TOWARDS REAL-TIME SEA-FLOOR SURFACE RECONSTRUCTION AND CLASSIFICATION USING 3-D SIDE-SCAN SONAR

by

Ryan Michael Goldade

B.A.Sc. (Engineering Science), Simon Fraser University, 2009

Thesis submitted in partial fulfillment
of the requirements for the degree of

Master of Applied Science

in the
School of Engineering Science
Faculty of Applied Science

SIMON FRASER UNIVERSITY

Summer 2014

APPROVAL

Name: Ryan Michael Goldade

Degree: Master of Applied Science

Title of Thesis: Towards Real-time Sea-floor Surface Reconstruction and Classification using 3-D Side-scan Sonar

Examining Committee:

Chair: **Dr. Andrew Rawicz**
Professor

Dr. John Bird
Senior Supervisor
Professor

Dr. Mirza Faisal Beg
Supervisor
Associate Professor

Dr. Marinko Sarunic
Internal Examiner
Associate Professor

Date Approved: July 29, 2014

Partial Copyright Licence

Abstract

This thesis presents a computer algorithm to solve two major hurdles for generating real-time automated sea-floor maps with composition classification using 3-D side-scan sonar data. The algorithm consists of two distinct parts: sea-floor profiling and sea-flooring classification with computation acceleration from a graphics processing unit (GPU). The sea-floor profiling algorithm is an automated method that identifies bathymetry data corresponding to the sea-floor while ignoring bathymetry corresponding to water column objects and multi-path returns. The algorithm improves upon a fuzzy curve tracing method to handle discontinuities in the point-cloud data along the sea-floor and to discriminate between the sea-floor and other data. With an average error of 2.6% and a computation time of 7.40ms, the sea-floor profiling algorithm is extremely accurate and efficient.

Classification of the sea-floor regions consists of applying image texture methods and machine learning classifiers to side-scan sonar images. In this thesis, a feature space for each side-scan sonar image pixel is created using image texture analysis algorithms, and classified with an artificial neural network. The accuracy and performance of the algorithm is tested with side-scan sonar images from the Underwater Research Lab's Pam Rocks sonar survey. Real-time classification was achieved by the use of GPU computing. Porting the algorithm onto the GPU using OpenCL reduced the per-ping computation time to an average of 100ms, with an average error of 3.4%, making it a viable real-time solution in a sonar system.

Keywords: Side-scan sonar; fuzzy curve tracing; image texture classification; GPGPU

Acknowledgments

Although this thesis is the clu I would like to thank Dr. John Bird for his direction and feedback while allowing me to direct my own research. His encouragement to spread my own wings has been a great motivator during graduate school.

I would also like to thank Stephen Pearce and Sabir Asadov as their daily wisdom has helped me mature as a student of science and a person. Additionally, to everyone involved with the URL as your dedication to the field of sonar has provided me with the survey data necessary to write my thesis.

As an aside, I was involved with two internships during the course of this degree. I would first like to thank both Dr. Robert Bridson and Todd Keeler at UBC for their direction and supervision during the Mitacs internship with Microsoft Studios. The project results won us best poster at SCA 2013 and led to my second internship. I want to extend a thank you to Dr. Chris Wojtan at IST Austria for inviting me to join his lab in Vienna. The fully-funded trip was an amazing experience and one I will cherish.

I would also like to thank my family and friends who supported me during this journey. You were the support I needed while buried under a pile of papers.

Finally, to Monica Yau, thank you for explaining the importance of fish hybridization in a way that an engineer could understand.

Table of Contents

List of Tables

List of Figures

1. Introduction

The goal of this thesis is to solve two outstanding challenges when processing 3-D side-scan sonar data: 1. reconstructing the sea-floor profile to provide topographical information and 2. segmenting the side-scan sonar images in real-time based on sea-floor composition. Sea-floor profiling is an important problem to solve because the collected sonar data tend to contain additional data points away from the actual sea-floor. The non-sea-floor data points makes reconstructing the true sea-floor a significant challenge. Additionally, real-time sea-floor classification information will allow surveyors to identify regions of interest while still in the field so they can proceed with further exploration if necessary.

Figure 1.1 presents an example of the information flow for the sonar system used in this thesis. An acoustic pulse is sent into the water column and its corresponding echoes are received by the sonar system and processed (signal amplification, etc.). The data from the pulses are further processed to produce both a side-scan sonar image and a 2-D map that estimates where the echoes originated from (signal return point estimation). The thesis contributions deal with processing the resulting side-scan image to classify sedimentation regions and processing the signal returns to isolate the profile of the sea-floor. The result of these two processes can then be combined to create 3-D classification maps of the sea-floor surveyed by the 3-D side-scan sonar system.

The thesis contributions are also designed efficiently to execute in real-time. This performance achievement allows both methods to be integrated into a sonar system. Existing solutions are generally limited to providing this information through post-processing by human operators and time consuming algorithms. The process of creating the final 3-D classification maps was not implemented to compute in real-time and is left as future work.

In addition to computational efficiency, both methods in the thesis contribution provide accurate results. The sea-floor profiling method is almost accurate as a human operator when determining signal returns that correspond to the sea-floor. The side-scan sonar classification method has a low cross-validation error when tested with operator-classified side-scan sonar images. The operator's selections were verified with drop-camera surveys of the sonar surveyed area.

The sea-floor data presented in this thesis was obtained by a sonar system that captures topographical information about the sea-floor (referred to as bathymetry); however due to

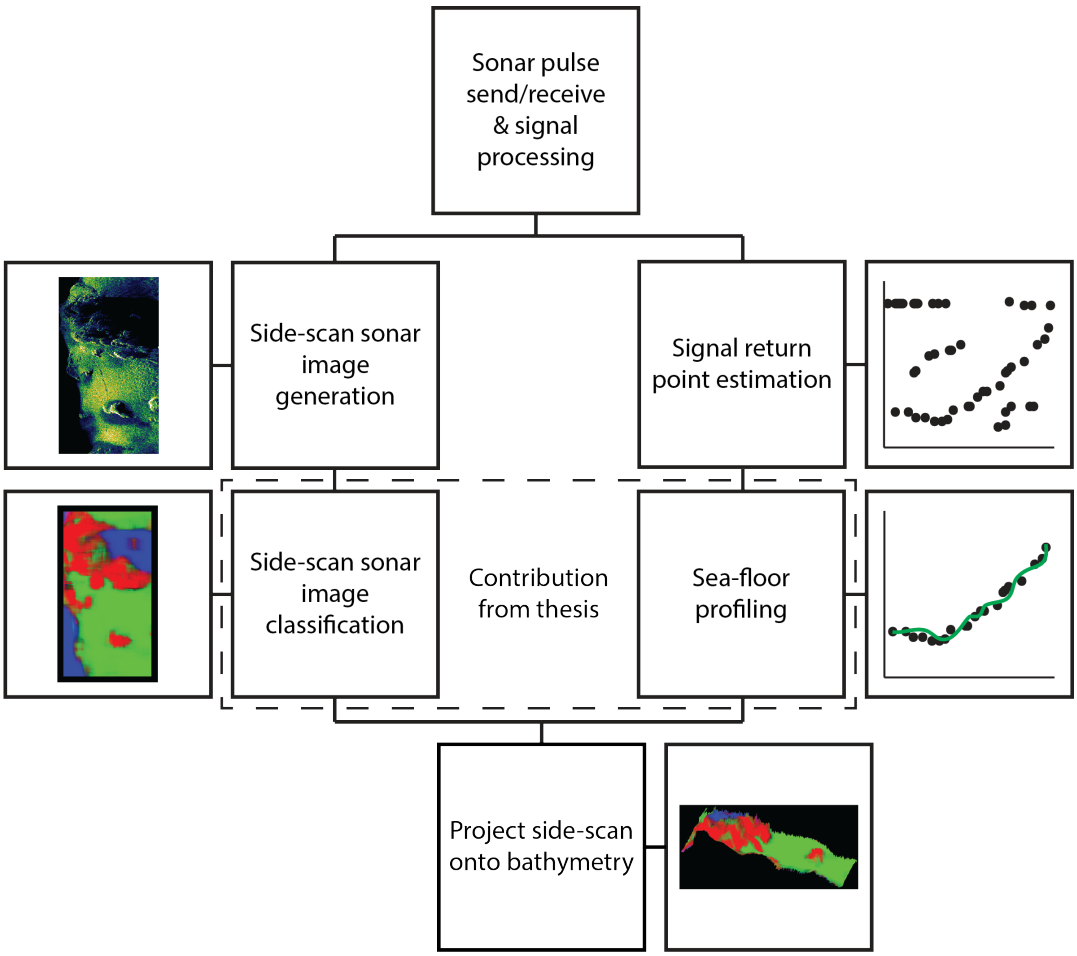

Figure 1.1: Sonar system overview with the contribution from thesis thesis indicated with a hashed-line box.

phenomena encountered in the environment (explained in section 1.1), the sonar system tends to capture additional data that doesn't correspond to the sea-floor. Although these data are completely valid, for the context of profiling the sea-floor, the non-sea-floor data will be henceforth referred to as outlier data. The outlier data complicate the system's presentation of the sea-floor during a sonar survey because interpolating bathymetry data with outliers causes a large divergence from the true sea-floor profile. Because of the difficulty of isolating the sea-floor in the bathymetry data, the on-line presentation of the sea-floor during a sonar survey is generally limited to simpler 2-D side-scan sonar images. Figure 1.2 is an example of a side-scan sonar image that would be visible to the operator during a survey with the sonar system used to collect the data in this thesis. Although such images provide detailed texture information of the sea-floor, they lack a third dimension necessary to describe the topography of the observed region, namely depth.

This thesis presents an automated solution to remove outlier data from the bathymetry

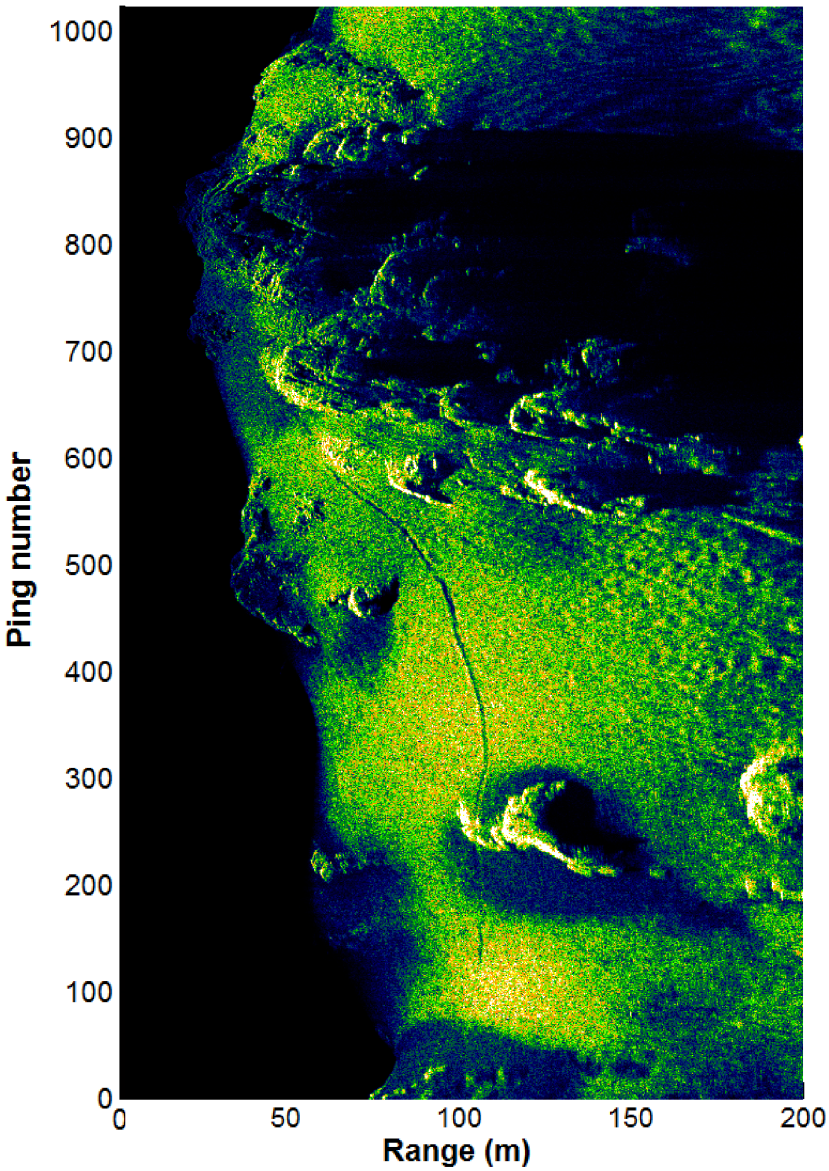

Figure 1.2: Example of a side-scan sonar image. The colour intensity is based on the backscatter strength returned to the transducer. The black regions are range values that receive no signal.

data in real-time. The absence of outlier data allows the operator to view reasonably accurate 3-D representations of the sea-floor during a sonar survey. Additionally, the side-scan image from figure 1.2 can be combined with the outlier-removed bathymetry data to represent a high quality textured 3-D representation of the sea-floor. Figure 1.3 is an example of combining side-scan and bathymetry to produce a 3-D side scan image that is capable of being generated with the solution presented in this thesis.

