the working comparison between MODIN and Pandas.

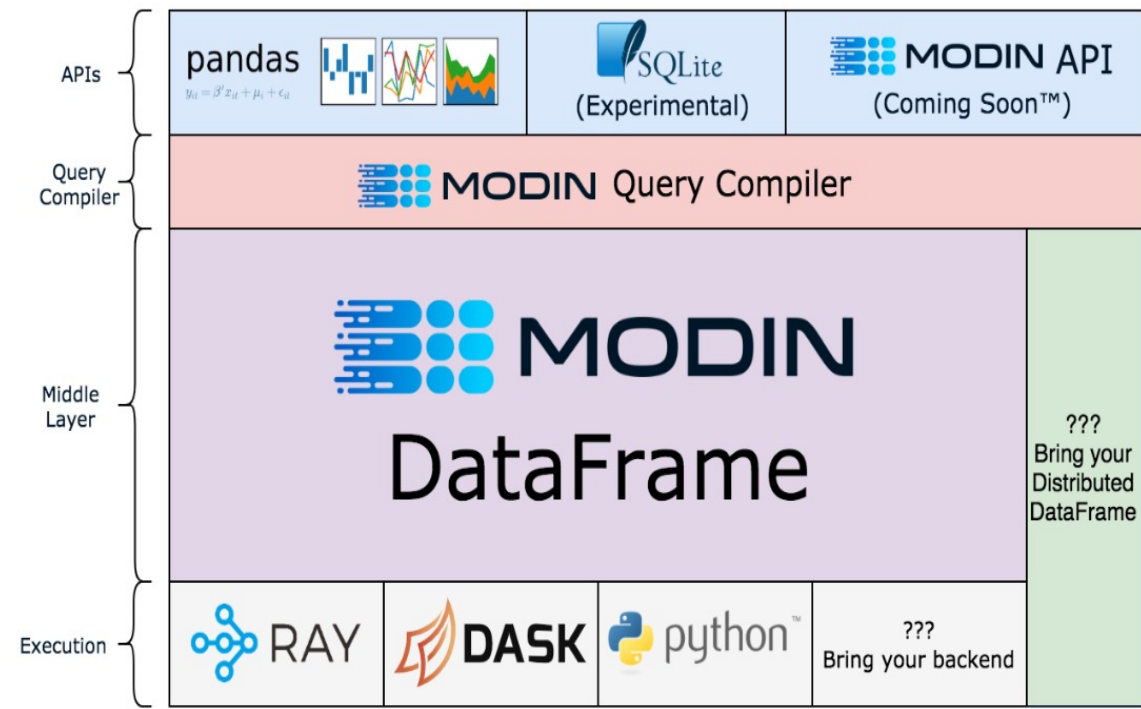


Figure 4.1: Architecture of MODIN [3]

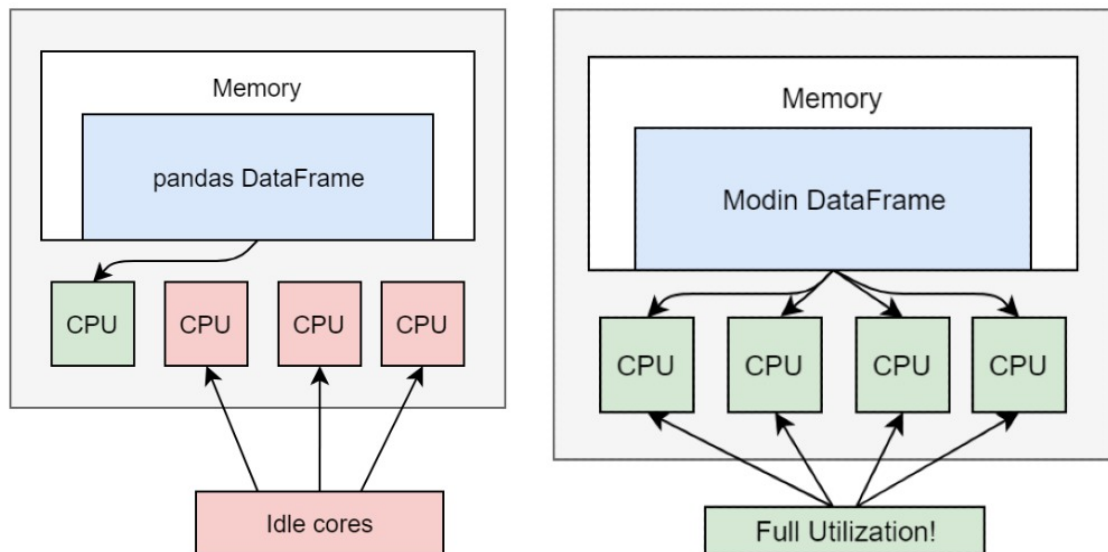


Figure 4.2: MODIN Vs Pandas Working Comparison [1]

4.4 Multi-Index

An important aspect of the dataframe is the index values of the dataframe. Any value in the dataframe can be located using the indexes of the dataframe. Both the column and rows of the dataframe can be identified by their index values. By default, the dataframes are single-indexed, but the dataframes can be multi-indexed by making a column value as the index and referencing all the values to the column value.

As OceanMappingDataframe extends the principle of the MODIN dataframe structure, it utilizes the concepts of MODIN's indexing principles. The indexes of a dataframe are row and column labels. A standard single-indexed dataframe has an array of a range of integers as row and column labels, starting from 0. Using these numeric indexes, the specific rows and columns can be retrieved. Using this representation, the values of the dataframe can be referenced to one row and column label. This makes it difficult to represent multi-dimensional data, where the values of the dataframe can be referenced to more than one row and column label. To represent multi-dimensional data, the MODIN dataframe allows the user to set multiple row labels and represent the data as a multi-indexed dataframe. Each row label is a one-dimensional array consisting of hashable data types such as int, float and string. The users can set multiple column values as row labels and represent the multi-dimensional data.

The OceanMappingDataframe, while constructing the dataframe by reading the streams of input multibeam data, first populates the dataframe with row labels as an array of integers starting from 0. After the dataframe is populated, it sets the record and beam number columns as the row labels. This creates a tuple of record and beam number values to be the row index, and with this multi-index row label,

all the values of the dataframe are referenced. This tuple of record and beam number values can be used to select the portion of the required data easily.

The process of setting multi-index to the row labels is shown in Figure 4.3. As shown in Figure 4.3, the transformation from single index to multi-index shows that in the multi-index dataframe, the depth value can be located using the tuple row label (record, beam number). For example, row label (0,1) represents the record as 0 and beam number as 1 and returns the depth value for the particular record and beam number as 5.

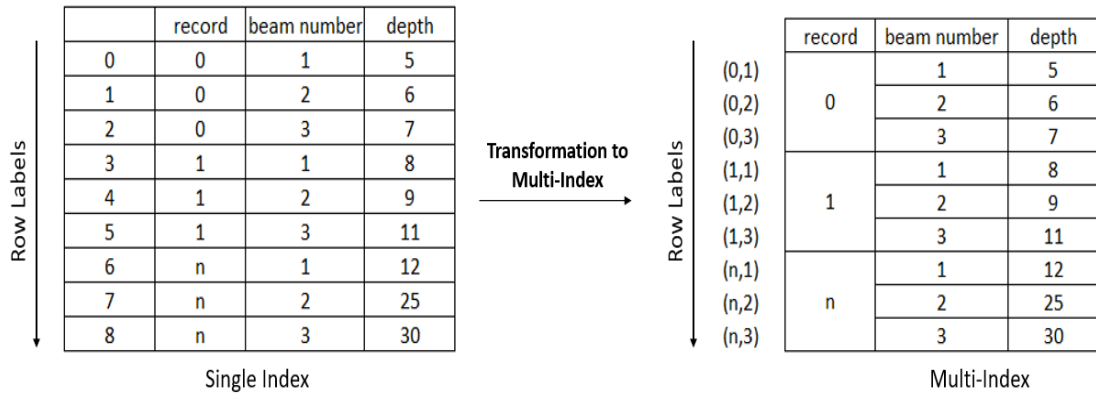


Figure 4.3: Setting Multi-Index To OceanMappingDataframe

The OceanMappingDataframe uses multi-index principles to reference the data values in the dataframe. The multi-indexes used in the OceanMappingDataframe are record and beam number. Record is the moment a ping of sound is released into the sea, and a ping of sound contains a certain number of beams, which hit the seabed and return to the sonar. The data of the multibeam sonar will have the record number, the depth of each beam in a record and a location for each beam. By indexing the dataframe to record and beam number, a user can easily go to a

particular record and query for a particular beam detail. The data structure of the OceanMappingDataframe is shown in Figure 4.4.

| Record | Beam Num | Col1 | Col2 | Col3 |
|------------------------------|----------------------------|------|------|------|
| 0 | 0 | | | |
| | 1 | | | |
| | 2 | | | |
| | ... | | | |
| | i = Maximum Beam Number | | | |
| n = Maximum Record Number | 0 | | | |
| | 1 | | | |
| | 2 | | | |
| | ... | | | |
| | i = Maximum Beam Number | | | |

Figure 4.4: Data structure of OceanMappingDataframe

As seen earlier, the primary reason Kluster chose Xarray for broadcasting the multi-beam sonar data was the ability to represent multidimensional data and to conveniently scale using Dask while loading a large amount of multibeam sonar data [53]. However, since Xarray lacks tight integrations with data science and AI libraries, it makes it less effective to use the Xarray data structure for developing data science and AI products for multibeam sonar data. The OceanMappingDataframe tries to overcome the issue in the Xarray data structure without compromising the ability to store and maintain large volumes of multidimensional data. To achieve this, OceanMappingDataframe uses multi-index principles to represent the multi-dimensional data with the indexes set to record and beam number and to achieve

scalability, OceanMappingDataframe uses MODIN as the backend. As the OceanMappingDataframe still maintains the dataframe structure required by data science and AI libraries, it is convenient for the hydrography community to represent large volumes of multidimensional data and apply data science and AI libraries to the data.

4.5 APIs for OceanMappingDataframe

As OceanMappingDataframe follows a dataframe structure and extends from the MODIN dataframe, OceanMappingDataframe has a list of APIs that allows the user to load, transform and store the multibeam sonar data. The following is the list of APIs that have been developed for OceanMappingDataframe.

I/O Operations

1. Read GSF

```
omd.read_gsf(file_path, set_index=True)
```

This API allows the users to read multibeam data stored in GSF files and transform the data into an OceanMappingDataframe structure.

Parameters:

file_path: str

Path to the file.

set_index: boolean, default 'True'

If set to 'True' returns a multi-indexed dataframe with row indices set to record and beamNum. If set to 'False' returns a single-indexed dataframe.

Returns:

An OceanMappingDataframe with labelled axes.

2. Read CSV

```
omd.read_csv(file_path, set_index=True, record, beamNum)
```

Using this API, the users can read multibeam data stored in CSV files and transform the data into an OceanMappingDataframe structure.

Parameters: `file_path`: str

Path to the file.

`set_index`: boolean, default 'True'

If set to 'True' returns a multi-indexed dataframe with row indices set to record and beamNum. If set to 'False' returns a single-indexed dataframe.

`record`: str

The column name in the CSV file that corresponds to the record feature.

`beamNum`: str

The column name in the CSV file that corresponds to the beam number feature

Returns:

An OceanMappingDataframe with labelled axes.

3. Read Parquet

```
omd.read_parquet(file_path,row_filter,column_filter)
```

Through this API, users can read portions or the entire multibeam data stored in Parquet files [51] and transform it into an OceanMappingDataframe. It can maintain the multi-index structure while loading if the data was stored using multi-index.

Parameter:

`file_path`: str Path to the file.

`row_filter`: list of tuples of predicates, default =None

Returns the list of rows that matches the predicates. Example `[('col1' = 'val1') ('col2' = 'val2')]`

`column_filter`: list of strings, default = None The dataframe is returned with only the columns present in the list. Example ['col1','col2']

Returns:

An OceanMappingDataframe with labelled axes.

4. Save To Parquet

```
omd.save(file_path, compression, partition_columns)
```

This API allows saving the OceanMappingDataframe as partitioned Parquet files based on the user's partition condition.

Parameters:

`file_path`: str

Path to the save the file.

`compression`: str, default='snappy'

The compression technique to use out of {'gzip', 'brotli', 'snappy' and None}

`partition_columns`: list of strings

The columns upon which the data gets partitioned while storing.

Returns:

A Binary Parquet File

Join and Groupby Operation

1. Join

```
omd.join(df1,df2, on, how)
```

This API allows the users to join two OceanMappingDataframes based on a condition.

Parameters:

df1: OceanMappingDataframe (single or multi-indexed)

The left OceanMappingDataframe upon which the join operation is to be executed.

df2: OceanMappingDataframe, MODIN or Pandas based dataframe (should have the same or partial indices of the left OceanMappingDataframe)

The right dataframe upon which the join operation is to be executed.

on: str, default indices

The key column(s) upon which the join operation will execute.

how: str, default 'left'

The type of join to be executed {'left', 'right', 'outer' and 'inner'}.

Returns:

OceanMappingDataframe with appropriate indexes

2. GroupBy

`omd.groupby(df, level)`

Using this API, the users can split the data into groups based on some conditions.

Parameters:

df: Multi-Indexed OceanMappingDataframe with row labels record and beamNum

level: str or list of string, default ['record', beamNum']

The index(es) upon which the groupby function should be executed

– {'record', 'beamNum', ['record', beamNum']}

Returns:

A GroupBy object that contains information about the groups

Analytics and AI operation

1. Correlate

```
omd.correlate(df, col1, col2)
```

Through this API, the users can check the correlation value between two columns in the OceanMappingDataframe.

Parameters:

df: OceanMappingDataframe (single or multi-indexed)

col1: str

The left column upon which the correlation function should take place

col2: str

The right column upon which the correlation function should take place

Returns:

A correlation score

2. Create Reinforcement

```
omd.create_reinforcement(df, records, timesteps, save_path)
```

This API allows the users to create a Reinforcement Learning model on selected data.

Parameter:

df: Multi-Indexed OceanMappingDataframe with row labels record and beam-Num

records: list of integers

The specific records upon which the Reinforcement Learning algorithm should train. Example [100,101,180] or range(0,80)

timesteps: int, default 500000

The total number of timesteps upon which the agent should be trained.

save_path: str

The folder in which the should be saved

Returns: A Reinforcement Learning model

3. Apply Reinforcement

```
omd.apply_reinforcement( model_path, df, records)
```

Using this API, the users can load a pre-trained Reinforcement Learning model and apply the model to selected data.

Parameter:

model_path:str

The path to the Reinforcement Learning model

df: Multi-Indexed OceanMappingDataframe with row labels record and beam-Num

records: list of integers

The specific records upon which the Reinforcement Learning algorithm should be applied. Example [100,101,180] or range(0,80)

Returns:

The model prediction as a MODIN Series

4.6 Loading GSF Files

The multibeam sonar data is stored in exclusive formats such as Generic Sensor Format (GSF) [8] and Kongsberg .all format [2]. To read these files, the user needs specific multibeam data format readers. For this thesis, the GSF files were loaded using the pygsf open-sourced GSF reader into the OceanMappingDataframe.