Several different approaches exist for sea-floor composition classification, each one of

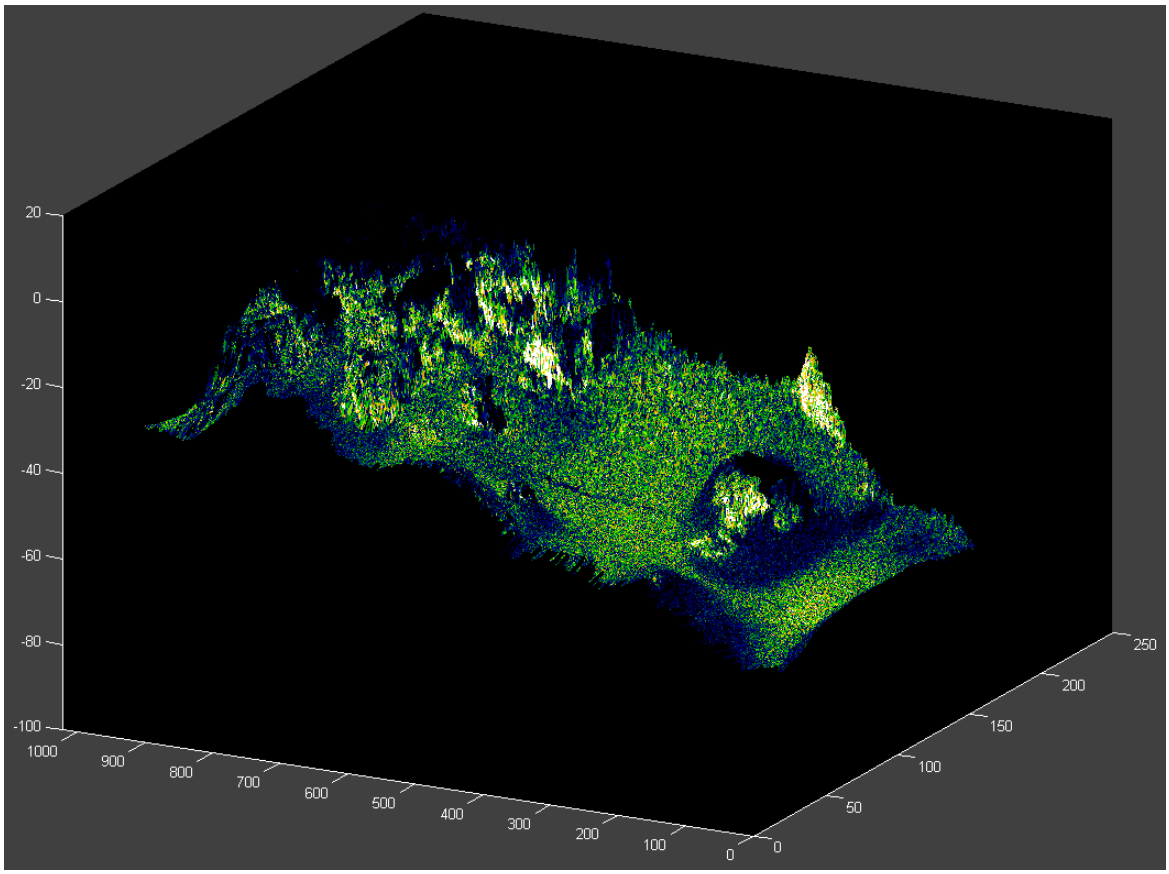

Figure 1.3: Example of the side-scan image from 1.2 projected onto the corresponding outlier-free bathymetry data.

which involves one or more tradeoffs between computation speed and survey size. Fast classification methods are capable of operating in real-time; however, they employ sonar techniques that can only survey an extremely small area at any given time. Although this application is computationally efficient, the survey-size limitation requires the survey vessel to be in operation for longer periods of time. The category of composition classification methods capable of segmenting large images, like ones produced by side-scan (figure 1.2), are accurate but extremely time consuming.

This thesis presents a technology-based solution to these computation restrictions and takes steps forward to provide survey operators with large survey-size composition classification results in real-time. Figure 1.4 presents an example of a segmented a side-scan sonar image using the sea-floor composition classification method. In this example, three different composition regions are identified and color-coded to provide a visual representation of rocky (*red*), sandy (*green*), or shadow/no-signal regions (*blue*).

To provide an operator with the ability to view a 3-D representation of the composition classification, the classified side-scan image (*left*) from figure 1.4 is projected onto the

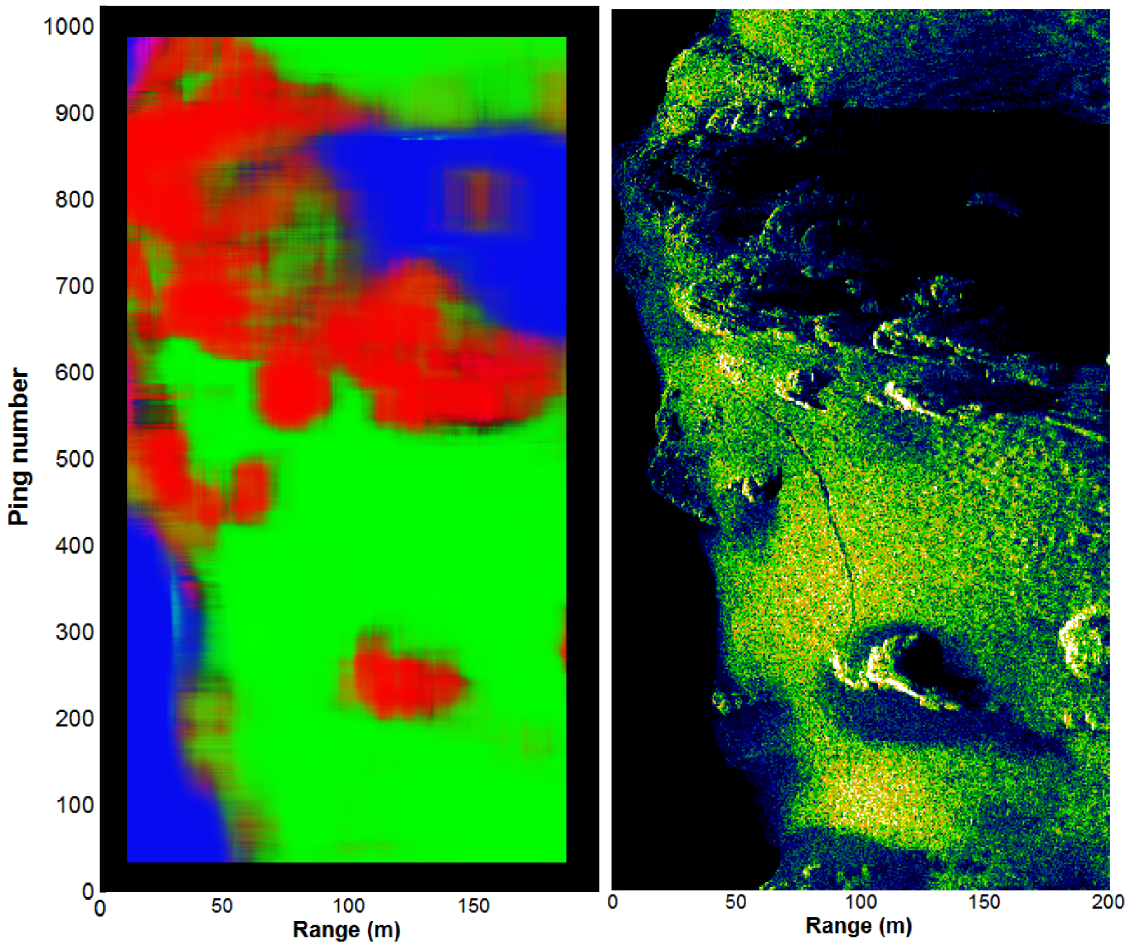

Figure 1.4: Classified side-scan image example from 1.2. The regions are segmented into rocky outcroppings (*red*), sandy regions (*green*), and shadow/no-signal regions (*blue*).

bathymetry in the same way that the original side-scan image is in figure 1.3. Figure 1.5 presents an example of a 3-D classified side-scan image. The consistency between sea-floor topography and the composition classification in figure 1.5 is indicated in the large red region because its classified as a rocky outcropping and it contains large vertical changes in the topography. Although the examples presented in this thesis only classify a small set of sea-floor composition features, further development could adapt the solutions presented herein to include the identification of more complex habitat features specific to any particular region of interest. The accuracy and performance of the methods presented in this thesis allow a survey operator to identify such regions of interest during a survey and make decisions about in-depth autonomous underwater vehicle of diver investigation immediately.

The sonar data presented throughout this thesis was collected from a survey of the Pam Rocks area conducted by the Underwater Research Lab (URL) at Simon Fraser University (SFU). Pam Rocks is a designated rock fish conservation area in Howe Sound, B.C.

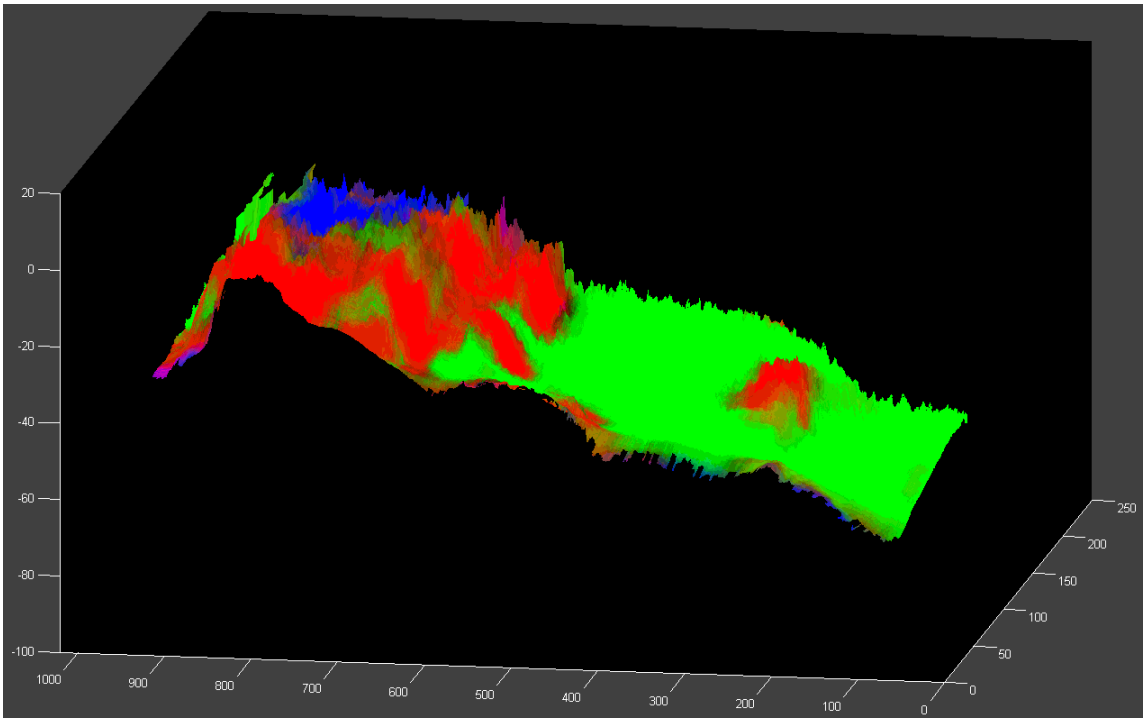

Figure 1.5: Classified 3-D side-scan example with figure 1.4 projected onto bathymetry data from figure 1.3

where fishing methods are restricted to minimize bycatch among the threatened rock fish population. The rocky terrain at this location is an ideal habitat for this species, and classifying *rocky* regions in the side-scan sonar images of the collected sonar data is a good proof-of-concept case for habitat identification. Also, the rocky terrain provides complicated topography, which makes sea-floor profiling more challenging and a good test case for a robust solution.

1.1. Sonar Techniques

The sea-floor classification and bathymetry processing solutions presented in this thesis are mainly based on image processing and machine learning techniques. However, it is important to describe the sonar system used during the URL's Pam Rocks survey to properly explain the problems that inspired this thesis and the motivation for adopting a multi-disciplinary solution for handling these problems.

The sonar system used produces an acoustic pulse with a beam that is narrow in the direction that the vessel is travelling (or *along-track* dimension) and wide in the plane that is orthogonal to the along-track dimension (*depth* and *across-track* dimensions). Once the acoustic pulse contacts the sea-floor, the *esonified region* propagates along the sea-floor with the travel of the acoustic pulse and creates acoustic echoes at each contact. This type

of sonar is described as side-looking because it surveys the sea-floor off to the side of the vessel. More technically, this type of sonar is referred to as side-scan sonar.

Figure 1.6 provides a visual example of a typical side-scan acoustic pulse (or ping). The wide swath of the beam is represented by hashed lines and the front and back of the pulse is represented with solid lines. The *star* symbol indicates the location where the front of the pulse intersects with the sea-floor and where an echo will be generated. Both the pulse and the intersection location will continue to propagate in the direction indicated by the arrows, creating a stream of echoes as time proceeds.

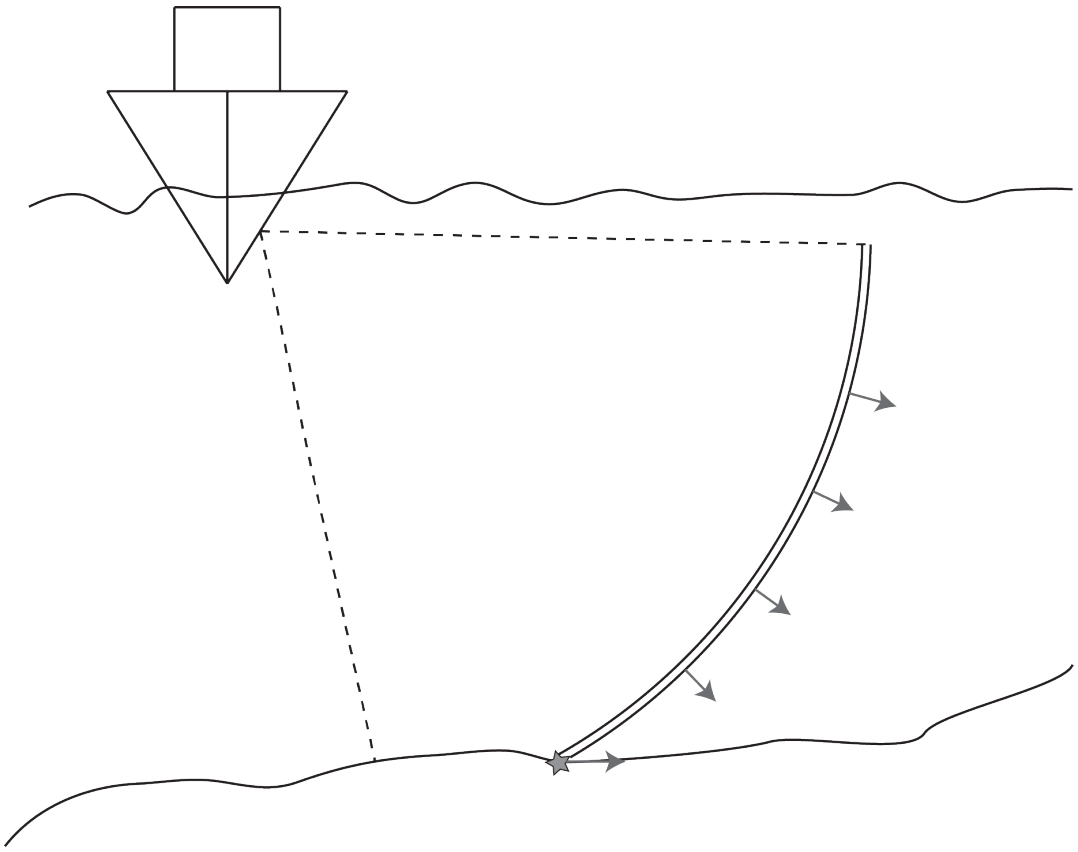

Figure 1.6: Side-scan sonar swath example. An acoustic beam is project outwards from the side of the vessel as a swath (*hashed lines*) that is narrow in along-track and wide in across-track. The acoustic pulse (*solid lines*) contacts the sea-floor (represented with a *star* symbol) and creates a series of echoes as the pulse propagates along the sea-floor.

A side-scan sonar system measures the amplitude of a returned echo for a particular moment in time. Because side-scan is directed at an entire region of the sea-floor, the amplitude of the echoes from the entire survey area are captured. Each discrete return represents an interaction with the sea-floor at a particular distance (or *range*), which is determined by the time delay from sending a ping to receiving the ping's echoes (or *signal*

returns). Collecting signal returns from a side-scan sonar ping results in a 1-D strip of amplitude values corresponding to the echo (or *backscatter*) strengths as the esonified region propagates along the sea-floor. By compiling a series of ping returns as the vessel moves in the along-track direction, a 2-D side-scan image of the sea-floor is created (see figure 1.2).

Side-scan sonar images can capture high-resolution detail of the sea-floor. The variations in the signal strength that create these details are determined by the acoustic scattering of the sea-floor regions as they are esonified. For example, the homogenous region in the middle of figure 1.2 corresponds to a diffuse-scattering sandy composition. The highly variant return strengths in the top-left quarter of figure 1.2, in contrast, correspond to specular-scattering rocky outcroppings. The returned signal strength also depends on the grazing angle from the transducer to the plane or the sea-floor. These variations in scattering properties of the sea-floor, combined with different grazing angles, create the variations in the side-scan images that describe the details of the sea-floor. Even small details, such as the thread-like anchor scar through the middle of figure 1.2, can be captured with side-scan sonar.

Although side-scan sonar captures high-resolution detail from sequential signal returns, traditional methods cannot resolve the direction from which a signal return arrives. With only range and along-track distance information, side-scan sonar can only create a 2-D representation of the sea-floor. The angle-of-arrival (AOA) of the signal returns is required to create a 3-D representation of the sea-floor and provide topographical information. In addition, without the ability to resolve AOA of a signal returning to the transducer, side-scan sonar cannot discriminate between multiple signal returns arriving at the transducer simultaneously from different directions. Figure 1.7 presents a situation where an acoustic pulse contacts multiple objects at the same range. Multiple simultaneous ping-object intersections will result in multiple signal returns arriving at the transducer at the same time, but from different directions. The inability to resolve AOAs for multiple signal returns is a very serious drawback because the resulting amplitude value for that range value will be a blend of signal returns originating from a multitude of locations. An advanced version of side-scan sonar, known as 3-D side-scan sonar, largely solves the problem of simultaneous signal return arrivals and provides bathymetry data in addition to high-resolution side-scan sonar images.

1.1.1. 3-D Sidescan Sonar

There are two generic sonar techniques that resolve the AOA information from a signal return (in addition to range): multi-beam and multi-angle. Although both methods create an acoustic pulse in the same sense as side-scan sonar, they differ in the way the transducer receives the returned signals. An explanation of the multi-beam sonar method is beyond the scope of this thesis, however interested readers should refer to [1].

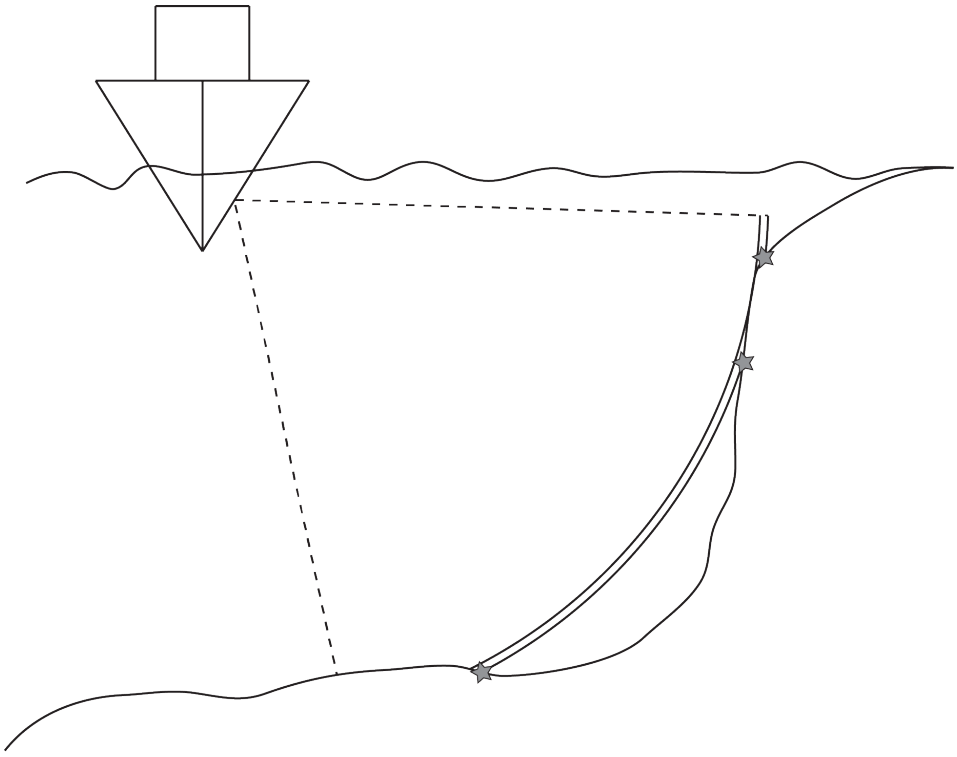

Figure 1.7: Multiple ping contacts with the sea-floor example. The acoustic ping interacts with the sea-floor at multiple locations (*stars* symbols). The multiple interactions with create echoes that will all arrive back at the transducer at the same time.

The sonar system used to collect data for this thesis was designed by researchers in the URL [2, 3]. This side-scan sonar method employs multiple piezoelectric elements, oriented in a vertical stack, in a single transducer. This sonar system resolves AOAs by the phase difference of a received acoustic pulse between elements in the transducer. Figure 1.8 presents a simple example of AOA estimation, where the transducer consists of two elements and the acoustic pulse is represented by a plane wave.

In the example presented in figure 1.8, the acoustic pulse is in contact with both transducer elements but at different phases in the acoustic pulse. This phase difference, θ, is referred to as electric angle. For the case where the plane wave and the transducer face are parallel, there will be no phase difference between transducer elements and the electric angle is zero. Conversely, when the plane wave and the transducer face are orthogonal, the phase difference is directly based on the element separation distance, d, and the wavelength of the pulse, λ, and the electric angle is $\theta_{\perp} = \pm\frac{2\pi d}{\lambda}$. The transducer design used in the URL sonar system has an element spacing of $d = \frac{\lambda}{2}$, so the electric angle θ is bounded on the interval $[-\pi, \pi]$ [4, 2].

Using the process of AOA estimation for a signal return, combined with range of the

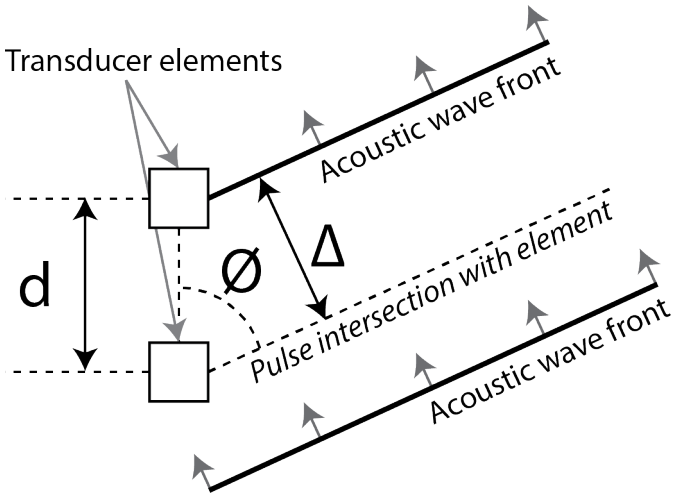

Figure 1.8: Angle-of-arrival example. The separation of the two intersections, Δ, is the physical representation of the phase difference (or electric angle), θ, by $\Delta = \frac{\theta\lambda}{2\pi}$. The physical angle-of-arrival, ϕ, is determined by $\phi = \sin^{-1}(\frac{\Delta}{d})$.

signal return (determined by the send-receive time delay), the multi-angle side-scan sonar system captures 2-D information for each ping. Because a sonar survey consists of a series of sequential pings, the system appends the electric angle and range data from subsequent pings in the along-track dimension to create three dimensional bathymetric side-scan data. From this point on, this system is referred to as 3-D side-scan sonar.

The positional data (i.e. without signal amplitude) from 3-D side-scan sonar is collected and stored in terms of electric angle and range for each ping. However, an operator might find it easier to visualize a 3-D representation of the sea-floor when it is expressed in a cartesian-coordinate representation, such as: across-track distance, along-track distance, and depth (x-,y-,z- axis in cartesian space, respectively) for the signal return data points[1].

Before the bathymetry data can be represented in cartesian space, the phase difference of a signal return must be described in terms of its physical angle with respect to the transducer. The physical angle, ϕ, of a signal return's plane wave with respect to the transducer face (from figure 1.8), is determined from the physical representation of the electric angle, $\Delta = \frac{\theta\lambda}{2\pi}$, by:

$$\phi = \sin^{-1}(\frac{\Delta}{d}). \tag{1.1}$$

The following two equations describe the process of converting a signal return's physical angle, ϕ, and range, r, into across-track, x, and depth, z, positions:

[1]Please note that although the z-axis is referred to as depth throughout this document, a location below the sea-level is expressed with a negative z coordinate value.

$$x = r\cos(t_\phi + \phi), \tag{1.2}$$

$$z = -r\sin(t_\phi + \phi) + d_t, \tag{1.3}$$

where t_ϕ and d_t represent the tilt angle and depth of the transducer relative to sea-level, respectively. Figure 1.9 is an example of the data displayed in the two coordinate spaces. The top example describes the bathymetry data of a single ping in electric angle and range. The bottom example describes the equivalent x-,z- axis representation by applying (1.1), (1.2), and (1.3) to the data in the top example. Both examples also include the sea-level boundary (red dotted line), the transducer position (black dot), and the bathymetry data (blue dots) to provide a visual picture of the relationship between the two coordinate spaces.

The AOA estimation from figure 1.8 only considers the trivial case of a single signal return contacting the transducer at a given time. The configuration presented in figure 1.8 cannot determine the AOA for multiple signal returns because there must be enough elements (i.e. degrees of freedom) present in the transducer to solve a linear system of unknown arrival angles. A transducer would need $n+1$ elements to resolve n simultaneous AOAs. Although an in-depth technical explanation of this method is beyond the scope of this thesis (interested readers should refer to [4, 2, 3]), the consequences of this limitation is important to understand. For any instance of $n+1$ or more simultaneous signal returns, the linear system will be undetermined and, with infinitely many solutions, the AOAs cannot be resolved. The URL sonar system employs six elements, so it is capable of resolving the AOA for five simultaneous signal returns. However, only two or three AOAs are resolved at any given time because, in order to minimize possible transducer issues, the URL sonar system uses a subarray method. The subarray method averages signal data between elements with the benefit of improving the accuracy of the resolved AOAs, however also with the downside of reducing the system's degrees of freedom. There are usually enough elements to capture sea-floor bathymetry, except for an area of the sea-floor known as a nadir region [5, 6].