To aid in the process of reading the native GSF files, Guardian Geomatics open-sourced their pygsf reader [15]. The pygsf reader can take a .gsf file and output the information as NumPy arrays [37]. As multiple sensor information is stored in a .gsf file, the main information about the depth, motion sensor and GNSS are returned under the SWATH_BATHYMETRY class by the pygsf reader. By invoking the SWATH_BATHYMETRY class, the NumPy arrays of the information associated with a record (ping of a sound) are retrieved.

In the implementation, pygsf reader and its SWATH_BATHYMETRY class were used to return the information stored in the .gsf files as NumPy arrays. The NumPy arrays are collected by the OceanMappingDataframe’s read_gsf method and organized the arrays to a dataframe. The dataframe is later multi-indexed to the record and beam number levels by the read_gsf method. The workflow of the read_gsf method is shown in Figure 4.5.

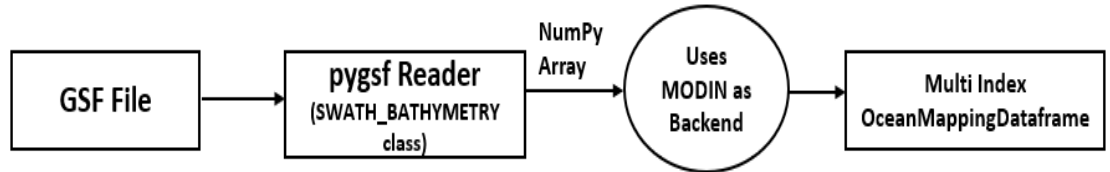


Figure 4.5: Workflow of read_gsf

By invoking the read_gsf API call, the user can easily read a native .gsf file to a dataframe indexed to record and beam number. A sample example of the result of the read_gsf method is given in Figure 4.6.

Code:

```
import oceanmappingdataframe as omd

data = omd.read_gsf( 'sample.gsf', set_index=True)

print(data)
```

| | | time | roll | pitch | heave | lat | lon | depths | across_track | along_track | travel_t |
|--------|---------|--------------|-------|-------|-------|-----------|-----------|--------|--------------|-------------|----------|
| record | beamNum | | | | | | | | | | |
| 0 | 0 | 1.406626e+09 | -1.30 | 0.24 | -0.02 | 50.360541 | -4.158959 | -3.943 | 0.05 | 1.10 | 0.00 |
| | 1 | 1.406626e+09 | -1.30 | 0.24 | -0.02 | 50.360541 | -4.158959 | 9.509 | -21.73 | 3.02 | 0.03 |
| | 2 | 1.406626e+09 | -1.30 | 0.24 | -0.02 | 50.360541 | -4.158959 | -3.943 | 0.05 | 1.10 | 0.00 |
| | 3 | 1.406626e+09 | -1.30 | 0.24 | -0.02 | 50.360541 | -4.158959 | -3.943 | 0.05 | 1.10 | 0.00 |
| | 4 | 1.406626e+09 | -1.30 | 0.24 | -0.02 | 50.360541 | -4.158959 | -3.943 | 0.05 | 1.10 | 0.00 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 3722 | 507 | 1.406626e+09 | -1.52 | 0.00 | 0.01 | 50.356693 | -4.162985 | 7.796 | 21.08 | -0.31 | 0.03 |
| | 508 | 1.406626e+09 | -1.52 | 0.00 | 0.01 | 50.356693 | -4.162985 | 7.797 | 21.17 | -0.32 | 0.03 |
| | 509 | 1.406626e+09 | -1.52 | 0.00 | 0.01 | 50.356693 | -4.162985 | 7.790 | 21.23 | -0.32 | 0.03 |
| | 510 | 1.406626e+09 | -1.52 | 0.00 | 0.01 | 50.356693 | -4.162985 | 7.794 | 21.33 | -0.33 | 0.03 |
| | 511 | 1.406626e+09 | -1.52 | 0.00 | 0.01 | 50.356693 | -4.162985 | 7.792 | 21.41 | -0.33 | 0.03 |

1906176 rows × 13 columns

Figure 4.6: An example illustrating use of read_gsf

4.7 Saving to Parquet Files

Upon completing the analysis and transformations of the loaded multibeam data, the OceanMappingDataframe can be saved in a Parquet [51] format. Parquet is a file format designed to store complex and large data. The Parquet format organizes the files in a column-wise manner. It uses the principles of compressing the data in a column-wise manner.

4.7.1 Evaluation Process to Select Parquet Format for OceanMappingDataframe

Before the Parquet was selected as the data storage format, popular big data storage formats such as CSV [47], feather [7], HDF [11], and pickle [50] were selected and were evaluated. These file formats were tested to check the storage consumption, time taken for read/write operation and consumption of main memory during read/write operation.

For this experiment, one million rows were inserted into a Pandas dataframe with 15 numerical columns and 15 categorical columns. The 15 categorical columns were considered strings during the test. The Pandas dataframe was stored in CSV, feather, Parquet, HDF and pickle formats. The data stored in different formats were again loaded into a Pandas dataframe. From this process, the storage consumption, time taken for read/write operation and consumption of main memory during read/write operation were identified.

The first experiment was to identify the time taken by different formats to save and load the Pandas dataframe. The experiment results are given in Figure 4.7, where it was observed that the CSV takes the maximum time to perform the save and load operations. The HDF and pickle perform better than the CSV format. In comparison, the Parquet and feather formats took the least amount of time to perform both operations.

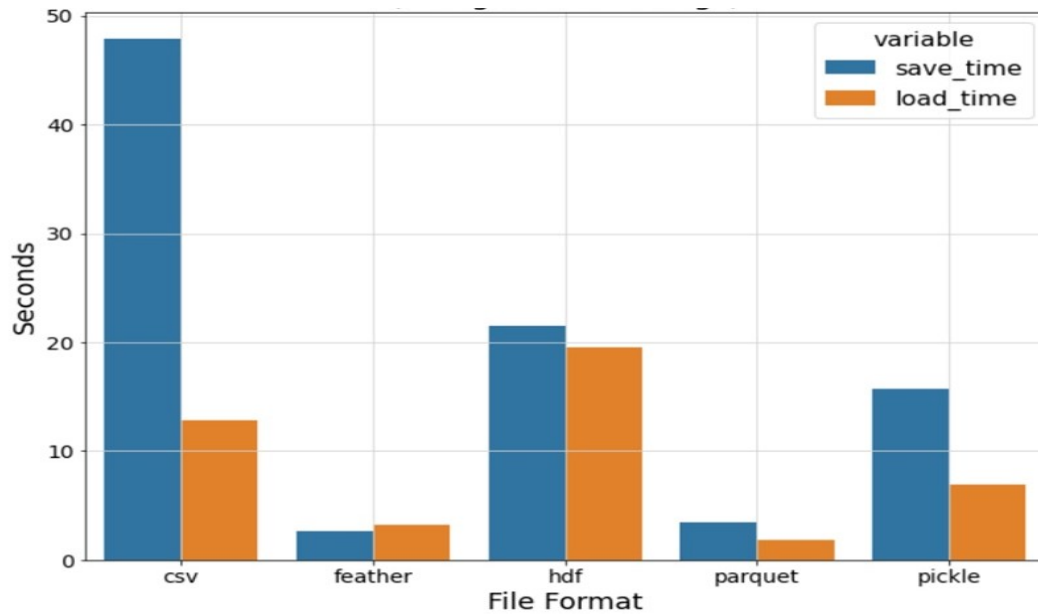


Figure 4.7: Time taken to store and retrieve five different storage formats

The second experiment was to identify the main memory consumption by different formats to save and load the Pandas dataframe. The results of the experiment are given in Figure 4.8. It was observed that Parquet and feather consumed a reasonable amount of main memory as compared to HDF and pickle. In contrast, CSV consumed the least amount of main memory while performing both operations.

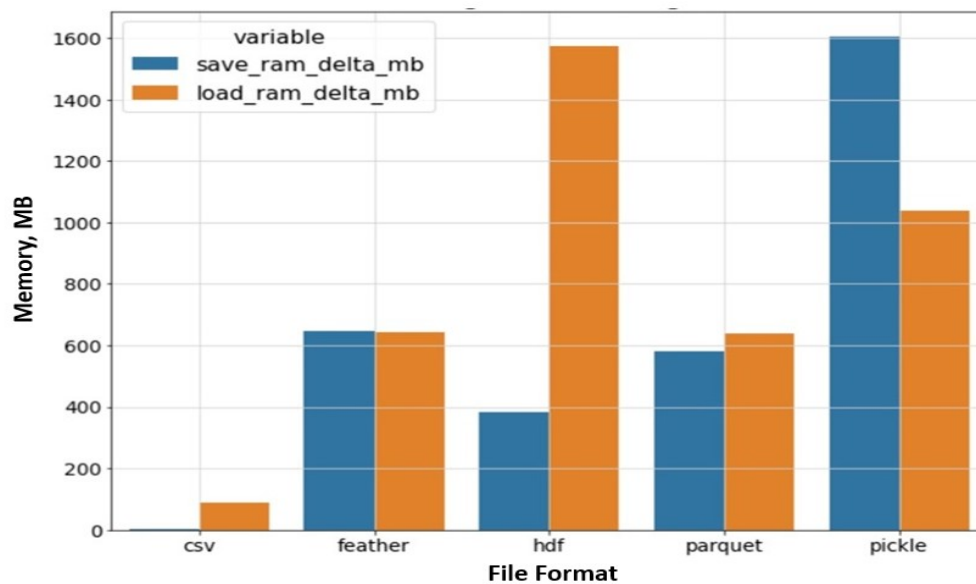


Figure 4.8: Memory consumption comparison for different formats

The third experiment was to identify the storage consumption used by different formats while saving the Pandas dataframe. The results of the experiment are given in Figure 4.9. From the results, it was observed that the CSV consumed the maximum storage, followed by HDF and pickle. Parquet and feather consumed the least amount of storage space.

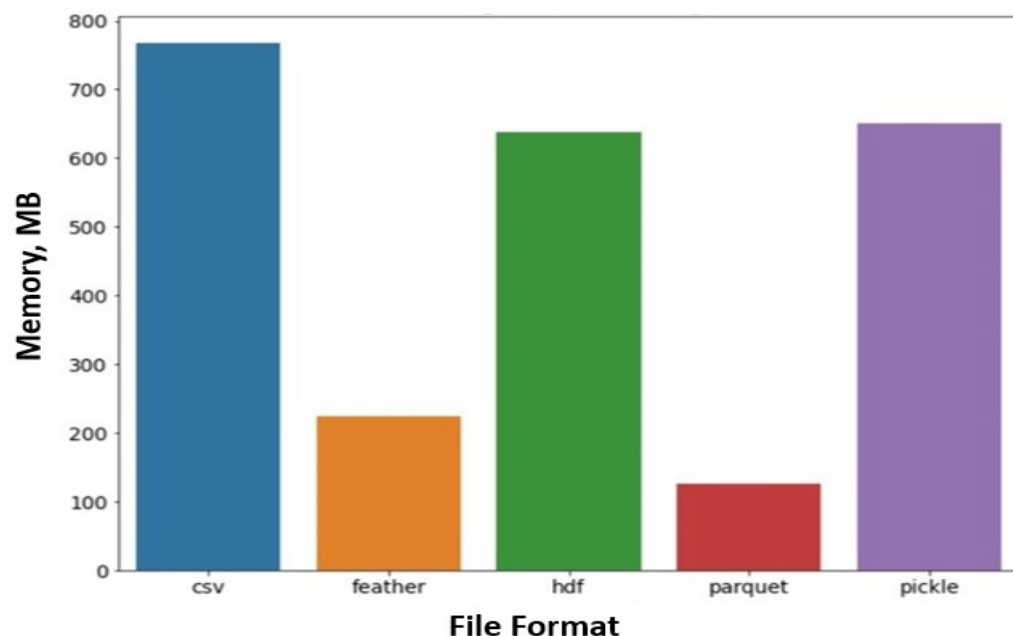


Figure 4.9: Storage comparison for different formats

After conducting the experiments, the main parameters focused on were the storage consumption and time consumption to perform read/write operations by the selected formats. The CSV format consumes the maximum storage and time consumption compared to the other formats. Whereas trying to store the data in HDF and pickle format gives a marginal reduction in storage and time consumption compared to CSV format, they still consume a lot of storage and time to perform read/write operations compared to feather and Parquet.

From these experiments, it was obvious to eliminate CSV, HDF and pickle. Another experiment was conducted to understand the performance of feather and Parquet more clearly. In this experiment, spatial values were added to the data. For this test the GeoPandas dataframe [38] was used. GeoPandas is a Pandas based library to accommodate spatial objects such as points, lines and polygons in the dataframe.

In the second experiment, one million rows were inserted into a GeoPandas dataframe with 15 numerical columns, 15 categorical columns and one spatial column that consisted of points. Similar to the previous experiment, 15 categorical columns were considered string objects. First, the GeoPandas dataframe was stored in feather and Parquet formats. The data stored in both formats were again loaded into a GeoPandas dataframe. From this test, the storage consumption, time taken for read/write operation and consumption of main memory during the read/write operation of feather and Parquet were identified.

For the experiment, all the available compression techniques of feather and Parquet were considered. The results of the experiments are given in Table 4.1. It was observed that both the formats consumed comparable time consumption to perform both the save and load operations. Parquet consumed lesser storage space than feather in all the compression techniques and has been highlighted in the table.

Table 4.1: Comparisons between Parquet and Feather

| | Save | | | Load | |
|----------------------------|---------|-------------------------|---------------|---------|-------------------------|
| | Time(s) | Memory Consumption (MB) | Size(MB) | Time(s) | Memory Consumption (MB) |
| Feather No Compression | 35.43 | 636.00 | 704.00 | 15.88 | 1157.75 |
| Feather lz4 Compression | 33.43 | 753.02 | 232.00 | 15.68 | 1040.95 |
| Feather zstd Compression | 34.00 | 660.32 | 174.50 | 15.97 | 987.48 |
| | | | | | |
| Parquet No Compression | 37.43 | 625.46 | 149.50 | 14.17 | 1097.85 |
| Parquet Snappy Compression | 37.68 | 644.07 | 131.50 | 14.60 | 468.84 |
| Parquet Gzip Compression | 45.77 | 645.31 | 121.50 | 17.17 | 539.35 |
| Parquet Brotli Compression | 40.57 | 645.01 | 120.10 | 18.65 | 559.68 |

From this experiment, it was identified that Parquet performed better than feather when spatial objects were involved. Since multibeam sonar data also consists of spatial objects, Parquet was more suitable as a storage format for OceanMapping-Dataframe.

In addition to storing the multibeam sonar data in Parquet format, OceanMapping-dataframe allows users to partition the data into files using the column value. By partitioning the data, a user can retrieve just the portions of data they need, rather than the entire dataset. The partitioning of the files is extended from the Parquet storage format system. Parquet allows the user to specify the column upon which the partitioning should occur. While using the OceanMappingDataframe, the save method can be used to save the dataframe in Parquet files and specify the column upon which the partition should take place by mentioning the list of column names in the partition_columns parameter. An example Python script of the the save method is given below.

Code:

```
import oceanmappingdataframe as omd

omd.save(data, 'gsf_data_api_test_set_index.parquet', partition_columns
= ['beamNum'])
```

4.8 Reading Parquet Files

After storing the hydrography data in Parquet format, users can use the OceanMappingDataframe to read the files into the dataframe. As mentioned earlier, one of the key aspects of OceanMappingDataframe is chunk-based retrieval. As the data volume is expected to be large, a user can specify the required chunk of data, and the OceanMappingDataframe with MODIN's back-end can fetch the required portion of data from Parquet partitions in a distributed manner. Also, if the data is stored in a multi-index structure, the OceanMappingDataframe can maintain the same structure while loading the data again.

OceanMappingDataframe offers two levels of filtering; the user can filter the data by row and column. To filter the data, the user should specify the conditions in a tuple format and pass the list of tuples as parameters to the `read_parquet()` API of the OceanMappingDataframe. An example of `read_parquet` is shown in Figure 4.10 along with the Python scripts.

Code:

```
import oceanmappingdataframe as omd

dataset = omd.read_parquet('gsf_data_api_test_set_index.parquet',
row_filter=[('beamNum', '=', 256)]) print(dataset)
```

| | | time | roll | pitch | heave | lat | lon | depths | across_track | along_track |
|--------|---------|--------------|-------|-------|-------|-----------|-----------|--------|--------------|-------------|
| record | beamNum | | | | | | | | | |
| 1792 | 256 | 1.406626e+09 | -1.61 | -0.14 | 0.02 | 50.358561 | -4.160854 | 11.622 | 0.51 | 1.2 |
| 1793 | 256 | 1.406626e+09 | -1.57 | -0.13 | 0.02 | 50.358560 | -4.160855 | 11.702 | 0.51 | 1.2 |
| 1794 | 256 | 1.406626e+09 | -1.51 | -0.12 | 0.02 | 50.358559 | -4.160856 | 11.744 | 0.51 | 1.2 |
| 1795 | 256 | 1.406626e+09 | -1.44 | -0.12 | 0.02 | 50.358558 | -4.160857 | 11.644 | 0.50 | 1.2 |
| 1796 | 256 | 1.406626e+09 | -1.36 | -0.13 | 0.02 | 50.358556 | -4.160858 | 11.557 | 0.50 | 1.2 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 2299 | 256 | 1.406626e+09 | -1.47 | 0.03 | 0.00 | 50.357870 | -4.161389 | 9.518 | 0.45 | 1.2 |
| 2300 | 256 | 1.406626e+09 | -1.40 | 0.02 | 0.00 | 50.357869 | -4.161390 | 9.495 | 0.45 | 1.2 |
| 2301 | 256 | 1.406626e+09 | -1.34 | 0.01 | 0.00 | 50.357868 | -4.161391 | 9.507 | 0.44 | 1.2 |
| 2302 | 256 | 1.406626e+09 | -1.27 | -0.02 | 0.00 | 50.357867 | -4.161392 | 9.513 | 0.44 | 1.2 |

Figure 4.10: Sample example of `read_parquet` method

4.9 Reinforcement Learning

In the previous sections, the data structure of OceanMappingDataframe was discussed. The OceanMappingDataframe’s purpose, in addition to loading large amounts of hydrography data, was to bridge the developments of artificial intelligence (AI) into the world of hydrography. As mentioned earlier, the main barrier to developing the AI models and products for hydrography data was the incompatibility of the data format with the AI development frameworks.

To add further, popular deep learning frameworks such as Tensorflow [29], Keras [34] and Pytorch [43] accept images, audio and text formats and convert them into NumPy arrays to send to deep neural network architectures for training deep learning models. However, they cannot directly accept hydrography data as the AI development frameworks cannot parse it and convert it into the format they can process.

By creating the OceanMappingDataframe, an intermediate data structure is created that can parse the hydrography data and hold the data in a structure that can be accepted by the AI development framework. This stage is achieved because OceanMappingDataframe uses a MODIN based backend and stores all the information as MODIN objects.

Since MODIN supports APIs similar to Pandas dataframe and all the popular AI frameworks accept Pandas framework. By processing the hydrography data in OceanMappingDataframe structure, the user can connect to all the popular data science libraries such as sklearn [44] and deep learning frameworks like Tensorflow, Keras and Pytorch.

By using the OceanMappingDataframe, the users can develop their models in any popular Python environments such as Jupyter Notebook [39], and Google Colab [10]. Moreover, users can leverage cloud-based environments such as AWS [4] and Google Cloud Platform [9] for developing the AI models using OceanMappingDataframe.

The next section explains how to use OceanMappingDataframe to solve a hydrography problem with Reinforcement Learning.

4.9.1 Outlier Identification Problem

As discussed earlier in the thesis, an important step in the hydrography workflow is outlier detection in multibeam data. Since echo sounding is used to map the seabed, and due to the nature of the sound and environment, the sonar sometimes captures false return signals from the seabed. As the hydrography community usually spends $2\times$ more of their working hours cleaning the files manually than acquiring the data [40], it creates a bottleneck for delivering the final results.

As mentioned in the related work sections, various attempts were developed to use automation and filtering techniques to identify outliers in the multibeam data. However, only a few approaches have used AI techniques such as, deep learning to identify outliers in multibeam data. In this thesis, a demonstration shows how to use the proposed OceanMappingDataframe to build a Reinforcement Learning-based approach to solve the outlier detection problem in multibeam data. The demonstration of the tight integration of the proposed OceanMappingDataframe and AI libraries provides a basis for further research in hydrography using AI and data science. In this thesis, the Reinforcement Learning approach is developed for one of the problem statements in hydrography, but by using the proposed OceanMappingDataframe, the users can build data science and AI models for the other problem statements.

4.9.2 Swath Editor

The swath editor [21] is one of the primary ways the hydrography community used to clean the multibeam data. The swath editor removes the georeferencing properties of the depth soundings. The user can choose a certain number of records of the multibeam data, and the swath editor plots the depth against the beam number for the selected amount of data. The user can then flag the outliers present in the selected region of the data.

By plotting the depths against the beam number, the main advantage the swath editor gives the user is to see the feature of the seabed from the behind view, as the user cannot see any features from the top view. An example of the top view and behind view of the same seabed feature is shown in Figure 4.11 using the QPS swath editor.

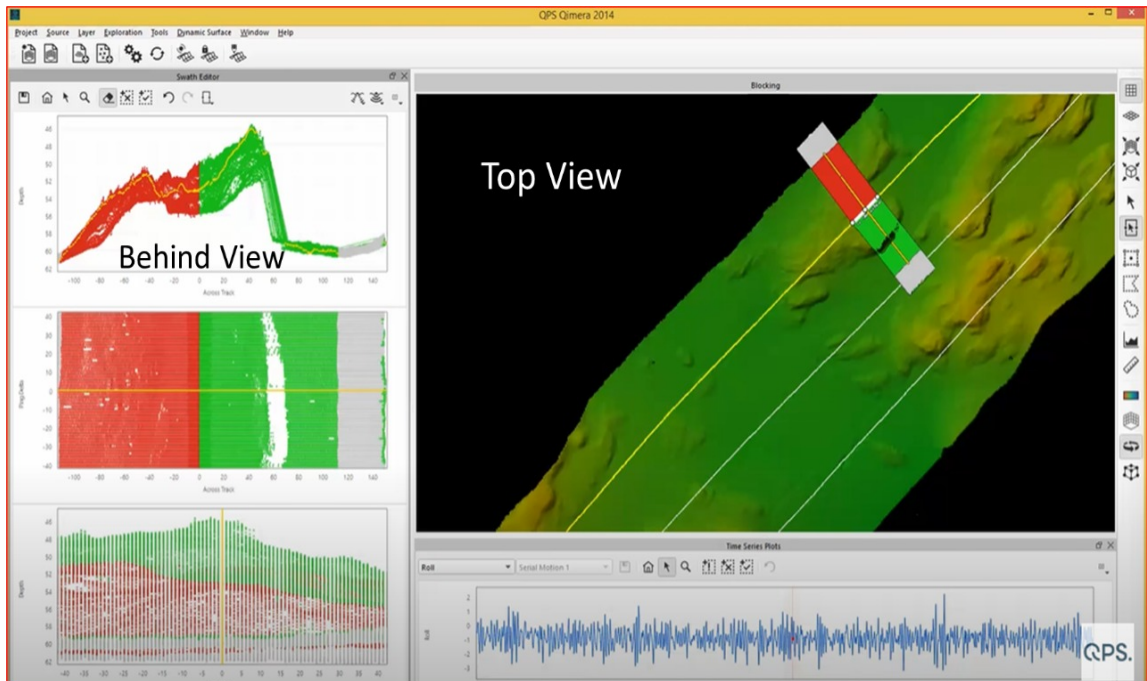


Figure 4.11: QPS Swath Editor top view and behind view [21]

When seeing the data from behind view, the user can easily identify outliers and the seabed from the feature. With this view, the users keep navigating to all the parts of the data to identify the outliers from the seabed.

This research tries to develop a Reinforcement Learning approach that can take the behind view of the swath editor and identify outliers present in the data.

4.9.3 Designing of the Reinforcement Learning

As mentioned in the previous chapter, the Reinforcement Learning environment provides three components - observation space, action space and rewards for designing the agent to train.

One of the libraries that provides a framework to create a Reinforcement Learning environment is OpenAI [32]. OpenAI provides gym spaces where the user can create the environment for making the agent take steps and get rewards based on its actions. OpenAI was initially developed for video games such as Atari and board games. OpenAI made all the game environments and released the environment to the user where the user can test their Reinforcement Learning agent. However, with the growing need to develop environments for purposes other than games, OpenAI released a framework where the users can develop their environments for their own needs.

This research has developed a Reinforcement Learning environment that can take a swath editor view of the multibeam data, make the agent identify the outliers in the data, and generate a reward for the agent based upon its identification. In the further coming sections, the design of the environment is described.

The outline given by OpenAI for designing the custom environment is that the environment should be a single method that invokes the *gym.spaces* library. The method should consist of the following functions within it - `init()`, `step()` and `reset()`. The parameters and functionalities of each function are given below.

- `init()` - The `init` method is the initialization function of the environment. The user sets two variables within this function - `observation_space` and `action_space`. The `observation_space` defines the shape of the observation the agent sees at a particular time, and the `action_space` defines the set of actions the agent can perform.
- `step()`- The `step` function is to make the agent take steps and generate rewards. The `step` method takes a parameter called `action` and returns four parameters named `observation`, `reward`, `done` and `info`.
 - `action` - The `action` parameter stores the value of the set of actions the agent has taken by seeing an observation.
 - `observation` - The `observation` parameter stores the value of the observation for the agent to take action upon.
 - `reward` - The value of the reward generated for the particular action on a particular observation is stored.
 - `done` - This is a boolean parameter which is set to `True` when the agent has completed the task.
- `reset()` - This method is called the first time to create the first observation for the agent. The agent observes and generates the set of actions that are stored in the `action` variable, which is passed to `step()`.

In the following section, the design of each method and the workflow within the environment are discussed.

The first step in the workflow is creating the observation for the agent. From the thesis point of view, the observation would be the depth of soundings the agent is observing. The depth soundings are arranged similarly to achieve the swath editor behind view. To achieve the view, `OceanMappingDataframe` is used, which loads the hydrography data into the dataframe. Using the multi-index feature of the dataframe, the user can navigate and select a certain number of records.

From the selected number of records, a pivot table having the records as the columns, beam number as the rows and depth as the values is created. This pivot table is called the depth table.

When the depth table is plotted, the user can see the hydrography data from the behind view of the swath editor. Each row in the depth table corresponds to the depth values collected by a beam for a certain number of records. A sample-generated depth table is shown in Figure 4.12 for data from 80 records and 432 beams. The view achieved by plotting the depth table is given in Figure 4.13, where green represents outliers and blue represents inliers. To plot the depth table Matplotlib [14] package was used.

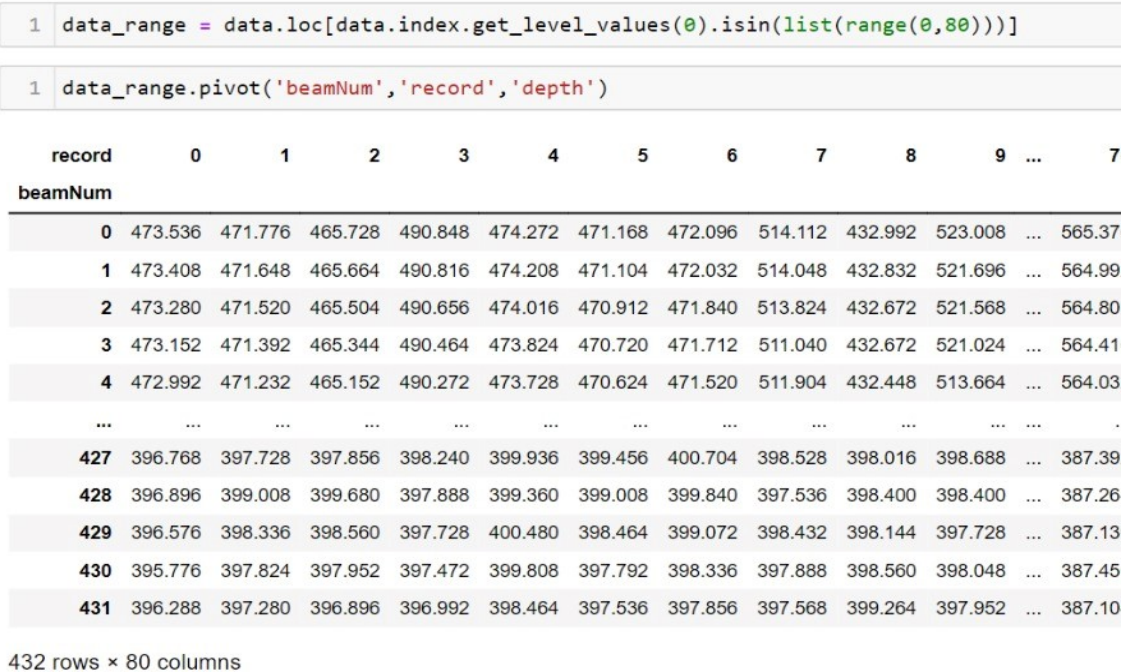


Figure 4.12: An example of depth table

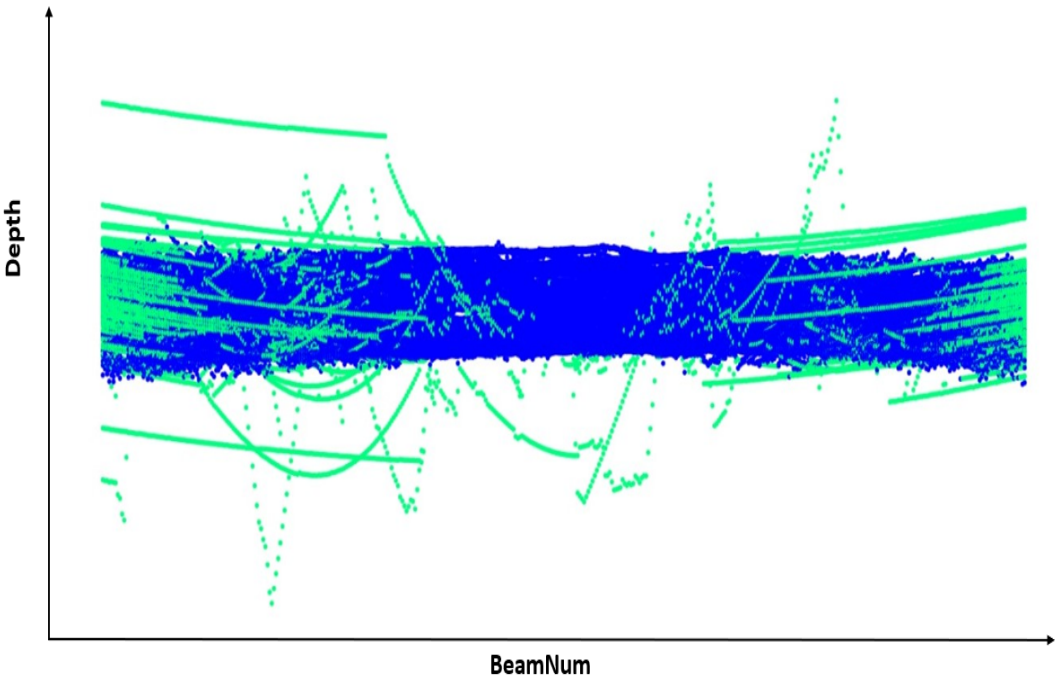


Figure 4.13: View of the Depth Table

After the view for the selected number of records is created, the `init()`, `step()`, and `reset()` functions are implemented. To implement the functions, and to demonstrate that `OceanMappingDataframe` can work seamlessly with popular AI frameworks, the thesis uses the `stable-baselines3` [28] library. The `stable-baselines3` is a popular PyTorch-based Reinforcement Learning library used for defining the variables inside the `init()`, `step()`, and `reset()` functions and for invoking the Reinforcement Learning agent that can work seamlessly with the functions inside the environment.