The concept of the nadir region will be explained through a thought experiment. If a plane wave contacts a planar boundary who's normal is orthogonal to the plane, there will be infinitely many points of contact between the wave and the boundary. Figure 1.7 demonstrates that multiple contacts at the same range from the transducer will result in multiple signal returns at the transducer at the same time. By combining the plane wave/normal example with the multiple contacts example, there will be infinitely many signal returns arriving at the transducer due to the plane wave contacting the parallel planar boundary. As explained in the above paragraph, infinitely many signal returns at the transducer will result in an underdetermined system for which there is no solution. The local wavefront of an acoustic ping is approximated as a plane wave, and regions of the

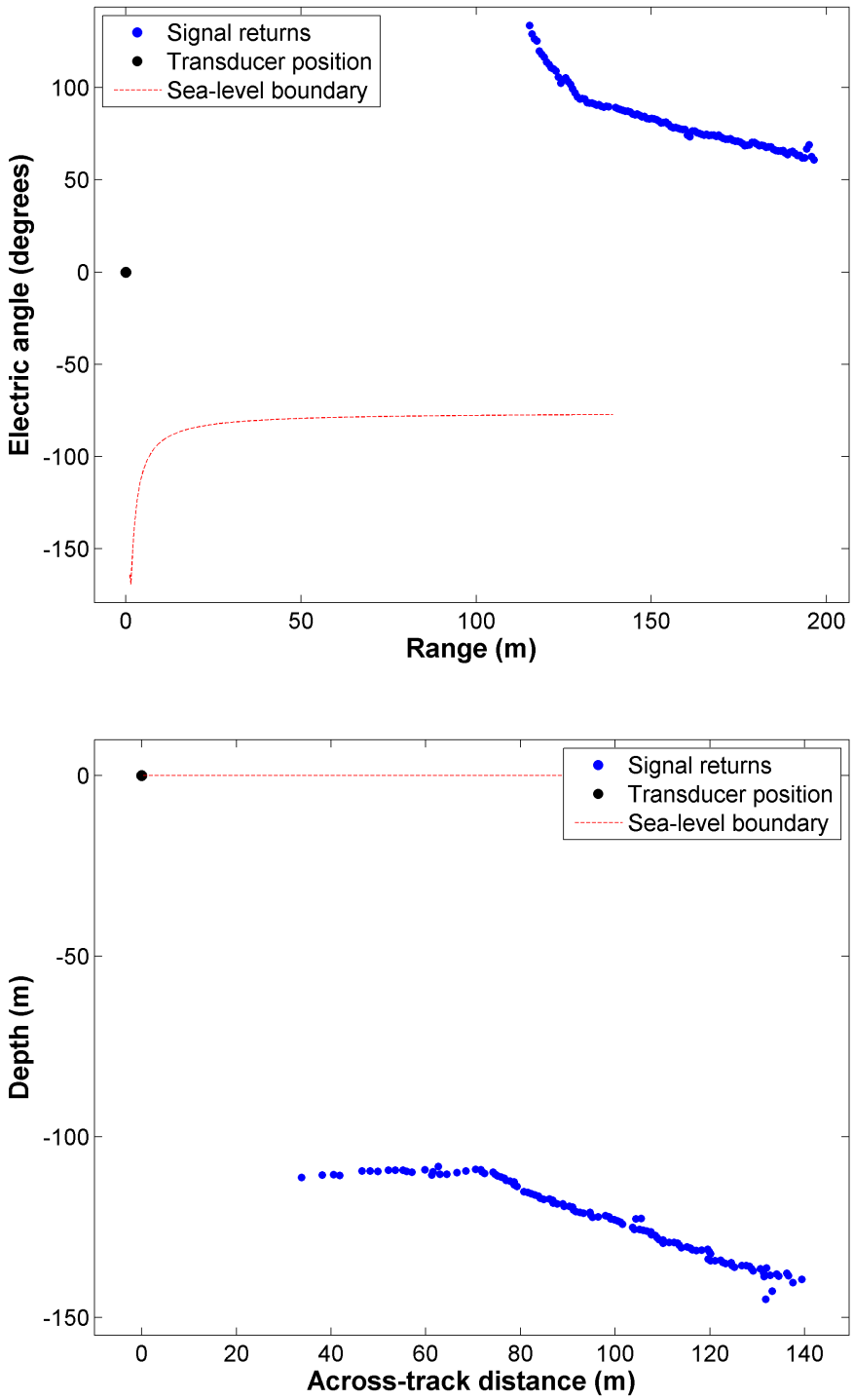

Figure 1.9: Example of bathymetry data represented in electric-angel/range (*top*) and across-track/depth (*bottom*) coordinate spaces.

sea-floor with normals orthogonal to the plane wave cannot be resolved through the angle estimation method. This region is referred to as nadir, and is visually observed as a gap in the bathymetry data.

Another continuity problem in producing sea-floor topography with 3-D sidescan sonar is acoustic shadows. In the same sense as line-of-sight in optics, sonar cannot resolve topography beyond the line-of-sight from the transducer. The shadows are represented by dark regions in side-scan sonar images (like the top-right dark region of figure 1.2) and by discontinuities in the sea-floor profile.

Figure 1.10 presents two examples of bathymetric profiles with a nadir region (top) and a shadow region (bottom) in cartesian coordinates. In both cases, there is a significant gap in the representation of the sea-floor. These discontinuities from nadir and shadow regions are typically resolved by interpolation methods. While this is not ideal, the general behaviour of the sea-floor is still reasonably approximated. There are, however, additional phenomena present in bathymetry data that exist as outliers from the true sea-floor and complicate sea-floor profiling via interpolation.

Up to this point, the only bathymetry examples presented are acoustic pulses that contact the sea-floor and the corresponding backscatter signals return directly to the transducer. However, it is quite common to observe signal returns that don't directly correspond to the sea-floor. These instances complicate the bathymetric data because they are not consistent with the sea-floor profile and exist as outliers.

One complication occurs when an acoustic ping encounters reflective objects in the water column. Additionally, because the air-water boundary creates a large change in acoustic impedance, acoustic pings tend to reflect directly back to the transducer from a rough water surface. These instances are referred to as surface-returns. Smooth surfaces tend to reflect the acoustic ping back towards the sea-floor, which create additional return signals from the sea-floor direction.

Another common complication occurs when an initial backscatter signal from the sea-floor contacts one or more additional reflective surfaces before eventually returning to the transducer. The signal returns resulting from multiple reflections are referred to as multi-path. Because the sonar system has no knowledge about the route of a multi-path signal, the bathymetry data for a multi-path signal is represented by the final arriving electric angle (and therefore physical angle) and by half of the total distance travelled (for a direct signal return, half of the total distance is the distance of the object from the transducer). Therefore, multi-path is represented in a bathymetric profile as a ping that encounters an object at a range value that consists of the entire length of the path taken to return to the transducer. Additionally, only the final AOA of the signal return can be represented in the bathymetric profile. Of course this isn't actually what happened, but the system has no means to resolve the full path and the signal returns appear as outliers from the true

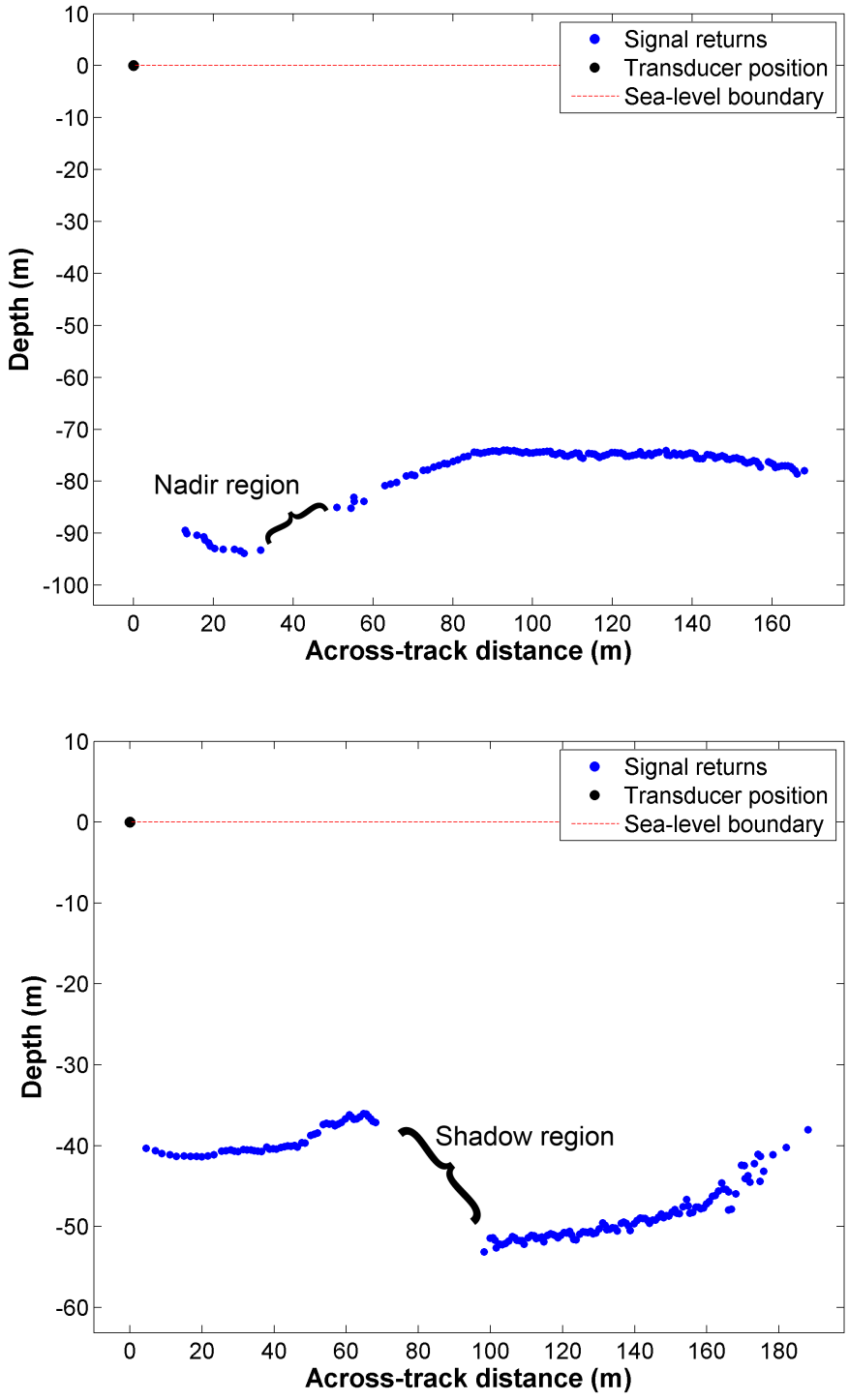

Figure 1.10: Bathymetric profile examples with nadir (*top*) and shadow (*bottom*) regions.

sea-floor profile.

Multi-path does not only occur when a signal return intersects the sea-floor multiple times, but also, like surface-returns, from intersections with the water surface. In the case of the multi-path signal reflecting off the water surface and back to the transducer, the system records signal returns arriving from above the water surface. In another example, the multi-path signal might reflect off the sea-floor, off the ocean surface, then off the sea-floor again and to the transducer. This results in the appearance of echoes of the sea-floor profile in the bathymetry data below the true sea-floor, as the AOAs are the same as the true sea-floor, but the range is greater due to the indirect path taken.

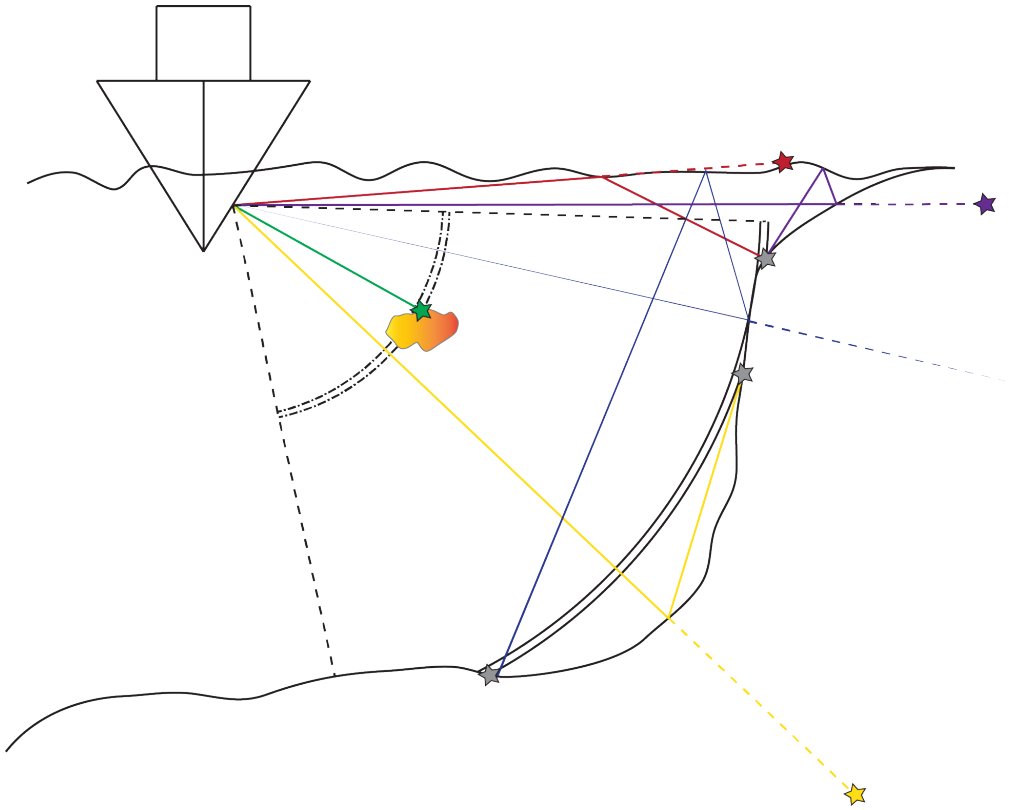

Figure 1.11: Outlier signal returns. Green: water-column target. Red: surface multi-path. Purple: shallow water multi-path. Yellow: sea-floor multi-path. Blue: attenuated multi-path

Figure 1.11 presents various examples of potential paths taken by backscatter signals that create the non-sea-floor returns mentioned above. The initial intersection of the ping with the sea-floor is indicated by grey stars, while the final return positions resolved by the sonar system are labelled by colored stars. The coloured dashed lines indicate the projection along the AOA corresponding to the range computed by the sonar system. In the instance of the blue multi-path return, the signal strength is attenuated significantly before the signal reaches the transducer. This attenuation is due to signal scattering at each boundary

collision by the signal (signal amplitude is also lost due to spherical spreading while the ping propagates through the water-column but this is corrected for in post-processing). Thankfully, this attenuation prevents multi-path from more than a few consecutive scatters from being collected at the transducer.