The `init()` function is first implemented. To implement the `init()` function, `observation_space` is defined to be a one-dimensional array having `i` number of values. The integer `i` is the number of records the user has selected for the agent to train. The `observation_space` is filled with the depth soundings of a single beam number from the depth table at a time. The `action_space` is defined as a one-dimensional boolean list having `i` values.

After defining the values in the `init()`, the `reset()` function is implemented. In this function, the first row of the depth table is made as a one-dimensional array and returned as the first observation to the `step()` function.

$$observation_0 = [d_{00}, d_{01}, d_{02}, \dots, d_{0i}]$$

Based on the $observation_0$, the agent takes the first action, $action_0$, to identify which depths are outliers and inliers from the $observation_0$ list. If it identifies the depth as an outlier, it flags it as 1, and if it identifies the depth as an inlier, it flags as 0. So, the agent, upon receiving the observation array, will return the action array, which would be $action_0 = [a_{00}, a_{01}, a_{02}, \dots, a_{0i}]$.

This action array is passed inside the `step()` function to generate a reward based on correctly identifying the soundings as inliers and outliers. To generate the reward, a status flag table similar to the depth table is created using the status flag variable from the `OceanMappingDataframe`. In Figure 4.14, the status flag table represents the flags assigned manually for each sounding present in the depth table. Flag 0 represents the sounding to be an inlier, and flag 1 represents the sounding to be an outlier.

1

`data_range.pivot('beamNum','record','statusFlag')`

1

| record | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ... | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 |
|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| beamNum | | | | | | | | | | | | | | | | | | | | | |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | ... | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | ... | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | ... | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | ... | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | ... | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 427 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 428 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 429 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 430 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 431 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

432 rows × 80 columns

Figure 4.14: Status Flag Table

Inside `step()`, the first row of the status flag is accessed to generate the *status_flag* array. The $status_flag_0 = [s_{00}, s_{01}, s_{02}, \dots, s_{0i}]$. The reward function takes in the $action_0$ and $status_flag_0$ array and performs an element-wise comparison to identify how many flags the agent gave match with the flags given manually. The number of

flags that match is given as a reward to the agent.

After generating the reward for the first beam number inside the `step()` function, the second beam number's depth is given from the depth table as $observation_1 = [d_{10}, d_{11}, d_{12}, \dots, d_{1i}]$ and the agent gives the flag array as $action_1 = [a_{10}, a_{11}, a_{12}, \dots, a_{1i}]$. The reward for the second beam is generated based on the action and status flag array. This process continues until the agent covers all the beams. Upon completion, the done variable in the `step()` function is set to True, and the environment returns the control to the main function. Using the popular Reinforcement Learning library, stable-baselines3 [28], the agent is made to train again from the first beam for a specified number of times. The control flow of the proposed Reinforcement Learning is shown in 4.15.

As shown in Figure 4.15, a GSF or a CSV file is loaded into an OceanMapping-Dataframe using the `read_gsf()` or `read_csv()` API. From the OceanMapping-Dataframe, the user selects i records. For the selected i records, using the `pivot()` method, the depth table and corresponding status flag are created. Then, by iterating through the beam number in the depth table, an array of depth values for a particular iteration is passed as observation to the Reinforcement Learning agent. The agent, upon observing the array of depth values, predicts the inlier and outlier flags for each depth. Finally, the array of predicted flags is compared with the manually annotated status flag array, and a reward is generated for the agent for that particular iteration.

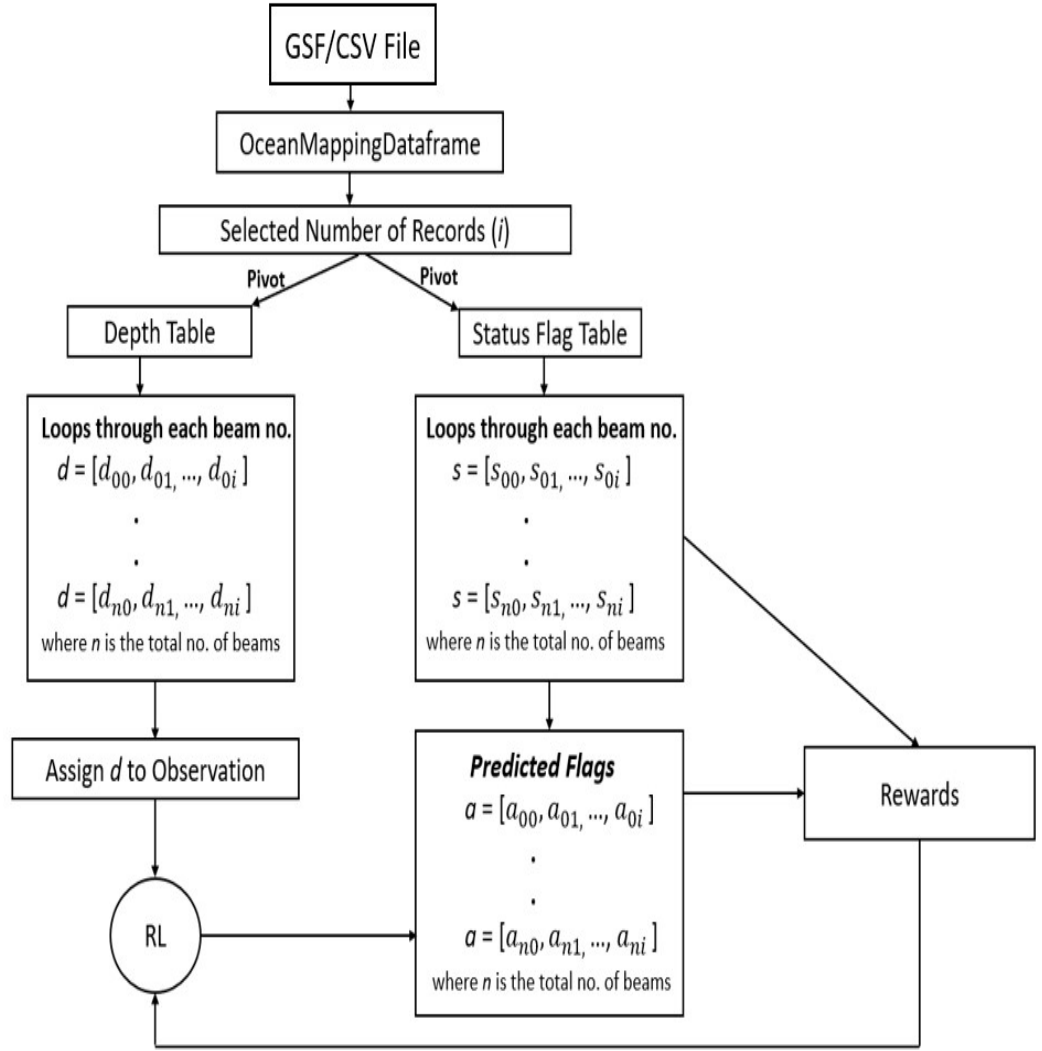


Figure 4.15: Control Flow of Reinforcement Learning

In this Reinforcement Learning process, the objective is to observe if an agent can learn to flag the soundings by observing the collection of depth values from a beam. The main idea is that when the data is flagged manually, the soundings that are away from the majority of the soundings are considered an outlier. This research aims to observe if the Reinforcement Learning agent can identify the same pattern on its own and learn to identify the outlier values from the majority values in a list of given depths for a beam. So in the next chapter, discusses the performance of the Reinforcement Learning agent and test if it can learn progressively to identify the outliers.

4.10 Concluding Remarks

In this chapter, the proposed approach and contributions were discussed. The proposed OceanMappingDataframe components was explained along with the dataframe's structure and API calls. The design of the Reinforcement Learning strategy to identify outliers in the multibeam data was also discussed. The next chapter will discuss the results and evaluation of the proposed system.

Chapter 5

Experimental Evaluation

5.1 Overview

This chapter explains the experimental evaluation process conducted to test the proposed OceanMappingDataframe along with the demonstration of the Reinforcement Learning algorithm. The evaluation and benchmarking of the OceanMappingDataframe is first presented and later the demonstration and performance of the developed Reinforcement Learning algorithm is explained.

5.2 Experiment Setup

The experiments were conducted in a dedicated Google Colab environment with 2 Intel(R) Xeon(R) vCPUs @ 2.20GHz. The system had 12 GB RAM and 107 GB disk space.

5.3 Dataset Description

To benchmark the OceanMappingDataframe, three different GSF file sizes - 100 MB, 300 MB, and 500 MB were used from the Ocean Mapping Group datasets. The data were collected during the 2015 Shallow Surveys using the Teledyne Reson SeaBat 7125 Multibeam Echosounder. The number of ping records and number of rows after processing through the OceanMappingDataframe is shown in Table 5.1. All three dataframes had 14 columns - record, beamNum, time, roll, pitch, heave, lat, lon, depths, across_track, along_track, travel_time, quality_factor, and beam_flags.

Table 5.1: Dataset description for OceanMappingDataframe

| Size | Number of Ping Records | Number of Rows |
|--------|------------------------|----------------|
| 100 MB | 3733 | 1,906,176 |
| 300 MB | 11169 | 5,718,528 |
| 500 MB | 18615 | 9,530,880 |

To benchmark the Reinforcement Learning, five manually annotated datasets from a multibeam survey were considered. This data also belongs to the Ocean Mapping Group and are different from the above-mentioned GSF datasets. The number of records that were considered from each dataset is given in Table 5.2. The number of records considered is the standard window size used for manually identifying outliers.

Table 5.2: Number of Records Considered for each Dataset

| Dataset | Number of Ping Records |
|----------|------------------------|
| Dataset1 | 50 |
| Dataset2 | 50 |
| Dataset3 | 60 |
| Dataset4 | 60 |
| Dataset5 | 80 |

5.4 Evaluation

5.4.1 OceanMappingDataframe Vs Pandas

The performance of loading GSF files into a Pandas dataframe and OceanMappingDataframe is first evaluated. This experiment is conducted to observe the scalability of OceanMappingDataframe over Pandas dataframe. The same GSF reader and `read_gsf()` as explained in section 4.6, the API was used to load the Pandas dataframe and OceanMappingDataframe. The results of the experiment are given in Table 5.3 and Figure 5.1.

OceanMappingDataframe performed $1.7\times$ faster than Pandas to create 5.7 Million rows of the dataframe and $1.3\times$ faster to create 9.5 Million rows of the dataframe. As OceanMappingDataframe is using MODIN as its backend, it uses distributed computing principle to utilize all the available cores of the CPU to process the op-

eration faster than Pandas, which only uses one core of the CPU. Also, as MODIN provides seamless distribution of the computation to a cluster of CPUs thereby the performance of the computation can scale if the users add more cores or CPUs. The `read_gsf()` API heavily uses the `concat()` method to keep concatenating the NumPy arrays returned from the GSF reader into a dataframe by utilizing MODIN's `pd.concat()` method. This approach means the OceanMappingDataframe can concatenate the dataframes faster than the Pandas dataframe by distributing the computation to all the available CPU cores.

Table 5.3: Comparison of times (in seconds) to load hydrographic multibeam datasets into a Python dataframe

| | Time Consumption(s) | |
|----------|---------------------|-----------------------|
| Size(MB) | Pandas | OceanMappingDataframe |
| 100 | 150.12 | 139.32 |
| 300 | 1527.15 | 903.56 |
| 500 | 3527.19 | 2794.46 |

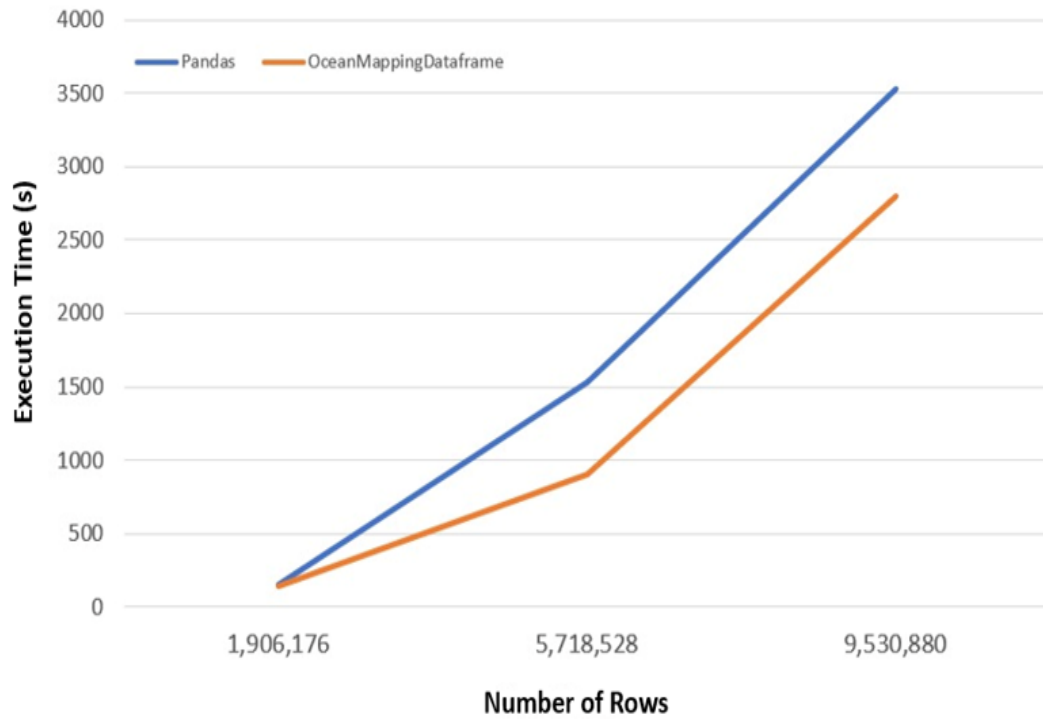


Figure 5.1: Time to load GSF Files in OceanMappingDataframe vs Pandas

5.4.2 Comparison between Multi-Index and Single Index OceanMappingDataframe

As OceanMappingDataframe uses multi-index to reference the objects present in the dataframe, the time taken to create a multi-indexed and a single-indexed OceanMappingDataframe was evaluated. To test this using `read_gsf()` API, the `set_index` parameter was initialized to `False` to obtain the single-indexed dataframe. To obtain the multi-indexed dataframe, the `set_index` parameter was initialized to `True`. In Table 5.4, the results of the experiments are shown.

Table 5.4: Time comparison (in seconds) for reading Single-Index and Multi-Index OceanMappingDataframe

| | Time Consumption(s) | |
|----------|---------------------|-------------|
| Size(MB) | Single-Index | Multi-Index |
| 100 | 139.32 | 212.42 |
| 300 | 903.56 | 1073.07 |
| 500 | 2794.46 | 3174.94 |

It can be observed that it takes $0.84\times$ and $0.88\times$ longer to create 5.7 Million rows and 9.5 Million rows of Multi-indexed dataframe than creating respective Single-Indexed dataframe.

5.4.3 Saving to Parquet Format

After the OceanMappingDataframe is created, the users can save the data in the Parquet format. To test this process, both the multi-indexed and single-indexed OceanMappingDataframe were saved using the `save()` API of OceanMappingDataframe.

The time it takes to save both versions of the OceanMappingDataframe and the storage they occupy were noted. For this test, partition columns were not set. Table 5.5 represents the results to save the dataframe to Parquet format using single-index and multi-index.

Table 5.5: Time to save (in seconds) to Parquet format using Single-Index and Multi-Index

| | Single-Index | | Multi-Index | |
|----------|---------------------|------------------|---------------------|------------------|
| Size(MB) | Time Consumption(s) | Parquet Size(MB) | Time Consumption(s) | Parquet Size(MB) |
| 100 | 81.67 | 145 | 74.12 | 144 |
| 300 | 216.753 | 435 | 292.581 | 431 |
| 500 | 710.42 | 725 | 714.65 | 719 |

It can be observed that as the data grows from 5.7 Million rows to 9.5 Million rows, the time it takes to store the Parquet increases by three times, whereas the storage size increases by $1.8\times$. Also, it is observed that there is a tradeoff between storing the multi-index dataframe as Parquet. Multi-indexed Parquet reduces the storage consumption, but the process of storing the multi-indexed dataframe consumes additional time.

5.4.4 Loading Parquet Files

Once the user saves the hydrography data in Parquet format, the user can use the `read_parquet()` API to read the Parquet files to an `OceanMappingDataframe`.

For this experiment, both the multi-indexed and single-indexed Parquet formats were retrieved. The time it takes to read both versions of the Parquet files to `OceanMappingDataframe` were observed. Table 5.6 shows the results of these experiments.

Table 5.6: Results of Loading Parquet Files

| | Time Consumption(s) | |
|----------|---------------------|-------------|
| Size(MB) | Single-Index | Multi-Index |
| 100 | 11.59 | 11.22 |
| 300 | 32.12 | 35.52 |
| 500 | 66.64 | 69.23 |

It is observed that the Parquet reader could parse all the datasets in under a minute by storing the files in Parquet format. Moreover, there was no significant time spent reading multi-indexed dataframe over single-indexed dataframe.

5.4.5 Select Function

The select operation is one of the primary and frequently used operations in dataframe. Often the users are required to select a portion of the data based on a condition. By principle, a multi-indexed dataframe returns the result of a select operation faster than a single-indexed dataframe. The multi-index dataframe performs faster because the cursor of the dataframe can easily retrieve the required index positions in the multi-index dataframe compared to performing an exhaustive search of a single-indexed dataframe to return the results that match the user's condition.

As OceanMappingDataframe uses a multi-index principle with row labels as record and beam number, an experiment was conducted to identify the performance of the select operation on the multi-index dataframe. To conduct this experiment, a select operation with a condition was chosen, and the operation was executed in both single-indexed and multi-indexed dataframe. To test this, the `read_gsf()` API was used, and the `set_index` parameter was initialized to False to obtain the single-

indexed dataframe. On the other hand, to obtain the multi-indexed dataframe, the `set_index` parameter was initialized to `True`.

The select query that was executed in both the dataframes was to identify and retrieve the beam numbers ranging from 0 to 255 from all the ping records. Each ping record has 512 beams, and the dataframe is queried to retrieve one-half of the entire dataframe. The statements used to run the query are as follows:

For Single Index Dataframe

```
data.loc[data["beamNum"].isin(range(0,256))]
```

For Multi-Index Dataframe

```
data.iloc[data.index.get_level_values(1).isin(range(0,256))]
```

The results of the experiment are given in Table 5.7. The results show that it takes twice as long to search and return the matching beam numbers in a single-indexed dataframe compared to the multi-indexed dataframe. The multi-indexed dataframe can retrieve the data matching the beam numbers faster because the multi-indexed dataframe holds the record and beam number as indexes, and it is easier for the cursor to locate the range of matching beam numbers from the index value. In a single-indexed dataframe, the cursor must perform an exhaustive search of the `beamNum` column to identify the matching beam numbers.

Table 5.7: Comparison of the time (in seconds) to select a subset of half a dataframe using a single index and a multi-index.

| | Time Consumption(s) | |
|----------|---------------------|-------------|
| Size(MB) | Single-Index | Multi-Index |
| 100 | 1.78 | 0.84 |
| 300 | 2.44 | 1.23 |
| 500 | 2.539 | 1.447 |

5.4.6 Pivot Function

The OceanMappingDataframe was tested with other operations besides I/O operations. One of the main dataframe operations is the pivot operation. Also, to create the inputs for the proposed Reinforcement Learning algorithm, the pivot function is heavily used to create the depth table and status table. To test this operation, 1k, 3k, and 5k ping records were taken, creating OceanMappingDataframes of 512K, 1.53 Million and 2.52 Million rows, respectively. On these dataframes, the pivot operation is applied on record, beamNum and depths columns to generate a pivot table where the record is the column axis, beamNum is the row axis, and the table values are depths. The results of the experiments are shown in Table 5.8.

Table 5.8: Time (in seconds) to perform pivot operations on three different OceanMappingDataframe sizes.

| Ping Records | Number of Rows | Time Consumption(s) |
|--------------|----------------|---------------------|
| 1k | 512,000 | 59.70 |
| 3k | 1,536,000 | 219.02 |
| 5k | 2,560,000 | 368.91 |

From the results, it is observed that as the data volume grew by three times, the time consumption to create the pivot table was $3.7\times$. However, as the data volume grew from 3 to $5\times$, the time consumption to create the pivot table was only $1.9\times$. Here it is observed that the OceanMappingDataframe can scale and perform the pivot operation as the data volume grows.

5.4.7 Reinforcement Learning

As mentioned in the previous chapter, to showcase the application of the OceanMappingDataframe, a Reinforcement Learning strategy is developed to identify outliers in the multibeam sonar. The Reinforcement Learning algorithm was tested on five datasets, each of which consisted of manually annotated flags of inliers and outliers. Then, the data was passed to the Reinforcement Learning algorithm, as mentioned in chapter 4.

To develop the Reinforcement Learning algorithm, the stable-baselines3 [23] library was used. The stable-baselines3 offers various Reinforcement Learning algorithms. The selection of the appropriate algorithm depends upon the type of observation

and action space. The proposed demonstration uses a discrete observation space and multi-binary action space. Thereby the popular and effective algorithms that can accomdate discrete observation space and multi-binary action space are Proximal Policy Optimization (PPO) and Advantage-Actor Critic (A2C) algorithms [23]. This thesis uses the PPO algorithm to train and test the agent on all the datasets.

Three learning scenarios were fixed to evaluate the algorithm’s performance. First, the algorithm was trained in 100k, 500k and 1 Million timesteps. A timestep is an episode of complete observations and actions. The idea behind Reinforcement Learning is that the agent keeps learning as the timesteps progress.

In this demonstration, a timestep indicates where the agent receives all the beams and completes identifying outliers in all the beams. After completing a timestep, the agent returns to the first beam and continues the process until the user-specified timesteps. After each timestep, the user receives the average reward of the agent for that particular timestep. A reward is a total number of agent flags matching the manual flags.

Algorithm 1: Pseudocode of the Reinforcement Learning algorithm

Input : user-specified timesteps

Output: average rewards for each timestep

1. Set `current_timestep` = 1
 2. Repeat Steps 2 through 5 until `current_timestep` < user-specified timesteps
 - (a) Set `total_reward` = 0
 - (b) Repeat Steps i and ii for all beams
 - i. Identify outliers in current beam
 - ii. Update `total_reward` based on the number of matching flags between the agent and manual flags
 3. Calculate `average_reward` for `current_timestep` by dividing `total_reward` by the number of beams
 4. Print `average_reward` for `current_timestep`
 5. Increment `current_timestep` by 1
-

The objective of this test is to first observe whether Reinforcement Learning can be applied to identify outliers from hydrographic data and to benchmark the algorithm's average reward performance on the three different scenarios.

5.4.7.1 Performance of RL on Dataset 1

Figure 5.2 (a) shows the manually annotated Dataset 1 of Table 5.2. This dataset consists of 50 records. The green points refer to the outlier flags, and the blue refers to the inliers or the seabed features. The Reinforcement Learning agent was trained on the dataset for 100k, 500k and 1 Million timesteps. The results of the average

reward obtained after training for each timestep are shown in Table 5.9.

Table 5.9: Performance of RL on Dataset 1 Results

| Time Steps | Average Reward | Accuracy (%) | Precision (%) | Recall (%) | Time Taken(s) |
|------------|-------------------|-----------------|------------------|------------|------------------|
| 100k | 29 | 61.57 | 16.40 | 44.56 | 202.38 |
| 500k | 35 | 72.83 | 18.05 | 28.19 | 968.50 |
| 1 Million | 32 | 66.60 | 17.71 | 39.93 | 1907.27 |

From Table 5.9, it was observed that the agent could predict 29 flags of 50 records in each beam, similar to manual flags at 100k timestep. As the agent was trained for 500k timesteps, it could predict 35 flags of 50 records in each beam, similar to manual flags. As the agent was trained more for 1 Million timesteps, the agent's performance decreased to predict correctly 32 flags of 50 records. The performance decreases due to bias and overfitting. Avoiding bias and overfitting is a completely independent problem in machine learning. This thesis does not solve the bias and overfitting problem but only records the agent's performance for different timesteps.

The following are the criteria used to evaluate the Reinforcement Learning performance measures.

1. *True Negative (TN)*: Number of outliers detected as outliers.
2. *False Negative (FN)*: Number of inliers detected as outliers.
3. *True Positive (TP)*: Number of inliers detected as inliers.
4. *False Positive (FP)*: Number of outliers detected as inliers.

Accuracy is defined as the total number of correct predictions and is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision is the ratio of true detection of malicious attacks over the sum of true malicious attacks and false detection of benign data and is calculated as follows:

$$Precision = \frac{TP}{TP + FP}$$

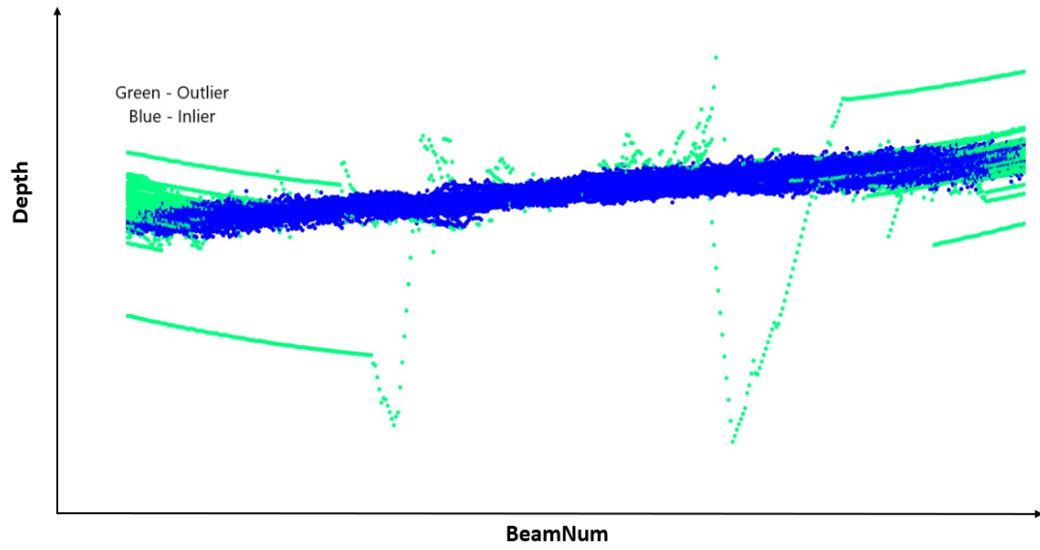
Recall is the ratio of true positives over the sum of true malicious detection and false detection of benign data and is calculated as follows:

$$Recall = \frac{TP}{TP + FN}$$

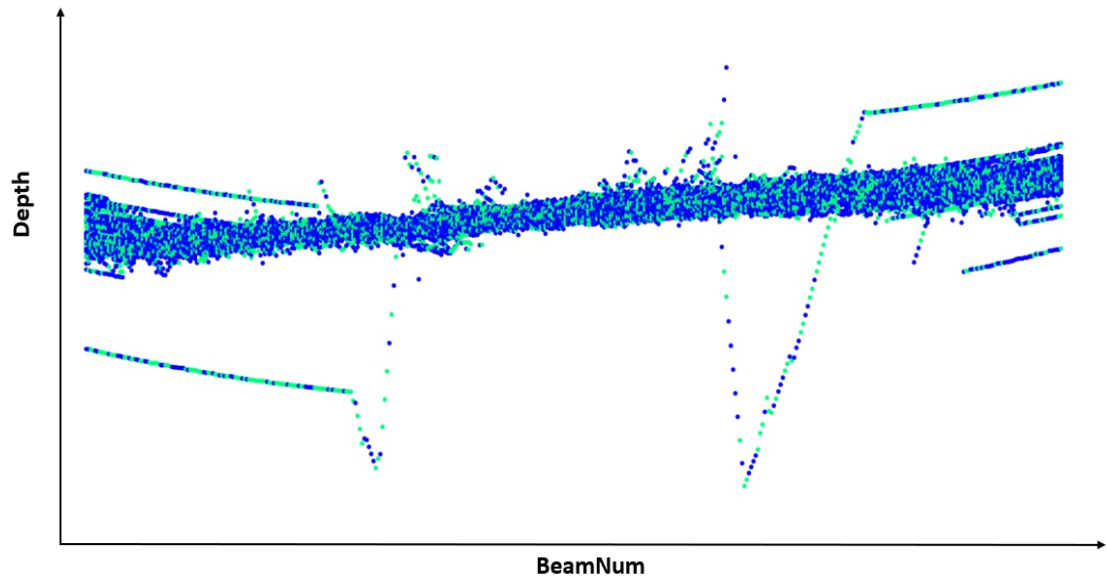
In Figures 5.2 (b) and 5.2 (c), the agent's performance at 100k timesteps and 500k timesteps is shown. The highlighted box in Figure 5.2 (c) show the places where the agent's performance has improved.