The purple path in figure 1.11 demonstrates the problem with multi-path in shallow water. The final position of the purple multi-path signal return is not sufficiently far away from the true sea-floor profile and it can be difficult to identify as an outlier. The signal-to-noise ratio (SNR) for bathymetry data in shallow regions tend to drop as a result of multi-path and the confidence in the location of the sea-floor can become undesirably low.

Figure 1.12 presents the full bathymetric profiles of example pings in figure 1.10. As with the original figures, the nadir and shadow regions are indicated. Additionally, the instances of multi-path above and below the surface, as well as surface-returns, are also indicated. Figure 1.13 presents, in similar fashion, entire bathymetric profiles that also contain, in addition to multi-path and surface-returns, a water-column object (top) and a low SNR region (bottom). Finally, the green line for each example in both figures 1.12 and 1.13 indicate the interpolated solution for the sea-floor profile across nadir and shadow regions and in the presence of outlier data. Unfortunately, these outlier data create substantial challenges for producing accurate interpolations of the sea-floor profiles because the existing system cannot easily sort the sea-floor signal returns from the outliers in a bathymetric profile.

Despite the challenges faced while extracting the sea-floor profile, the 3-D side-scan sonar technique is an important survey technology because it is a substantially simpler design than the multi-beam method; a method that is also prone to these types of outlier data. Additionally, multi-beam methods cannot produce the same image quality typically found in a side-scan system of similar frequency (frequency directly impacts spatial resolution) [7]. Because of the image quality limitation of multi-beam sonar, sea-floor classification methods traditionally combine multi-beam and side-scan sonar to produce bathymetry and high quality side-scan images. This combination is extremely expensive and limits accessability to sea-floor classification surveys for marine ecologists and non-industry-funded science. However, the URL sonar system is very cost effective and it provides both bathymetry data and high quality side-scan images from a single unit.

The following section discusses work relevant to isolating the sea-floor in bathymetry data and classifying regions of sea-floor through side-scan sonar images.

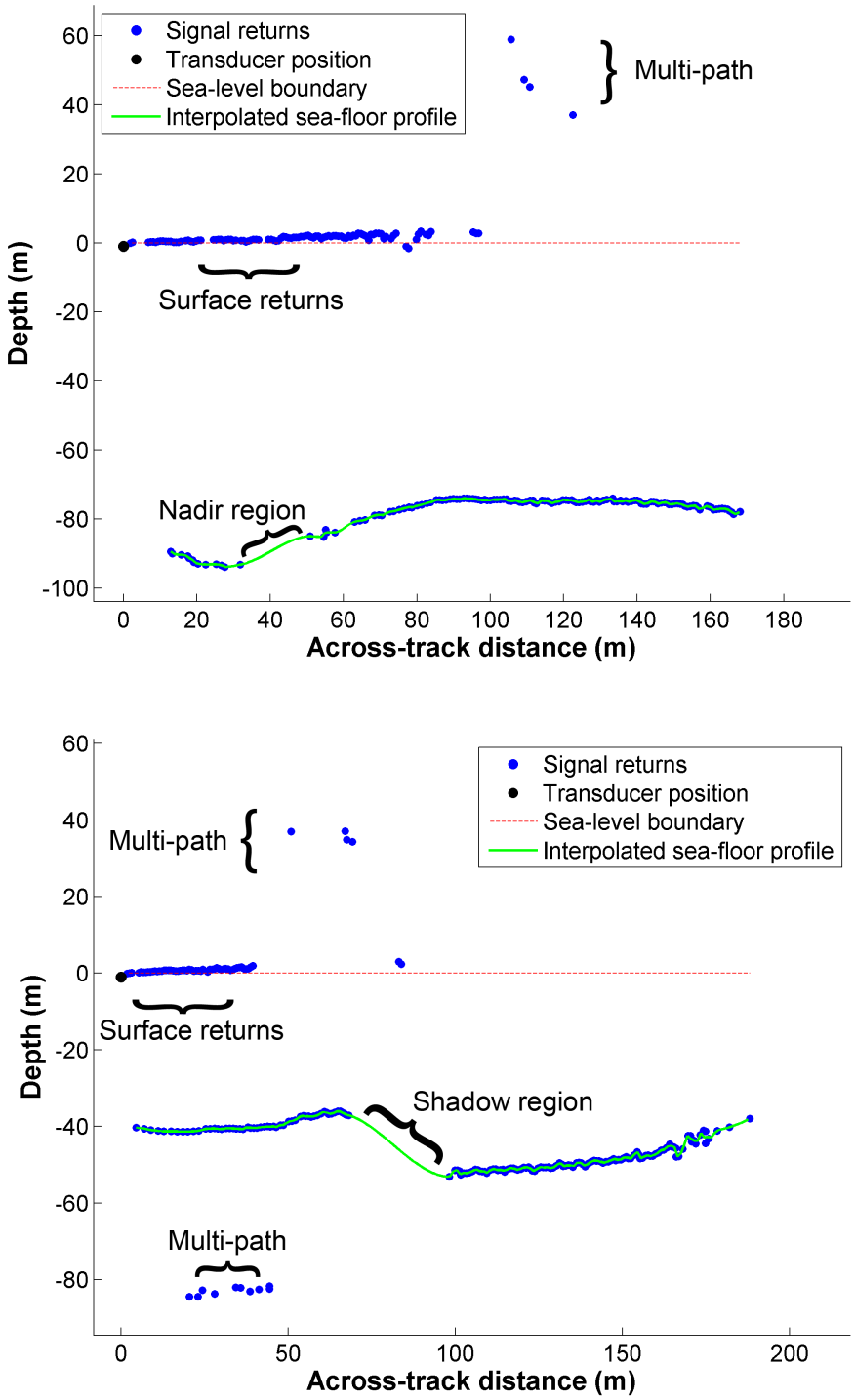

Figure 1.12: Bathymetric sonar ping examples with interpolated sea-floor profiles, labelled nadir and shadow regions and outlier data.

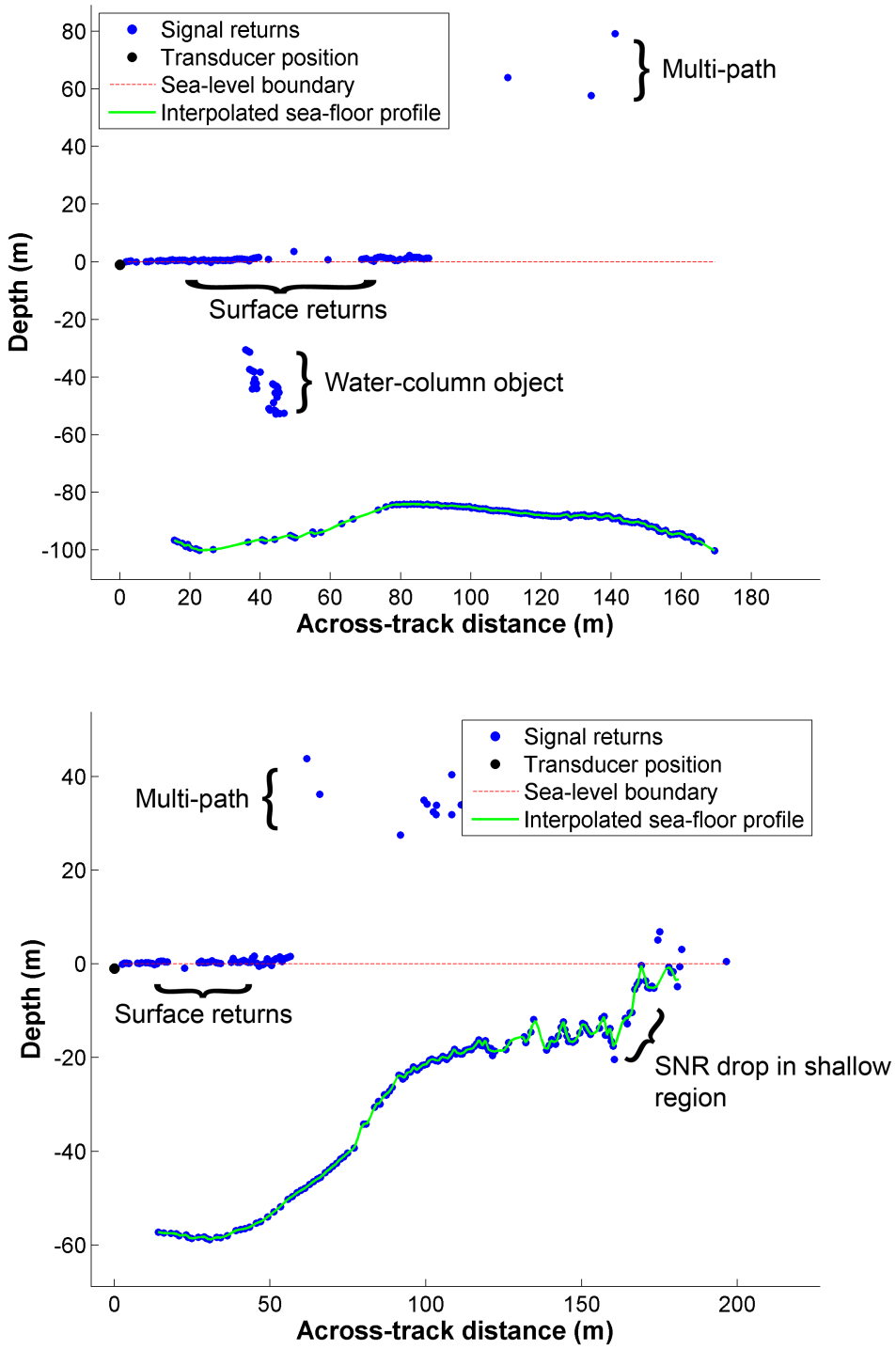

Figure 1.13: Bathymetric sonar ping examples with interpolated sea-floor profiles, labelled nadir and shadow regions and outlier data.

1.2. Related Work

1.2.1. Sea-floor Profiling Related Work

The challenge of accurately profiling the sea-floor from bathymetry data with outliers and discontinuities remains an open problem in sonar research. Generally, the bathymetry data is sorted per-ping by an operator before creating a 3-D interpolation of the sea-floor from the remaining data points. Given the industry-driven nature of the sonar field, it is possible that potential solutions to this problem exist but remain trade secrets. To the best of the author's knowledge, there is no published research that attempts to automatically resolve the sea-floor profile given outlier bathymetry data. This section provides a brief overview of related techniques in sonar, computational geometry, stochastic modelling and machine learning as a survey of potential methods to adapt for outlier invariance.

The goal of sea-floor profiling is similar to surface reconstruction problems in computational geometry. The bathymetry data consists of a series of points and the goal is to reconstruct the continuous sea-floor surface from the point-cloud data. Similar surface reconstruction problems exist with 3-D range scanners that use laser light for remote sensing instead of sound with sonar. Edelsbrunner et al. introduced one of the earliest surface reconstruction methods using a Delaunay triangulation method to build a triangle mesh out of a point cloud [8]. However, this and subsequent methods generally suffer from jagged features with noisy data. Implicit surface approaches are typically less susceptible to localized noise, as demonstrated by two seminal methods using radial basis functions [9] or solving a poisson problem [10] to build a level set surface from point-cloud data. These reconstruction methods are also capable of filling holds in the point cloud, which is a desirable solution for resolving gaps due to nadir and shadow regions. However, because 3-D scans are usually performed in a controlled environment, surface reconstruction methods only address local noise invariance. Sorting outlier data from the sea-floor would still be an outstanding issue that must be addressed prior to using these methods. Interested readers should refer to [11] for a thorough review of the prevailing techniques in point-cloud surface reconstruction.