From Figure 5.2 (c), it is observed that the agent at 500k timesteps was able to identify the seabed more accurately as compared to the agent at 100k timesteps.

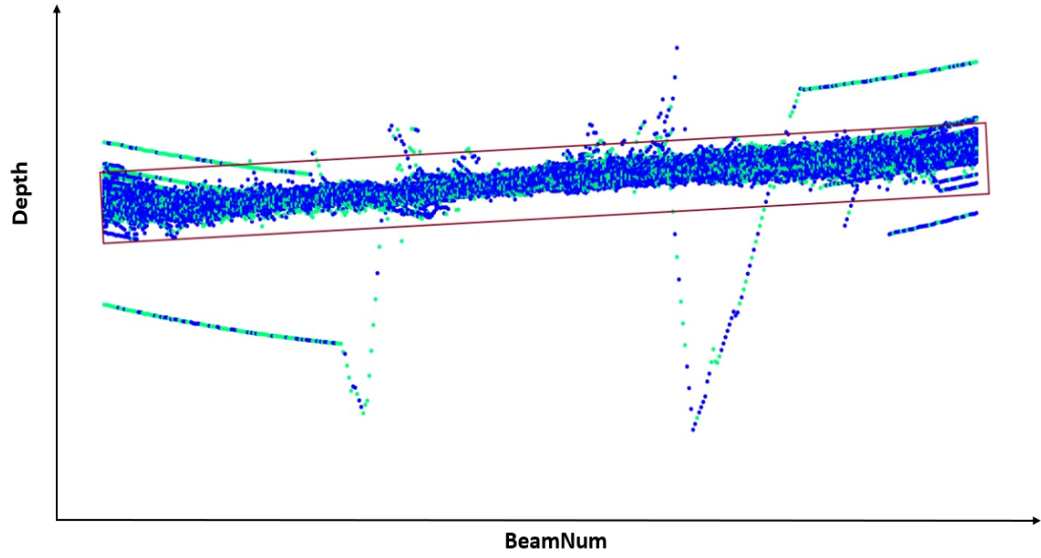
This difference in the ability to better identify is indicated in the highlighted box.



(a) Manually Annotated



(b) Result at 100K Time Step



(c) Result at 500K Time Step

Figure 5.2: Dataset 1 Results

5.4.7.2 Performance of RL on Dataset 2

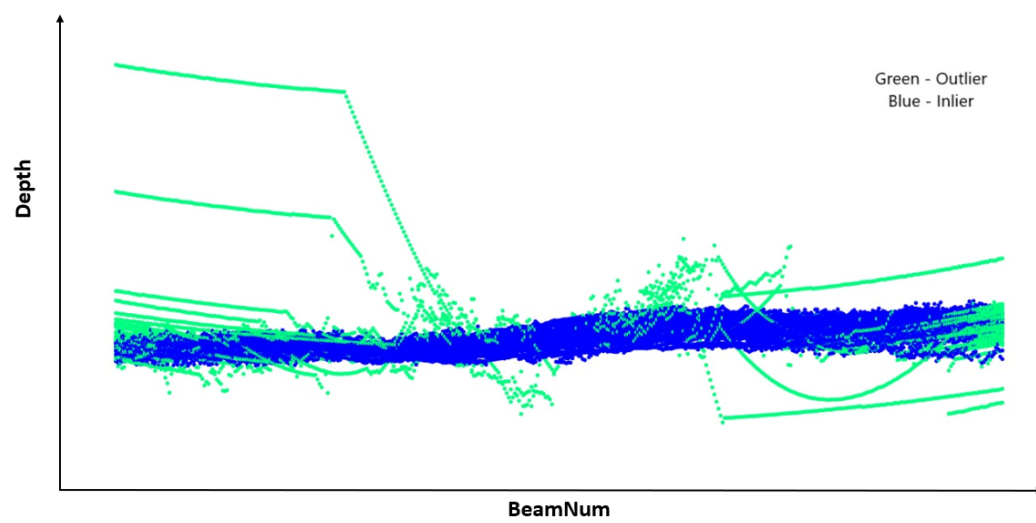
Figure 5.3 (a) shows the manually annotated Dataset 2. This dataset consists of 50 records. The green points refer to the outlier flags, and the blue refers to the inliers or the seabed features. The Reinforcement Learning agent was trained on the three timesteps, and the performance of the agent is given in Table 5.10. In this dataset, the agent performance increases from identifying 30 status flags to 37 status flags out of 50 records. However, later due to bias, the performance decreased.

Table 5.10: Performance of RL on Dataset 2 Results

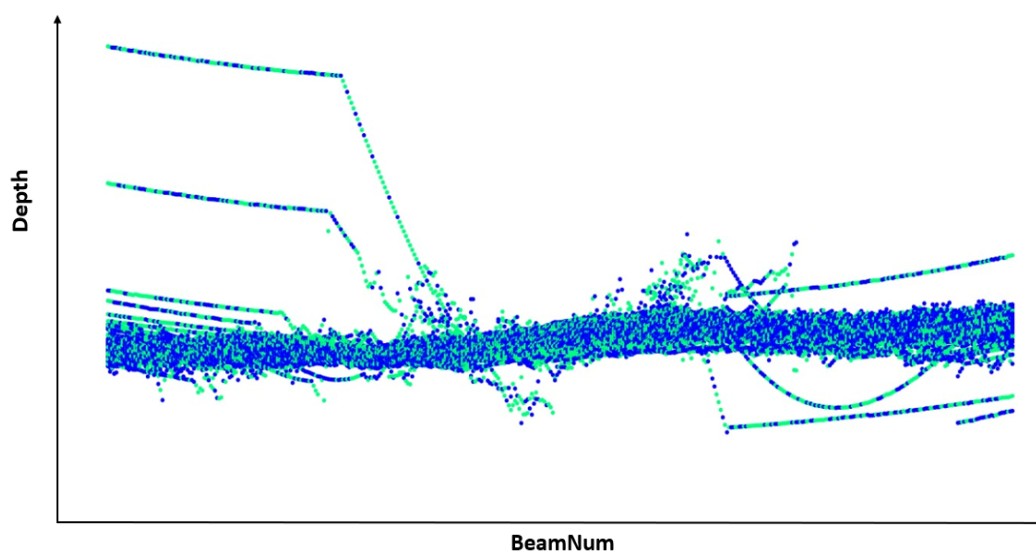
| Time Steps | Average Reward | Accuracy (%) | Precision (%) | Recall (%) | Time Taken(s) |
|------------|-------------------|-----------------|------------------|------------|------------------|
| 100k | 30 | 60.38 | 25.46 | 49.36 | 212.52 |
| 500k | 37 | 72.95 | 37.70 | 50.86 | 850.23 |
| 1 Million | 35 | 65.28 | 31.57 | 60.82 | 1709.33 |

In Figures 5.3 (b) and 5.3 (c), the agent’s performance at 100k timestep and 500k timestep is shown. The highlighted boxes in Figure 5.3 (c) show the places where the agent’s performance has improved.

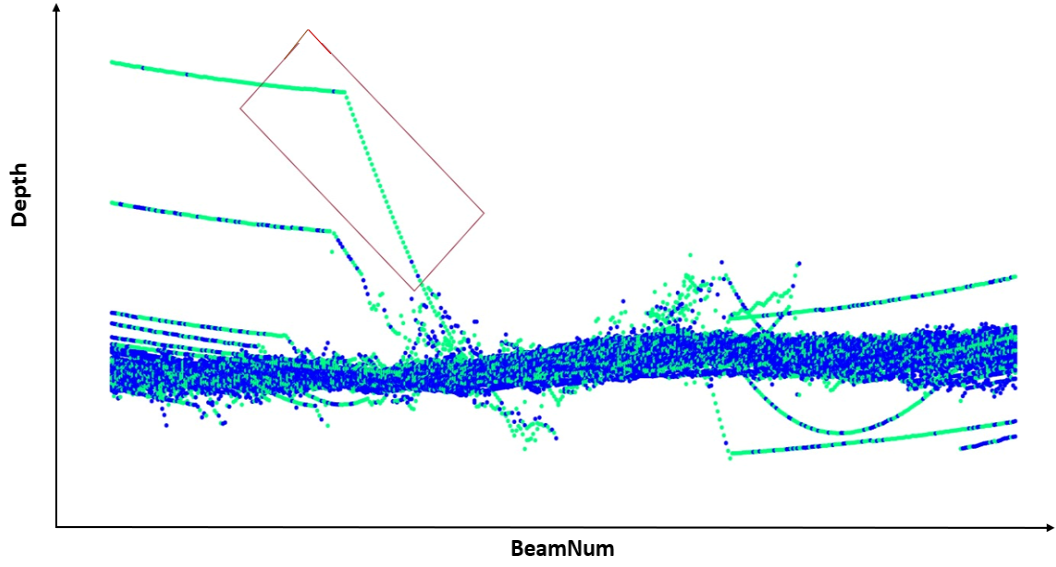
From Figure 5.3 (c), it is observed that the agent at 500k timestep was able to identify the outlier better as compared to the agent at 100k timestep, this difference in the ability to identify outliers is indicated in the highlighted box.



(a) Manually Annotated



(b) Result at 100K Time Step



(c) Result at 500K Time Step

Figure 5.3: Dataset 2 Results

5.4.7.3 Performance of RL on Dataset 3

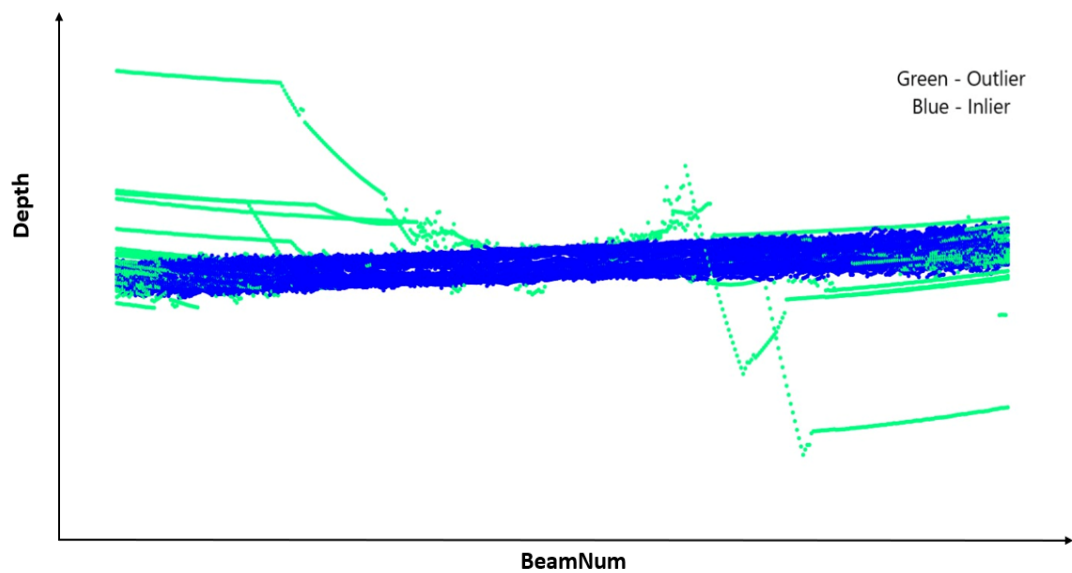
Figure 5.4 (a) shows the manually annotated Dataset 3. This dataset consists of 60 records. The green points refer to the outlier flags, and the blue refers to the inliers or the seabed features. The Reinforcement Learning agent was trained on the three timesteps, and the performance of the agent is given in Table 5.11. In this dataset, the agent performance increases from identifying 35 flags in each beam to 43 flags out of 60 records. However, later due to bias, the performance decreased.

Table 5.11: Performance of RL on Dataset 3 Results

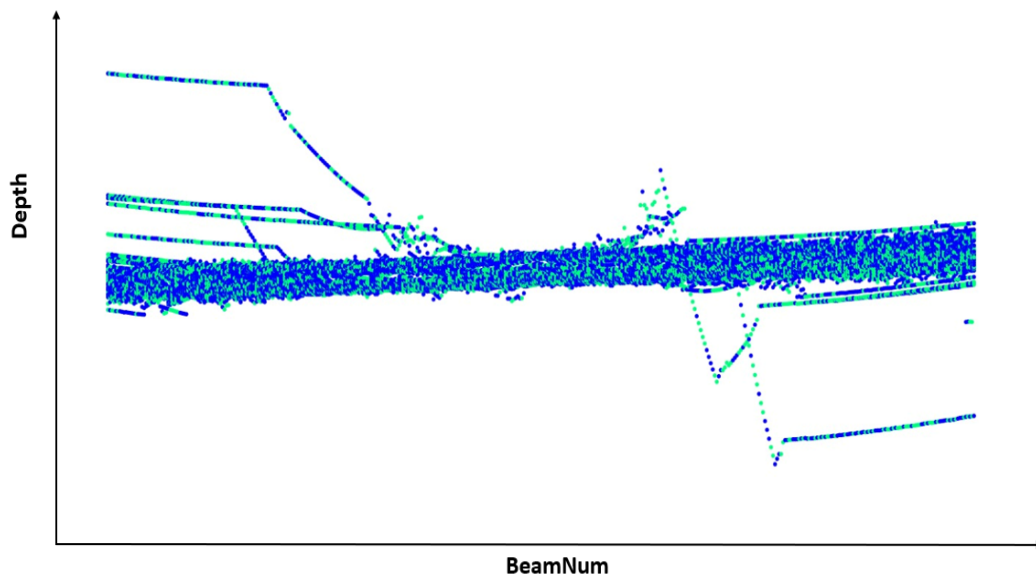
| Time Steps | Average Reward | Accuracy (%) | Precision (%) | Recall (%) | Time Taken(s) |
|------------|-------------------|-----------------|------------------|------------|------------------|
| 100k | 35 | 59.82 | 19.95 | 50.26 | 206.91 |
| 500k | 43 | 70.44 | 24.98 | 42.39 | 1084.20 |
| 1 Million | 40 | 68.64 | 24.64 | 46.69 | 2040.36 |

In Figures 5.4 (b) and 5.4 (c), the agent’s performance at 100k timestep and 500k timestep is shown. The highlighted boxes in Figure 5.4 (c) show the places where the agent’s performance has improved.

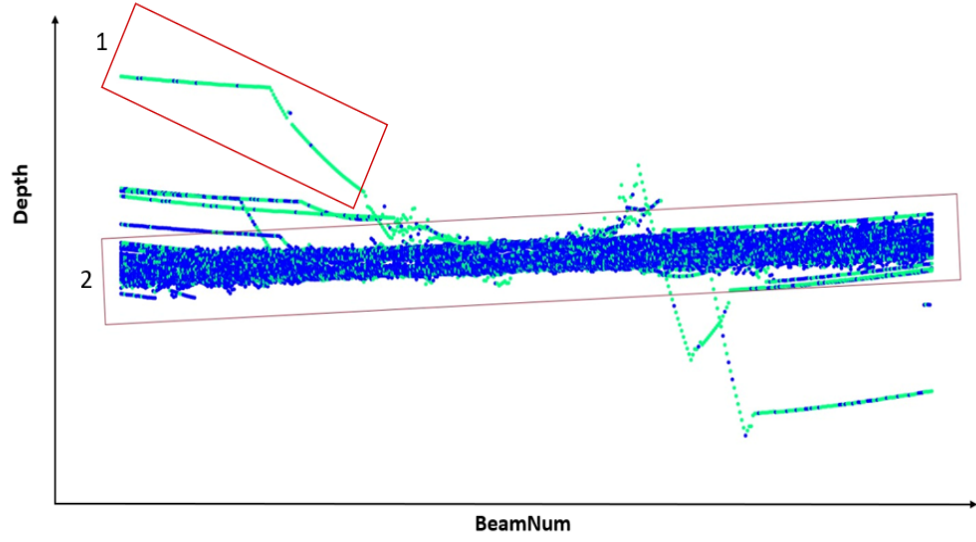
From Figure 5.4 (c), it is observed that the agent at 500k timestep could identify both the seabed and the outlier more accurately than the agent at 100k timestep. This difference in the ability to identify seabed better is indicated in highlighted box 1, and the difference in the ability to identify outlier better is indicated in highlighted box 2.



(a) Manually Annotated



(b) Result at 100K Time Step



(c) Result at 500K Time Step

Figure 5.4: Dataset 3 Results

5.4.7.4 Performance of RL on Dataset 4

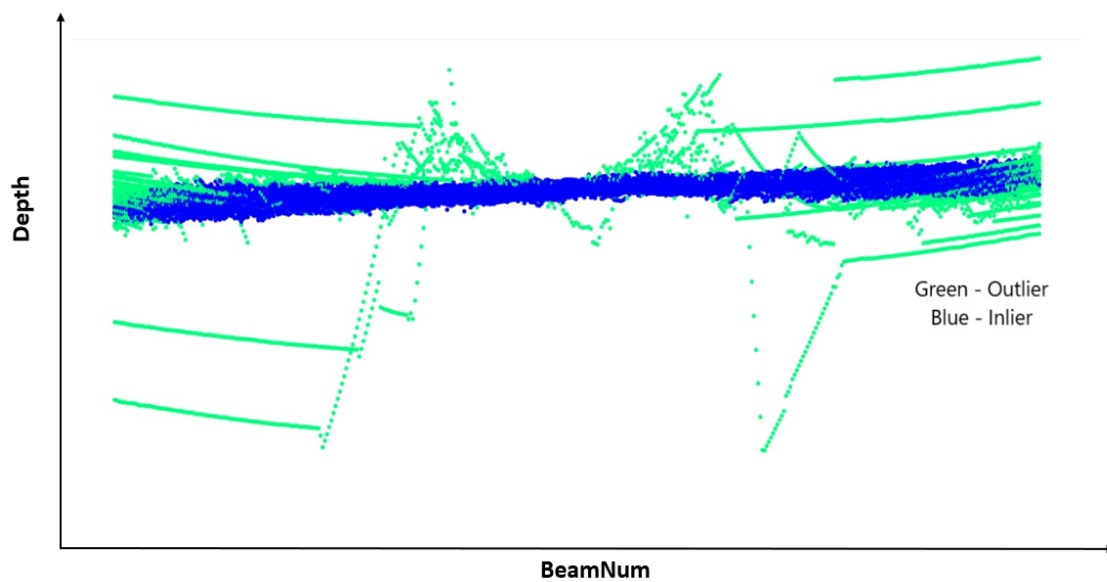
Figure 5.5 (a) shows the manually annotated Dataset 4. This dataset consists of 60 records. The green points refer to the outlier flags, and the blue refers to the inliers or the seabed features. The Reinforcement Learning agent was trained on the three timesteps, and the performance of the agent is given in Table 5.12. In this dataset, the agent performance increases from identifying 34 flags out of 60 records in each beam to 42 flags. However, later due to bias, the performance decreased.

Table 5.12: Performance of RL on Dataset 4 Results

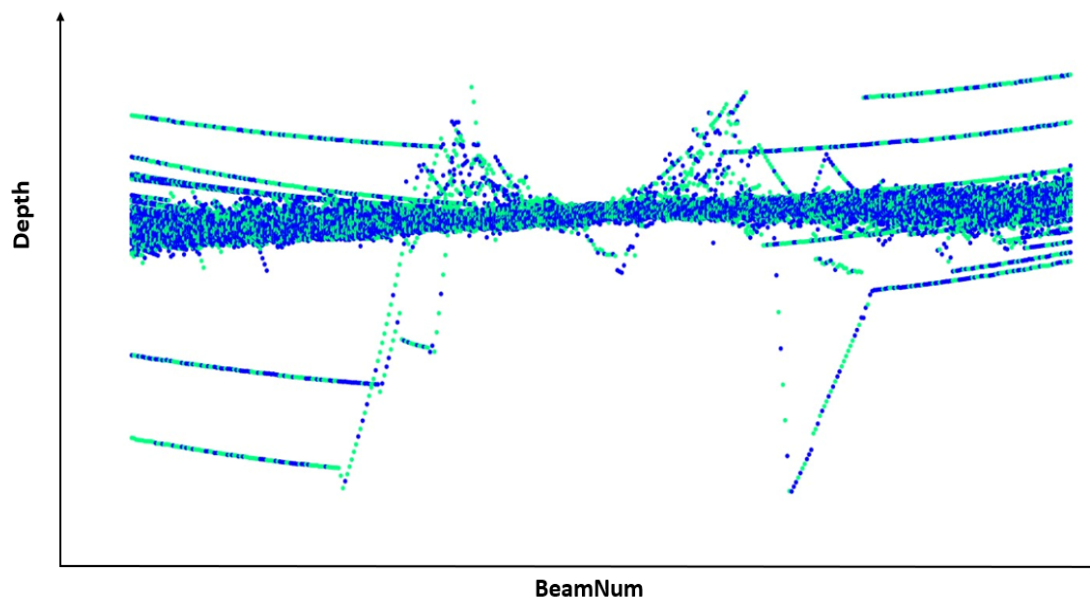
| Time Steps | Average Reward | Accuracy (%) | Precision (%) | Recall (%) | Time Taken(s) |
|------------|-------------------|-----------------|------------------|------------|------------------|
| 100k | 34 | 57.65 | 23.53 | 4832 | 227.05 |
| 500k | 42 | 67.92 | 31.44 | 49.20 | 1110.03 |
| 1 Million | 40 | 61.16 | 25.06 | 45.93 | 2103.36 |

In Figures 5.5 (b) and 5.5 (c), the agent’s performance at 100k timestep and 500k timestep is shown. The highlighted boxes in Figure 5.5 (c) show the places where the agent’s performance has improved.

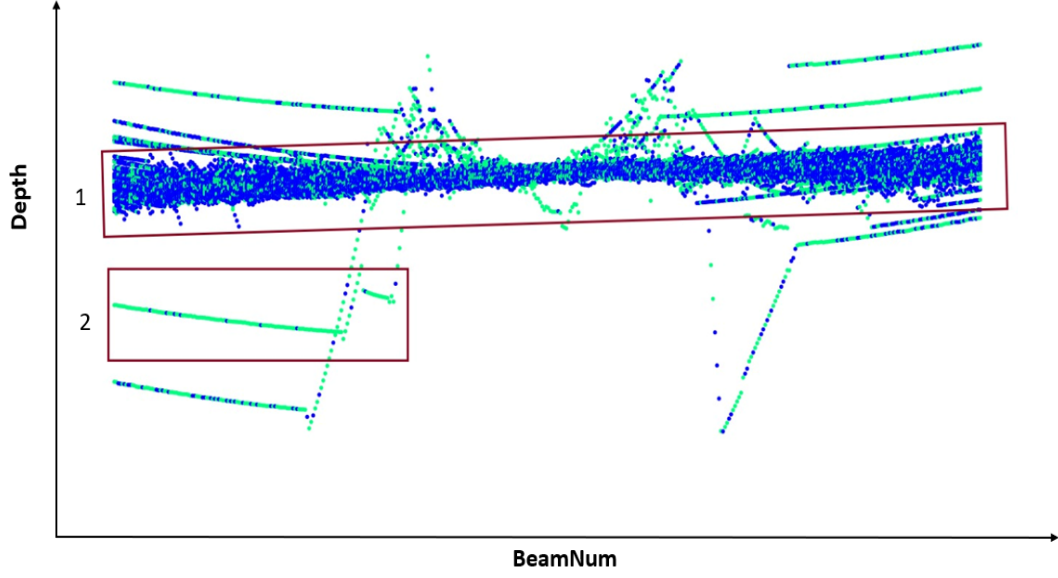
Figure 5.5 (c) shows that the agent at 500k timestep was able to identify both the seabed and the outlier more accurately than the agent at 100k timestep. This difference in the ability to identify seabed better is indicated in highlighted box 1, and the difference in the ability to identify outlier better is indicated in highlighted box 2.



(a) Manually Annotated



(b) Result at 100K Time Step



(c) Result at 500K Time Step

Figure 5.5: Dataset 4 Results

5.4.7.5 Performance of RL on Dataset 5

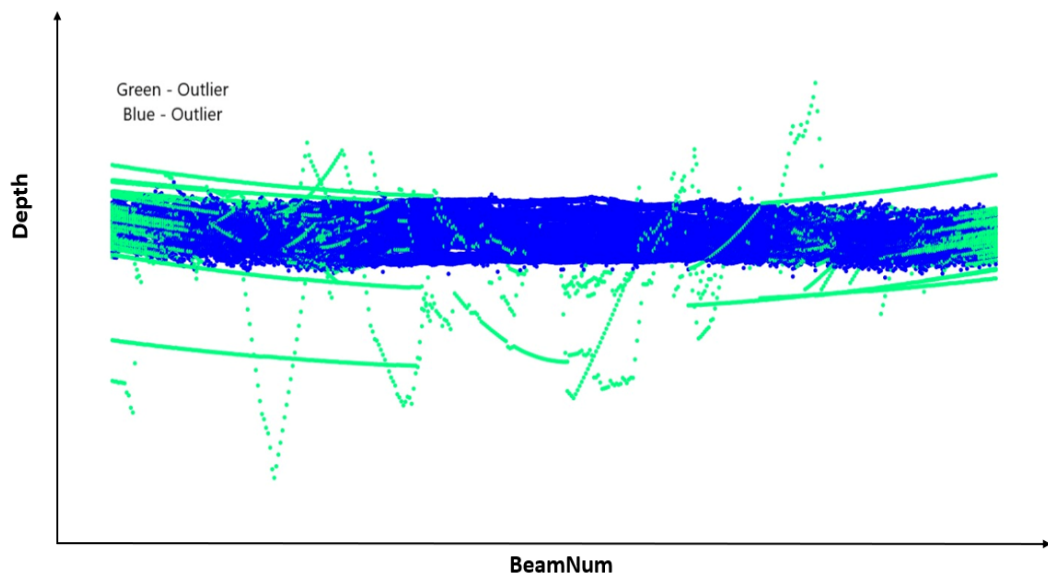
Figure 5.6 (a) shows the manually annotated Dataset 5. This dataset consists of 80 records. The green points refer to the outlier flags, and the blue refers to the inliers or the seabed features. The Reinforcement Learning agent was trained on the three timesteps, and the performance of the agent is given in Table 5.13. In this dataset, the agent performance increases from identifying 44 flags in each beam to 52 flags out of 80 records. However, upon training 1 Million times, the agent does not show any performance increase.

Table 5.13: Performance of RL on Dataset 5 Results

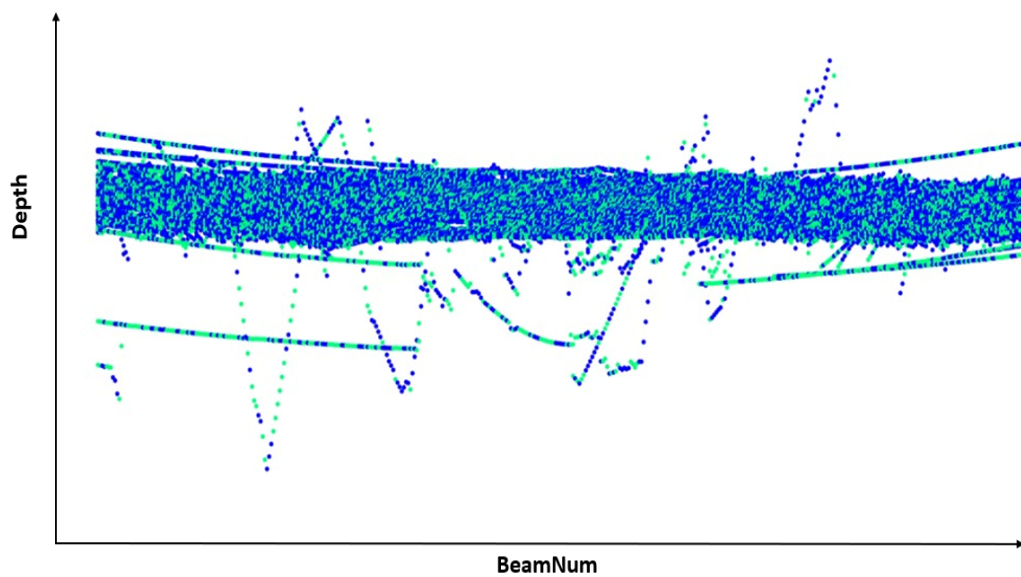
| Time Steps | Average Reward | Accuracy (%) | Precision (%) | Recall (%) | Time Taken(s) |
|------------|-------------------|-----------------|------------------|------------|------------------|
| 100k | 44 | 56.57 | 18.89 | 45.45 | 273.30 |
| 500k | 52 | 65.91 | 23.74 | 43.47 | 1161.43 |
| 1 Million | 53 | 67.79 | 22.67 | 36.04 | 2010.12 |

In Figures 5.6 (b) and 5.6 (c), the agent’s performance at 100k timestep and 500k timestep is shown. The highlighted boxes in Figure 5.6 (c) show the places where the agent’s performance has improved.