Existing sonar research related to surface reconstruction generally attempt to extract or enhance sea-floor bathymetry data through side-scan sonar imaging. Cervenka and Moustier present a solution for resolving nadir regions in bathymetry by extrapolating the sea-floor profile into the nadir region using side-scan-based range information (side-scan data does not have a nadir region) [12]. This method was only developed to resolve nadir regions directly below the vessel, using the first side-scan signal return as the "range" directly below the vessel to interpolate through. Johnson et al. and Coiras et al. present methods for surface reconstruction purely from side-scan data. They borrow from shape-from-shading techniques in image processing and apply it to side-scan images to approximate the sea-floor surface. The shape-from-shading method generally assumes homogenous mediums, where

changes in reflection intensity should directly correspond to changes grazing angle in the reflection model of the material. These methods essentially invert the reflection model of the sea-floor image intensity, and use an optimization scheme to solve for the sea-floor gradient [13, 14, 15]. This approach suffers from drawbacks due to the accuracy of reflection models and the need to appropriately segment and model the various sedimentation regions present in a side-scan sonar image. Generally speaking, the side-scan image must be segmented by composition with each composition material's reflection model known a priori for these techniques to provide reliable pseudo-bathymetry data. Finally, Lu and Oshima present a stereo-vision approach to this problem by combining multiple side-scan image surveys of a region from different locations [16]. The results from Lu and Oshima are compelling but require significantly longer survey times to cover each region twice.

The previous two paragraphs discuss processing an entire collection of 3-D side-scan ping data at once to reconstruct the surface, like figure 1.3. However, sequentially processing bathymetric data from individual pings in 2-D, like the examples in figures 1.12 and 1.13, is also a reasonable approach for isolating the sea-floor bathymetry data. Also, by processing each ping independently, a sequence of solutions can be streamed to an observer during a survey, instead of after an entire collection of data.

A standard method for fitting a linear model to a data set similar to 2-D bathymetry data is L2-regularized least-squares. Because of the outlier data in the system, however, these curve fitting methods are not likely to be robust and reliable enough to accurately approximate the sea-floor profile. A robust curve fitting method known as Random Sample Consensus (RANSAC) was developed to alleviate this problem [17, 18]. This method randomly samples from the data, fits a predetermined model to the sampled data and tests the entire data set against the fitted model. Data points that fit within an error tolerance are considered true data points, and if the number of these points is over a specified threshold, that particular model is selected as the desired model. This process is repeated several times to refine the data points that are consistent with the inliers. However, this method assumes that the regularized model is known a priori and that the inliers sufficiently out number the outliers. Because the sea-floor curve and outlier count is unpredictable, the above two conditions might not be guaranteed. For this reason, this method for robust least-squares curve-fitting is left as future research.

Finally, the field of curve tracing has shown promise when trying find an optimal piece-wise linear curve that approximates the true function with noisy point-cloud data. An initial approach by Bezdek and Andersons used a c-varieties method to find the best fitting polygons (or a closed piece-wise linear curve) to approximate the boundaries of point-clouds [19]. In a similar approach, fuzzy clustering methods were used to determine circle and elliptical boundaries [20, 21] . However, these approaches did not generalize well to an unpredictable environment. Yan presented his fuzzy curve-tracing algorithm to handle

a wide variety of curves described by noisy point-cloud data [22]. His results accurately trace through arbitrary curved and spiral regions by linking together fuzzy c-means cluster centers. Although his data set only consists of continuous curves with localized noise, this method can be extended to handle multiple curves. Yan also follows up with further analysis proving that, with proper parameter settings, the iterative process in the fuzzy curve tracing method will converge even with complicated curves [23]. Subsequent research focuses on handling self-intersecting curves and sharp corners, two issues that are generally not found in baythmetry data [24].

1.2.2. Side-scan Sonar Image Classification Related Work

The second component of this thesis aims to classify composition regions of the sea-floor in real-time during a sonar survey. Classification of sonar images was largely born out of the mid 1980s with a method of applying spectral analysis from Fourier transforms to side scan images [25, 26]. They segmented regions of the image based on the standard deviation of the spectral power distribution. This application is motivated by the understanding that classifying regions simply based on backscatter strength is inaccurate because changes in grazing angle and environment variability will impact backscatter strengths in homogenous regions. Reed and Hussong countered by creating feature vectors with the spatial grey-level dependence method (SGLDM [27, 28]) to use in side-scan sonar image segmentation [29].

Stewart et al. presented and followed up with comparative studies on various image texture and supervised machine learning methods [30, 31]. They compared spectral, adapted grey-level difference (GLDR) [32], and grey-level-run-length (GLRL [33, 34]) feature spaces. The authors demonstrate that a hybrid feature space combining aspects from GLDR and GLRL used with an artificial neural network classifier was an optimal solution for segmenting side-scan sonar images.

In the same time frame, Alexandrou and Pantazartis introduced statistical analysis on backscatter distribution models to discriminate between sea-floor regions [35]. This method was quickly supplanted by more accurate methods but introduced the concept of unsupervised self-organizing maps – an alternative form of backpropogation with artificial neural networks that does not require supervised training.

In the same year, Laine and Fan as well as Chang and Kuo introduced two similar wavelet-based image texture analysis algorithms [36, 37]. The motivation for this method was to create a resolution-invariant solution for texture analysis. Although they didn't compare results with already established methods mentioned above, they demonstrated very impressive performance with accurate classification by cross-validating their datasets. In some cases they reported close to 100% accuracy.

Tang and Stewart later followed up by the comparing the wavelet packet method to spectral analysis for sidescan sonar [38, 39]. The authors report that their specialized

fourier transform feature set compares in accuracy to the wavelet transform method for added noise on the Vistex dataset as well as inherently noisy sidescan sonar. Cochrane and Lafferty, however, further validate the use of the SGLDM method to segment their sidescan sonar dataset [40].

Tang later presented a reduced set of GLRL features that minimized correlated dimensions [41]. He was able to achieve results comparable to SGLDM and results better than wavelet packets. He suggests that the divergence from almost exact accuracy in [36, 37] is a result of an insufficient window size when sampling the image, which he was able to reconcile by increasing the sampling window size. This paper is also noteworthy because it is one of the first papers to suggest a parallel computation option for performance improvements with GLRL.

More recent image texture methods such as complete local binary pattern (CLBP) or complex networks have been shown to accurately describe features of images in the Vistex image database [42]. This database consists of a wide range of images with different textures in order to test the generality of an image texture method. However, to the author's knowledge neither image texture method has been directly applied to side-scan sonar classification. This thesis focuses on a graphics processing unit implementation of methods that have already been demonstrated to accurately classify side-scan images.

Marsh and Brown present a method of combining multi-beam bathymetry and sidescan image texture for classification [43]. Using a self-organizing map neural network, they were able to present compelling results. They present an eight-class color-coded classified sidescan draped over a 3-D bathymetric survey. However, because they are using unsupervised learning, they have no prior knowledge that there are indeed eight distinct sediment types present in the survey. Since there was no ground truthing performed, it is difficult to extract the accuracy of the system. This is a major drawback of unsupervised methods; even without prior knowledge for training a classifier, one still needs to know how many classes are present in their system. The addition of bathymetry is an interesting idea but the authors admit that near-nadir regions can cause problems and they don't directly correlate with image textures. Including bathymetry infomation into the feature vector for classification is beyond the scope of this project, but it's an idea that could be expanded on in future work.

1.3. Thesis Motivation and Contributions

As mentioned in the beginning of the chapter, the goal of this thesis is to create tools to integrate into a 3-D side-scan sonar system that provide real-time solutions for accurate sea-floor profiling and composition classification. The accuracy of the sea-floor profiling method is determined by comparing the resulting signal returns with the sea-floor signal returns sorted by a human operator. The accuracy of the composition classification method

is determined comparing the label a classifier assigns to a sample region of a side-scan image with the label a human operator assigned to the same image. The human operator determined the label of the side-scan image by comparing side-scan images with drop camera surveys of the same region. The motivation for these tools arise from the overall expense of performing a sonar survey because such an expense can limit the quality and depth of a study. This thesis presents an overall method that, once integrated into an existing 3-D side scan sonar system, can provide survey operators with large scale bathymetry and composition information of a sea-floor region while still in the field. These improvements provide surveyors with information to make accurate decisions regarding AUV or diver deployment for further investigation while on-board the vessel and without having to return to a computer station to post-process the data.

1.3.1. Real-time Sonar Systems

For a sonar system, a real-time process is qualified as a process that completes within the time it takes for a ping to be sent and fully received. That is, all data from ping 1 must be processed before the last signal return from ping 2 is received. If this condition is not met, the system has lag and cannot be labelled as real-time.

The URL sonar system used to collected the data for this thesis was operating out to a range of 200 meters. Using an approximate value for the speed of sound in seawater of 1480m/s and a travel distance of 400 meters (sound has to travel to and from target), acoustic pings can be generated at 3.7Hz[2]. This means the tools presented in this thesis need to complete within 270 ms to qualify as real-time.

1.3.2. Curve-tracing Bathymetry

As presented in the previous section, bathymetric profiles must be processed to accurately represent the sea-floor. For trivial cases with only sea-floor data containing only a small amount of noise, the true topography can be approximated by interpolation methods or piece-wise linear regression. In the general case, due to the outliers in the data, traditional curve fitting methods are not properly equipped to handle these various complications. Additionally, more advanced methods that are robust to outlier data are time consuming and converge unpredictably. This thesis presents a curve-tracing solution using a machine learning approach to profile the sea-floor, in real-time, from a bathymetric profile with outliers.

1.3.3. Image-texture Classification

Side-scan sonar classification using an image-texture feature space and a machine-learning classifier is an established solution, but it lacks the computational efficiency for real-time op-

[2]Please note that the ping rate estimation captures the worst-case scenario. The ping rate might be lower in practice due to existing ping processing bottlenecks on the vessel computer station

eration during a sonar survey. This thesis proposes a technological solution for texture-based feature classification in real-time during sonar surveys by the use of general-purpose computing on graphics processing units (GPGPU). The emergence of GPGPU-based computer programming has created a new paradigm in real-time operations due to the accessability to hundreds of threaded cores optimized for massive parallel processing. As an additional motivation for the GPGPU design, this technology is easily available on modern laptop computers with discrete graphics processing units (GPU). Because the central processing unit (CPU) in the sonar system is largely dedicated to processing and storing the acquired data for each ping, a GPU-based solution does not impact the processing time of existing methods in the sonar system.

The GPGPU-based library developed for this thesis is capable of segmenting the sea-floor texture information, in real-time, from figure 1.2 to produce the composition classification results in figure 1.4

1.3.4. Contributions

This thesis brings forward the contribution of:

1. A novel ray-casting method to filter outlier bathymetry data.
2. An adapted curve tracing algorithm applied to sea-floor profiling for autonomous 3-d mapping.
3. A real-time, texture-based sea-floor classification library by the use of GPGPU.

1.4. Thesis Organization

The remainder of this thesis is comprised of the following four chapters: Chapter 2 presents a modified curve-tracing algorithm to profile the sea-floor in bathymetry data with outlier data and discontinuities. Chapter 3 discusses a hybrid image-texture feature space that is used to classify sea-floor composition in side-scan sonar images as well as the technology used to create a real-time library. Chapter 4 provides experimental results and analysis for both methods to justify the label of accurate and real-time sonar tools. Chapter 5 concludes the thesis with a discussion on future work and usability of the tools presented.

2. Sea-floor Profiling

This chapter presents a solution for profiling the sea-floor from bathymetry data containing water-column objects, multi-path and surface-return outliers, low SNR in shallow water, and sea-floor discontinuities. Figure 2.1 introduces an example (*top*) of a typical bathymetric profile captured by the sonar system used in this thesis. Sorting the bathymetry data to produce the interpolated sea-floor profile, demonstrated in the bottom example of figure 2.1, is a challenging task to automate. Human eyes can easily extrapolate the contour of the sea-floor even in the presence of discontinuities and outliers, but this is an extremely difficult task for a computer system. Additionally, sonar system operators also possess prior knowledge about the sea floor topography that assist them in sea-floor profiling. Automating this process is a challenge that this chapter solves with the use of a curve-tracing method that profiles the sea-floor nearly as accurately as a human operator, with minimal computation time [22, 44].

The following sections describe a solution pipeline to determine an approximate sea-floor profile from original bathymetry data. The pipeline is divided into three distinct modules (highlighted in figure 2.2), labelled as: pre-processing filter, angle ray-casting, and fuzzy curve tracing (FCT). The first module, pre-processing filter, sorts the prior bathymetry data using prior knowledge based on position and signal strength to remove outlier data. Because the signal returns positioned above the sea-level boundary in figure 2.1 could not possibly correspond to the sea-floor, the boundary filter removes these data and only outputs the signal returns below the boundary. The remaining signal returns are passed into the amplitude filter, which removes any signal returns that are low in signal amplitude (usually corresponding to outlier data). The output from the pre-processing filter module serves as an input to both the angle ray-casting and fuzzy curve tracing modules.

The second module, angle ray-casting, applies the prior knowledge that during a sonar survey, a position on the sea-floor can only correspond to a single return at the transducer. Therefore, a sonar survey should only receive one signal return at the transducer from the sea-floor for any given physical angle. Multiple signal returns that are well separated along a particular angle indicate the existence of outlier data along that angle. Identifying which of the signal returns along the same angle correspond to outlier data is challenging, so the angle ray-casting method simply removes all the signal returns along the particular angle.