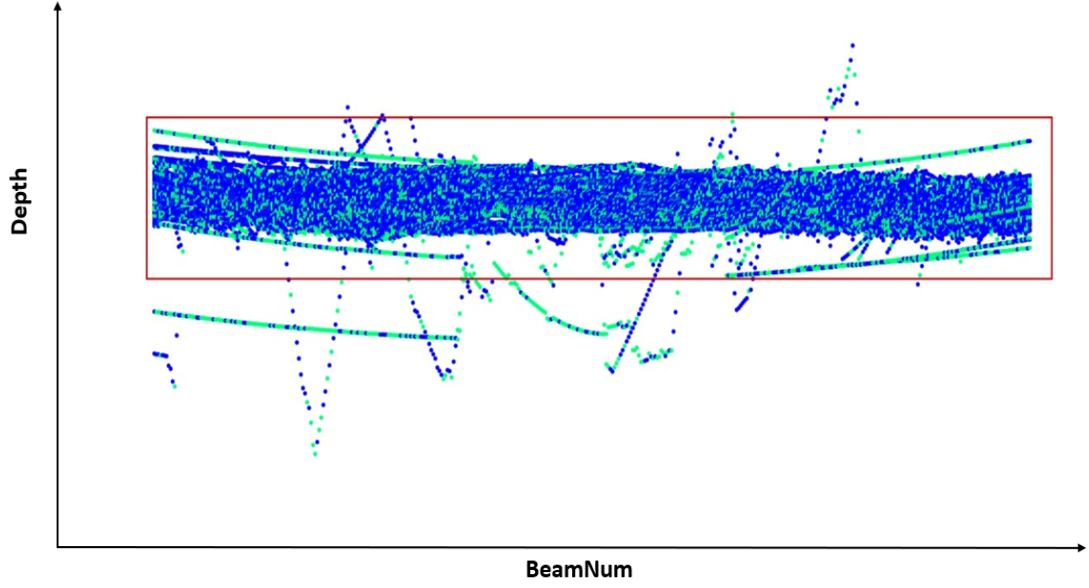
From Figure 5.6 (c), it is observed that the agent at 500k timestep could identify the seabed more accurately than the agent at 100k timestep. This difference in the ability to identify better is indicated in the highlighted box.



(a) Manually Annotated



(b) Result at 100K Time Step



(c) Result at 500K Time Step

Figure 5.6: Dataset 5 Results

From the results of all the datasets, it is demonstrated that using the OceanMappingDataframe users can develop Reinforcement Learning strategies for the hydrography data. Also, by using the OceanMappingDataframe, the users do not need to develop the Reinforcement Learning algorithm from scratch but develop the algorithm via a simple one-line API call - `create_reinforcement()` method by inputting the list of records upon which the Reinforcement Learning should be trained, the timesteps for the Reinforcement Learning and the file path to save the model. Then, the users can apply the existing Reinforcement Learning algorithm using the `apply_reinforcement()` method by inputting the file path of the saved model and the list of records upon which the Reinforcement Learning should be working.

Based on the results of the demonstration, the Reinforcement Learning agent at 100k timesteps generated only 50% of the same actions as manually annotated flags. But it showed the ability to learn the problem statement and generate actions 70% of the same actions as manually annotated flags at 500k timesteps.

Based on the demonstration and results, Reinforcement Learning has shown the potential to solve the problem of outlier detection in multibeam sonar data. This demonstration also created an opportunity to study Reinforcement Learning in more depth.

5.4.8 Concluding Remarks

In this chapter, the results of the proposed approach were discussed. First, the various I/O operations and pivot operations of the OceanMappingDataframe were evaluated and benchmarked. Later, the test results of the developed Reinforcement Learning algorithm to identify outliers in hydrography data were discussed.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The hydrography community primarily uses multibeam sonar for mapping the seabed primarily. Due to technological advancements, the data collected from the seabed is increasing, and large volumes of multibeam data are obtained. In order to make informed decisions using a large quantity of quality hydrographic survey data available, modern data science techniques must be employed to analyze the data. One of the popular data structures used for representing the geoscience data is Xarray, which uses multidimensional arrays to represent the multidimensional geoscience data. Previous approaches attempted to use the Xarray data structure to represent the multibeam sonar data.

The Xarray data structure has weak integration with data science and AI libraries, so it is difficult to build data science and AI models using the Xarray data structure directly. Dataframes such as Pandas which has tight integrations with data science and AI libraries, but do not have the ability to load large volumes of multibeam sonar data.

To aid further research in the hydrography industry by using data science and AI models, the OceanMappingDataframe, an exclusive dataframe to support multibeam sonar data, has been developed. The developed dataframe supports loading large volumes of multibeam sonar data and projecting the multidimensional data using multi-index principles, thus maintaining the strengths of the Xarray data structure and also providing tight integrations with modern data science and AI libraries. In this thesis, the proposed dataframe system design is explained. In order to demonstrate the tight integration of AI libraries, the outlier detection problem from the hydrography industry was used as an example. A Reinforcement Learning algorithm was developed using the proposed dataframe and popular AI libraries to identify outliers in the multibeam sonar data.

6.2 Future Work

In this thesis, the OceanMappingDataframe with a backend as MODIN has been developed. During the development and testing of the proposed method, it was found that the following components can be more optimized and tested.

1. Optimizing `read_gsf()` - From the benchmarking test, it is observed that the OceanMappingDataframe performs better than Pandas using the `read_gsf()` method. However, the OceanMappingDataframe can be optimized more if the pygsf reader is converted to use distributed computing principles to utilize all the cores of the CPU and use libraries that are more optimized for the MODIN architecture.

2. Adding support to other multibeam readers - In this thesis, the OceanMappingDataframe uses pygsf reader to load GSF files, but there are other popular multibeam data formats to support. The OceanMappingDataframe should add support to other format readers.
3. Improving Multi-Index operation - From the benchmarking, it is observed that MODIN consumes more time to perform the multi-indexing method. Therefore, it needs further investigation to improve the overhead.
4. Optimizing the Parquet format - From the test, it is observed that the data is larger than the original GSF file when saved in Parquet format. Future research should identify a more optimized way of storing the OceanMappingDataframe in Parquet format.
5. Developing Join Operations for MODIN - As MODIN is a newer dataframe structure, presently, MODIN does not have an implementation of the join operation. The join operation is directed to Pandas' backend, which cannot use multi-cores and distributed computing principles. So there is a need to develop a join operation for MODIN and a distributed multi-indexed and spatial join operation for the OceanMappingDataframe.

Also, during the development of the Reinforcement Learning algorithm to identify outliers in the hydrography data, the following elements were observed which requires more studies and testing.

1. Adding more variables - In the thesis, only the depth variable is used for training the agent. In the future, many more variables can be combined with the depth variable to identify outliers.

2. Designing new reward functions - In this thesis, a straightforward method is employed to evaluate the match between the manual flags and the agent's predictions, resulting in the generation of rewards. In the future, alternative methods may be tested.
3. Testing with other RL algorithms - The PPO algorithm is used in the thesis to train the agent, but in the future other algorithms can be used to build the agent. Also, the RL performance can be tested using hyperparameter tuning [12].

References

- [1] <https://www.analyticsvidhya.com/blog/2022/06/speed-up-pandas-in-python-with-modin/>.
- [2] www.kongsberg.com/contentassets/ec77c4305dcb.pdf.
- [3] *Architecture of MODIN*, <https://github.com/modin-project/modin>.
- [4] *AWS*, <https://aws.amazon.com/?nc2=hlg>.
- [5] *CARIS HIPS and SIPS*, <https://www.teledynecaris.com/en/products/hips-and-sips/>.
- [6] *CARIS MIRA AI*, <https://www.teledynecaris.com/en/products/whats-new/caris-mira-ai/>.
- [7] *Feather File Format*, <https://arrow.apache.org/docs/python/feather.html>.
- [8] *Generic Sensor Format, GSF*, <https://www.leidos.com/products/ocean-marine>.
- [9] *Google Cloud Platform*, <https://cloud.google.com/>.
- [10] *Google Colab*, <https://colab.research.google.com/>.
- [11] *HDF*, <https://portal.hdfgroup.org/display/HDF5/HDF5>.
- [12] *Hyperparameter tuning*, https://stable-baselines3.readthedocs.io/rl_zoo.html.

- [13] *IHO-S4*, <https://iho.int/en/standards-and-specifications>.
- [14] *Matplotlib*, <https://matplotlib.org/stable/users/project/citing.html>.
- [15] *Native python reader for Generic Sensor Format (GSF) files.*,
<https://github.com/guardiangeomatics/pygsf>.
- [16] *NetCDF*, <http://www.unidata.ucar.edu/software/netcdf/>.
- [17] *NOAA*, <https://oceanservice.noaa.gov/hydrography.html>.
- [18] *Noaa*, <https://www.ncei.noaa.gov/news/ncei-archive-growth-and-change>.
- [19] *Pangeo System Workflow*, <https://pangeo.io/architecture.html>.
- [20] *QPS Qimera*, <https://qps.nl/qimera/>.
- [21] *QPS Swath Editor Top View and Behind View*,
<https://www.youtube.com/watch?v=nFjvxv2AE0k>.
- [22] *Sklearn - Xarray*, <https://phausamann.github.io/sklearn-xarray/>.
- [23] *Stable-Baseline3 documentation*, [stablebaselines3.readthedocs.io/ppo.html](https://stablebaselines3.readthedocs.io/en/stable/ppo.html).
- [24] *VisPy*, <https://vispy.org/>.
- [25] *Workflow of gsflib , howpublished = https://www3.mbari.org/gsfliib03-05.pdf*.
- [26] *Xarrays*, <https://docs.xarray.dev/en/stable/>.
- [27] *Zarr*, <https://zarr.readthedocs.io/en/stable/>.
- [28] *Stable-baselines3: Reliable reinforcement learning implementations*, *Journal of Machine Learning Research* **22** (2021), no. 268, 1–8.

- [29] *Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng*, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015, Software available from tensorflow.org.
- [30] *Lars Arge, Kasper Green Larsen, Thomas Mølhave, and Freek van Walderveen*, Cleaning massive sonar point clouds, *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (New York, NY, USA), GIS '10, Association for Computing Machinery, 2010*, p. 152–161.
- [31] *Maarten A. Breddels and Jovan Veljanoski*, Vaex: big data exploration in the era of gaia, *Astronomy & Astrophysics* **618** (2018), A13.
- [32] *Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba*, Openai gym, *arXiv preprint arXiv:1606.01540* (2016).
- [33] *B. Calder and Larry Mayer*, Automatic processing of high-rate, high-density multibeam echosounder data, *Geochemistry, Geophysics, Geosystems* **4** (2003).
- [34] *Francois Chollet et al.*, Keras, 2015.
- [35] *Ian Church*, GGE6353, Heave Lecture. Fredericton: University of New Brunswick, 2022.

- [36] Hamman Joseph Ponte Aurelien Rath Willi (2019). *Proc. of the 2019 conference on Big Data from Space (BiDS'2019)* Eynard-Bontemps Guillaume, Abernathey Ryan, The pangeo big data ecosystem and its use at cnes, 2019.
- [37] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant, Array programming with NumPy, *Nature* **585** (2020), no. 7825, 357–362.
- [38] Kelsey Jordahl, Joris Van den Bossche, Martin Fleischmann, Jacob Wasserman, James McBride, Jeffrey Gerard, Jeff Tratner, Matthew Perry, Adrian Garcia Badaracco, Carson Farmer, Geir Arne Hjelle, Alan D. Snow, Micah Cochran, Sean Gillies, Lucas Culbertson, Matt Bartos, Nick Eubank, maxalbert, Aleksey Bilogur, Sergio Rey, Christopher Ren, Dani Arribas-Bel, Leah Wasser, Levi John Wolf, Martin Journois, Joshua Wilson, Adam Greenhall, Chris Holdgraf, Filipe, and François Leblanc, geopandas/geopandas: v0.8.1, *July 2020*.
- [39] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing, Jupyter notebooks – a publishing format for reproducible computational workflows, *Positioning and Power in Academic Publishing: Players, Agents and Agendas* (F. Loizides and B. Schmidt, eds.), IOS Press, 2016, pp. 87 – 90.

- [40] *Julian Le Deunf, Nathalie Debese, Thierry Schmitt, and Romain Billot*, A review of data cleaning approaches in a hydrographic framework with a focus on bathymetric multibeam echosounder datasets, *Geosciences* **10** (2020), no. 7.
- [41] *Artur Makar*, Cleaning of mbes data using cube algorithm, 06 2017.
- [42] *Wes McKinney et al.*, Data structures for statistical computing in python, *Proceedings of the 9th Python in Science Conference*, vol. 445, Austin, TX, 2010, pp. 51–56.
- [43] *Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala*, Pytorch: An imperative style, high-performance deep learning library, *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.
- [44] *F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay*, Scikit-learn: Machine learning in Python, *Journal of Machine Learning Research* **12** (2011), 2825–2830.
- [45] *Devin Petersohn, William W. Ma, Doris Jung Lin Lee, Stephen Macke, Doris Xin, Xiangxi Mo, Joseph E. Gonzalez, Joseph M. Hellerstein, Anthony D. Joseph, and Aditya G. Parameswaran*, Towards scalable dataframe systems, *CoRR* **abs/2001.00888** (2020).
- [46] *R Core Team*, R: A language and environment for statistical computing, *R Foundation for Statistical Computing*, Vienna, Austria, 2022.

- [47] *Dominic John Repici*, The comma separated value (csv) file format, 2010.
- [48] *Matthew Rocklin*, Dask: Parallel computation with blocked algorithms and task scheduling, *Proceedings of the 14th python in science conference*, no. 130-136, Citeseer, 2015.
- [49] *Leela Sedaghat, John Hersey, and Michael P. McGuire*, Detecting spatio-temporal outliers in crowdsourced bathymetry data, *GEOCROWD '13, Association for Computing Machinery*, 2013, p. 55–62.
- [50] *Guido Van Rossum*, The python library reference, release 3.8.2, *Python Software Foundation*, 2020.
- [51] *Deepak Vohra*, Apache parquet, pp. 325–335, *Apress, Berkeley, CA*, 2016.
- [52] *Fanlin Yang, Jiabiao Li, Feng you Chu, and Ziyin Wu*, Automatic detecting outliers in multibeam sonar based on density of points, *OCEANS 2007 - Europe (2007)*, 1–4.
- [53] *Eric Younkin*, Kluster: Distributed multibeam processing system in the pangeo ecosystem, *OCEANS 2021: San Diego – Porto*, 2021, pp. 1–6.

Vita

Candidate's full name: Vishwa Barathy Gandhi Kalidasan

University attended:

Bachelor of Computer Science and Engineering

SRM Institute of Science and Technology

2015-2019

Publications:

G K Vishwa Barathy, Abhishek Nath, Dipayan Das, S Niveditha, **"Visual Dialog Agent Based On Deep Q Learning And Memory Module Networks"**, IJRAR - International Journal of Research and Analytical Reviews (IJRAR), Volume.6, Issue 1, Page No pp.862-865, March 2019

Conference Presentations:

Vishwa Barathy Gandhi Kalidasan, Suprio Ray and Ian Church **"Performance Of Distributed Data Structure On MBES Data"**, Canadian Hydrography Conference, Canada, June – 2022