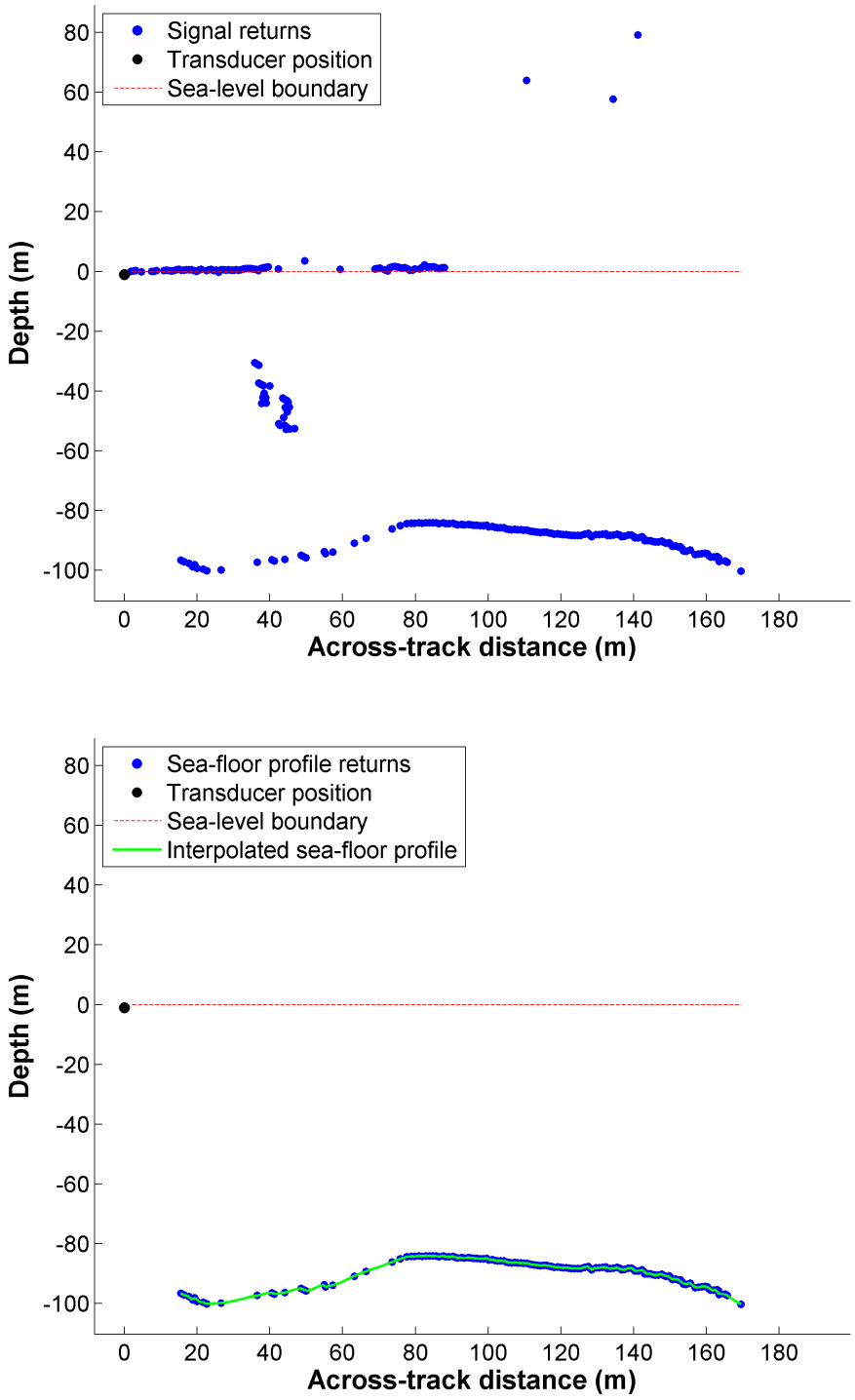

Figure 2.1: Sea-floor profiling input/output example. Original bathymetry data from the sonar system (*top*) and the final sea-floor profile using the solution presented in this chapter(*bottom*).

This technique is capable of removing the majority of outlier data, with the drawback that some sea-floor data are removed as well. Because not all outlier returns are removed and not all sea-floor returns are preserved, angle ray-casting is not a direct solution for profiling the sea-floor and to produce the final result from figure 2.1. Despite these drawbacks, angle ray-casting is used to initialize the clustering method for the FCT module because this angle ray-casting technique is extremely fast to compute and its results provide a nearly-converged initialization for the FCT cluster centers.

The final module, fuzzy curve-tracing, traces the curves of the sea-floor profile contained in the bathymetry data. The bathymetry data in this instance is the output from the pre-processing filter module. This curve-tracing method re-purposes the fuzzy c-means (FCM) clustering algorithm to approximate the sea-floor profile with the FCM cluster center positions [22]. After the cluster centers converge on to bathymetry data, they are linked together to form chains that approximate the sea-floor profile. Loops in the chain tend to form during this linking stage so a few chain maintenance steps are executed to clean up the chains. In the trivial case, the sea-floor is accurately profiled after the chain maintenance is complete.

The data set tested in [22] is comprised of only these trivial cases, so the FCT method presented in the paper doesn't handle more complicated problems caused by outlier data or discontinuities. The set of non-trivial bathymetry data tend to introduce multiple chains, with complicated loops in these chains and discontinuities between chains. Breaking such loops in a chain tend to introduce additional discontinuities and result in a sequence of broken chains. The gaps between chains require a joining scheme to attach the chains that appropriately profile the sea-floor without connecting to chains that profile outlier data. The chains are first joined by projecting outwards from the end of one chain and checking if the projection passes close to another chain. The remaining chains are then joined by determining if the break in the chains is caused by a shadow region in the bathymetry between the chains.

The result of the whole process is a set of one of more cluster chains, with only one chain that profiles the sea-floor and the remaining chains profile outlier data regions. The chain that actually profiles the sea-floor is identified from the group of chains by a heuristic that tests: the average distance between the chain and the bathymetry data that corresponds to the chain, the length of the chain, and the number of clusters in the chain. The optimal FCT chain determined by this criteria is identified as the chain that traces the sea-floor profile, and the bathymetry data associated with the chain are identified as the sea-floor profile bathymetry data. The interpolated results in the bottom example of figure 2.1 is the result of the entire system described in figure 2.2.

The following sections explain the modules from figure 2.2 in detail.

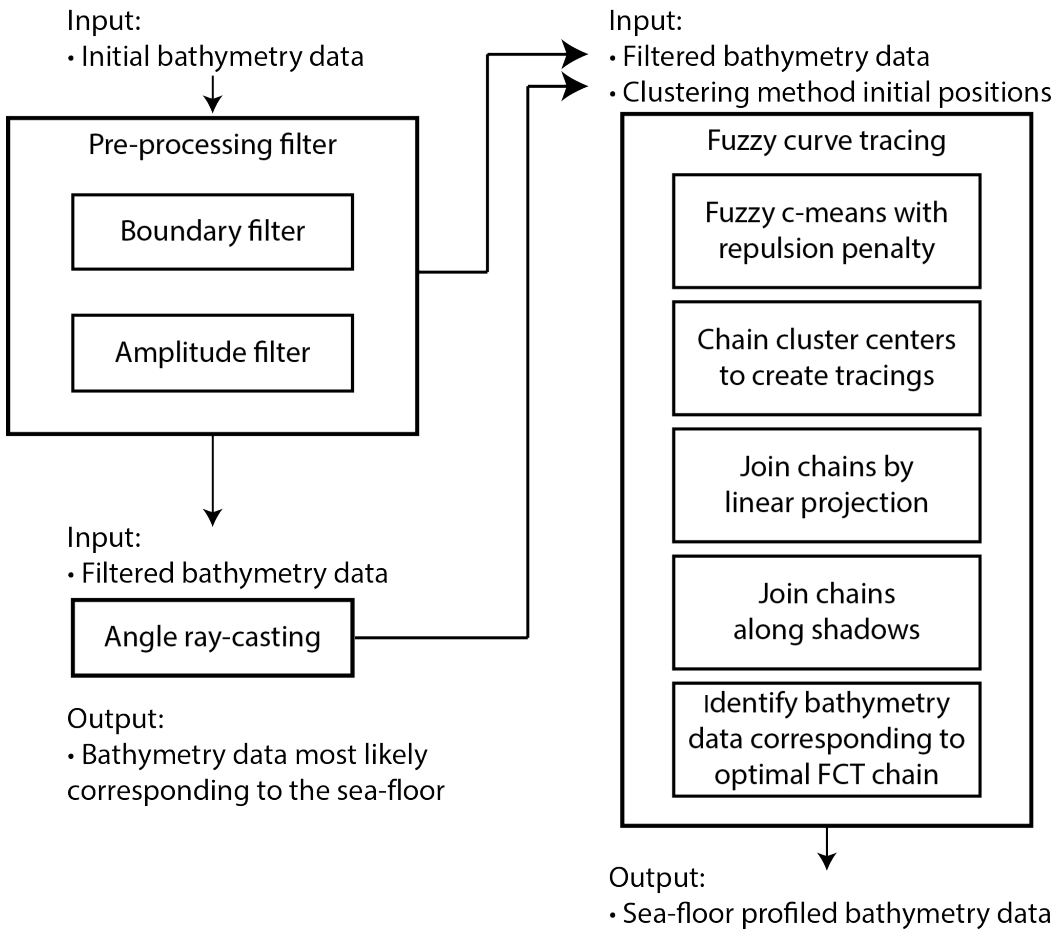

Figure 2.2: Flowchart of the sea-floor profiling algorithm. Bathymetry data captured by the sonar system is input to the algorithm. The pre-processing filter removes the simple outlier data. The pre-processing filter output is the input to the angle ray-casting and FCT modules. The output from angle ray-casting initializes the cluster center positions for the FCT module. The final output is set of signal returns corresponding to the sea-floor profile.

2.1. Pre-processing Filter

The initial step for successfully profiling the sea-floor from bathymetry data with outliers, begins by removing the simplest forms of the outlier data. The proceeding subsections explain two methods that reduce outlier data and assist the process of profiling the sea-floor through more advanced methods.

2.1.1. Sea-level Boundary Filter

The initial bathymetry data in figure 2.1 contain instances of data points that reside along or above the sea-level boundary. In both cases, these data points do not correspond to the sea-floor and should be removed from the bathymetry. This filtering process compares

the depth (z-axis) of the bathymetry data against a particular boundary, d_c, and removes the data from the bathymetry if they are above the boundary. Equation 2.1 describes this filtering technique, where $\mathbf{x_i}$ represents the i-th signal return in the bathymetry data, $x_{i,z}$ represents the z-axis component of $\mathbf{x_i}$, d_c represents the boundary cutoff, and $\emptyset$ indicates that the signal return is removed from the data set. The filter application in (2.1) is applied to every signal return in the bathymetry data.

$$\mathbf{x_i} = \begin{cases} \mathbf{x_i}, & \text{if } x_{i,z} \leq d_c \\ \emptyset, & \text{if } x_{i,z} > d_c \end{cases} \tag{2.1}$$

The boundary level is set slightly below sea-level because the surface-return data tend to spread slightly above and below the sea-level boundary. A deeper boundary level could potentially remove more surface-return data, however it could also remove actual sea-floor data in shallow water. To balance both cases, the boundary level for this system is set to $d_c = -1m$.

Figure 2.3 presents an example of this method applied to the input bathymetry from figure 2.1. In this example, the filter boundary, d_c, is represented by the black dotted line, the removed signal returns by the green dots, and the preserved signal returns by the blue dots. The preserved signal returns are passed into the amplitude filter for further processing.

2.1.2. Amplitude Filter

The signal return amplitude of the bathymetry data tend to be lower for surface-return, water-column target, and multi-path returns than for sea-floor returns. This attribute is used to remove additional outlier data by removing low amplitude signal returns from the bathymetry data. An amplitude threshold is used to remove all bathymetry data with signal amplitudes that fall below it.

The output from amplitude filtering must be consistent between different surveys even though sea-floor backscatter strength, transducers, and other equipment may differ. To address this concern, the amplitude threshold for data removal is determined independently, per ping, based on the amplitude distribution of the bathymetry data. Because the amplitudes of the outlier data will also likely be outliers in the amplitude distribution, the mean and standard deviation of the distribution will be biased by the outlier influence. This method describes the amplitude distribution with median instead of mean and median absolute deviation instead of standard deviation because they are robust to outlier influence.

Equation 2.2 describes the amplitude filtering technique, where $\mathbf{x_i}$ represents the i-th signal return in the bathymetry data, $x_{i,a}$ represents the amplitude component of $\mathbf{x_i}$, d_a is a multiplier to control the amplitude cutoff, and $\emptyset$ indicates that the signal return is removed from the data set. The robust statistics used in (2.2) are the median amplitude of the

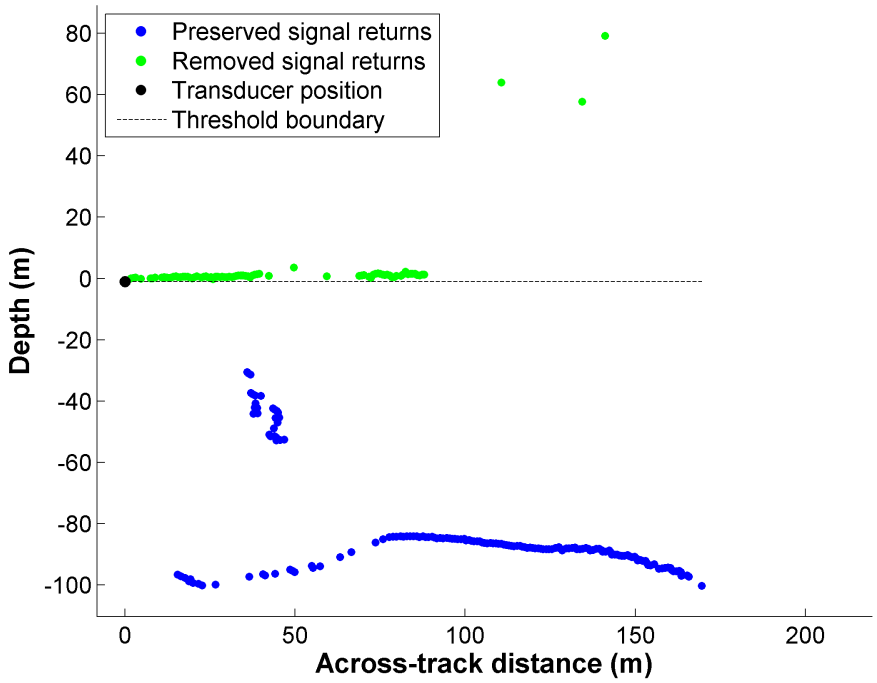

Figure 2.3: Sea-level boundary filter example. The bathymetry data above the sea-level boundary (*black dotted line*) are removed (*green dots*) and the ones below are preserved (*blue dots*).

bathymetry data, indicated by $\mu_{1/2}$, and the median absolute deviation of the amplitude of the bathymetry data, indicated by $\sigma_{1/2}$. The filter application in (2.2) is applied to every signal return in the bathymetry data.

$$\mathbf{x_i} = \begin{cases} \mathbf{x_i}, & \text{if } x_{i,a} \geq \mu_{1/2} - d_a\sigma_{1/2} \\ \emptyset, & \text{if } x_{i,a} < \mu_{1/2} - d_a\sigma_{1/2} \end{cases} \tag{2.2}$$

Figures 2.4 and 2.5 present color-coded examples of the amplitude filtering applied to bathymetry data with multi-path and water-column outliers, respectively. Both examples compare the original bathymetry from the boundary filter output to three d_a values.

The color legend on the right side of the each plot (for both figures) shows the amplitude representation of the data points, with 0 being the lowest amplitude value and 1 being the strongest. The different d_a values for figure 2.4 all manage to remove the multi-path data points, however $d_a = 1$ also begins to remove sea-floor data (which should be preserved). Figure 2.5 exemplifies further challenges when choosing an appropriate threshold value. The water-column object creates signal returns with amplitude values with similar amplitude values as the sea-floor region directly below it. This similarity is observed where the water-column signal return amplitudes and the sea-floor data amplitudes are colored as shades

of blue; indicating weak amplitude values based on the color legend. In this situation, the water-column object can be removed with a small d_a value, but at the cost of removing important sea-floor data. The filtering technique can remove some outliers, however it's clear that amplitude filtering alone cannot accurately remove all outlier data and preserve the sea-floor data. The goal of this thesis is to create an accurate representation of the sea-floor so preserving sea-floor bathymetry data is essential and a conservative d_a value is preferable. A multiplier of $d_a = 3.0$ is used in for the system because, as figures 2.4 and 2.5 indicate, it preserves the sea-floor data and still removes some outlier data.

These pre-processing methods demonstrate that simple techniques are capable of removing some outlier data but they can't fully discriminate between sea-floor and outlier signal returns. However, their execution times are extremely fast, so even a small improvement of the bathymetry data is valuable. To further separate the sea-floor bathymetry from the outliers, a more advanced process is required.

2.2. Angle Ray-casting

This section presents a novel filtering algorithm to remove outlier data but, unfortunately, at the cost of sea-floor data. Although the perseveration of sea-floor data was important in pre-processing, only the general trend of the sea-floor must be preserved with angle ray-casting. This difference in sea-floor preservation is because pre-processing initializes the bathymetry data for section 2.3, while angle ray-casting initializes the cluster center positions.

Angle ray-casting filters the bathymetry data from pre-processing using the prior knowledge that: if a ray is cast from the transducer towards the sea-floor, the ray should only encounter the sea-floor once. In this sense, for a set of signal returns along a given angle-of-arrival at the transducer, there should be only one signal return in the set that corresponds to the sea-floor; any additional signal returns along the particular angle must correspond to outlier data. Therefore, multiple returns along a particular angle with a large separation distance indicate there are undesirable water-column or multi-path returns (small separation distances are likely due to local noise). Unfortunately, it is difficult to resolve which of the signal returns correspond to the sea-floor, so this method simply eliminates all returns along that particular ray.

Nadir regions create complications for this method because the angle-of-arrival from the sea-floor can't be resolved, so outlier data along the corresponding angle might not be removed. In this case, the outlier data along an AOA with nadir will have a large difference in range values from the sea-floor signal returns with the closest AOA. To resolve this problem, the outlier is removed if there is a large jump in the range values between the data point and its closest neighbour (with respect to angle).

The majority of the bathymetric profiles presented in this thesis, up until now, are

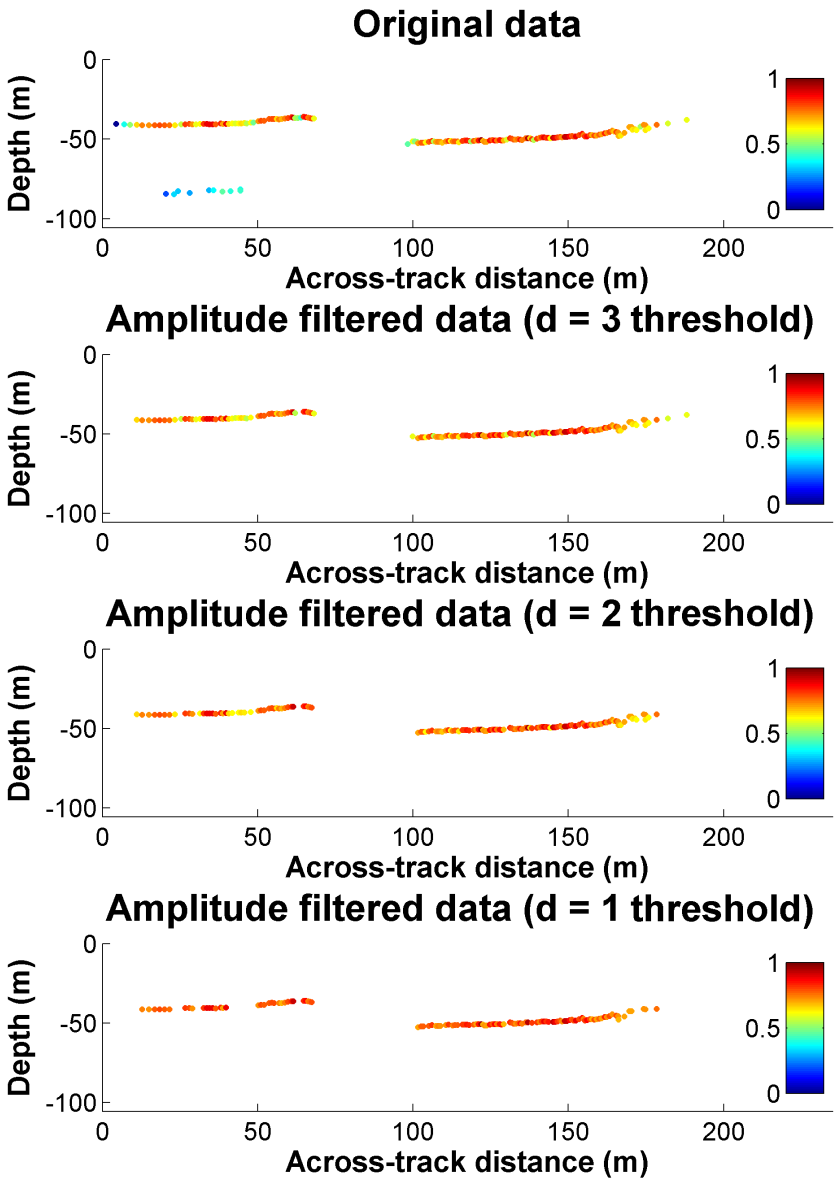

Figure 2.4: Amplitude filtering of bathymetry data with multi-path using multipliers $d_a = 3, 2, 1$ (*top-to-bottom*). The color legend on the right side of the each plot shows the amplitude representation of the data points, with 0 being the weakest amplitude strength and 1 being the strongest.

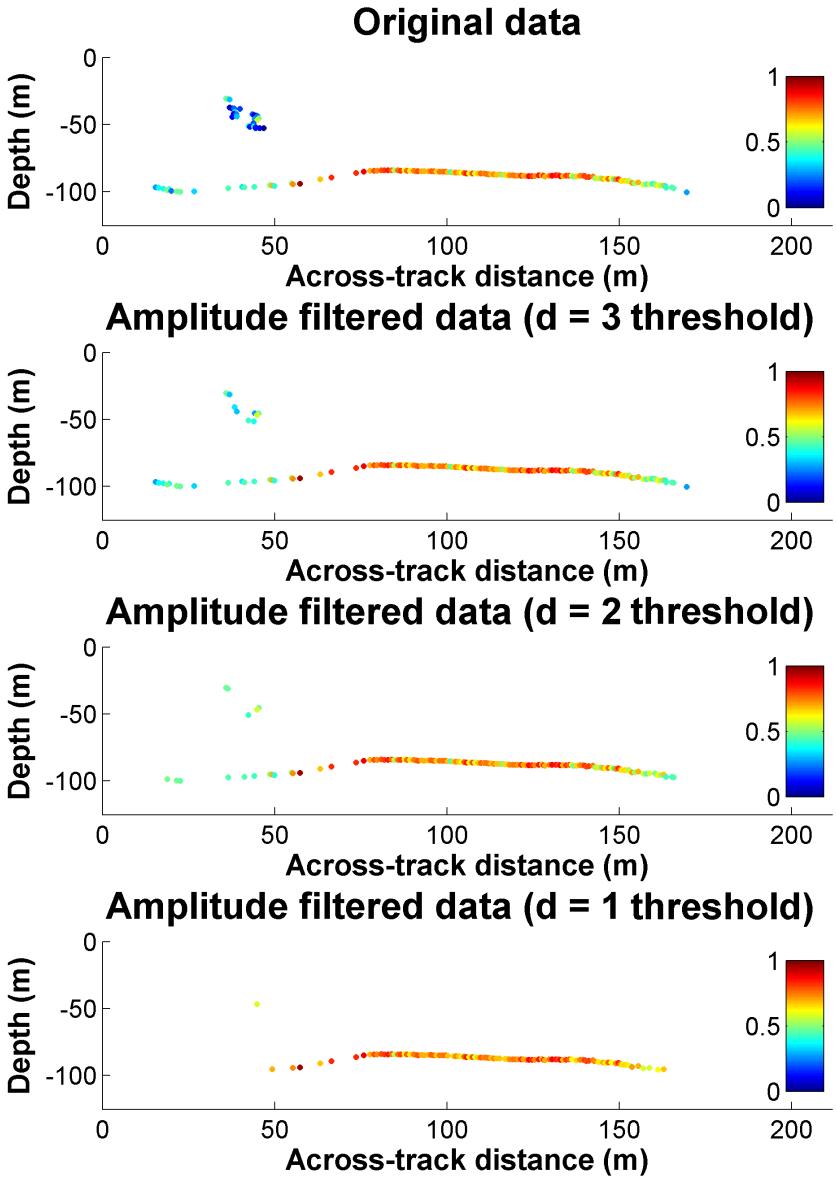

Figure 2.5: Amplitude filtering of ping's original data with water-column targets using multipliers $d_a = 3, 2, 1$ (*top-to-bottom*). The color legend on the right side of the each plot shows the amplitude representation of the data points, with 0 being the weakest amplitude strength and 1 being the strongest.

displayed in cartesian space (across-track and depth distances) for clarity. However, angle ray-casting processes the signal returns based on their AOA values, so the intermediate steps of method is presented entirely in electric-angle and range (polar coordinates) space instead. The final results of this method are presented in cartesian space once again. Figure 2.6 provides a visual comparison of the same bathymetric profile for the two different spaces. In both instances, the water-column object is highlighted to indicate these data will be removed through the procedure described below. Figure 2.7 demonstrates the concept behind this method, where a ray is casted horizontally along an angle value corresponding to a water-column object signal return. The ray continues after the water-column object and passes extremely close to a sea-floor return signal. Detecting these instances of near misses or intersections of multiple signal returns along a ray is key to removing outlier data points, and the main motivation for this method.

To simplify the detection of a ray passing near a sea-floor signal return, the data set is rasterized onto a uniform grid of polar coordinates. Detecting ray intersections with bathymetry data becomes just a simple comparison between the discrete-angles from rasterization. The discrete signal return position, $\{\theta_{i,d}, r_{i,d}\}$ is computed from the original position, $\{\theta_i, r_i\}$, with a grid spacing of $\Delta\theta$ and Δr using the following equations:

$$\theta_{i,d} = \Delta\theta \lfloor \tfrac{\theta_i}{\Delta\theta} + \tfrac{1}{2} \rfloor,$$
$$r_{i,d} = \Delta r \lfloor \tfrac{r_i}{\Delta r} + \tfrac{1}{2} \rfloor,$$

for every i in the bathymetry data set.

Figure 2.8 demonstrates rasterizing the bathymetry data onto a grid, where the blue dots in the figure correspond to the signal return data points from pre-processing and red dots correspond to the rasterized positions. The angle ray-casting technique removes all the data along the rows of the grid that contain multiple, filled grid cells. As a consequence of removing data in this way, the spacing of the grid will greatly effect the desired result of the filtering method. A large grid spacing can demolish a significant amount of sea-floor information by rasterizing a large region of sea-floor signal returns onto just one data point for the grid cell. Alternatively, a small grid spacing might not be large enough to rasterize outlier and sea-floor signal returns onto the same row. If the outliers aren't positioned along the same row, this method might not properly detect and remove the outlier data. A grid spacing of $3°\text{x}3m$ is used for this technique because it provides a good balance between sea-floor preservation and outlier elimination.

Rather than build the whole grid explicitly, an implicit-grid data structure is used to improve efficiency. In this sense, only grid cells with rasterized data points are actually created. This simpler data structure forms a C-length array of signal returns (where C is the total number of rasterized grid cells). To allow the system to easily traverse along rows

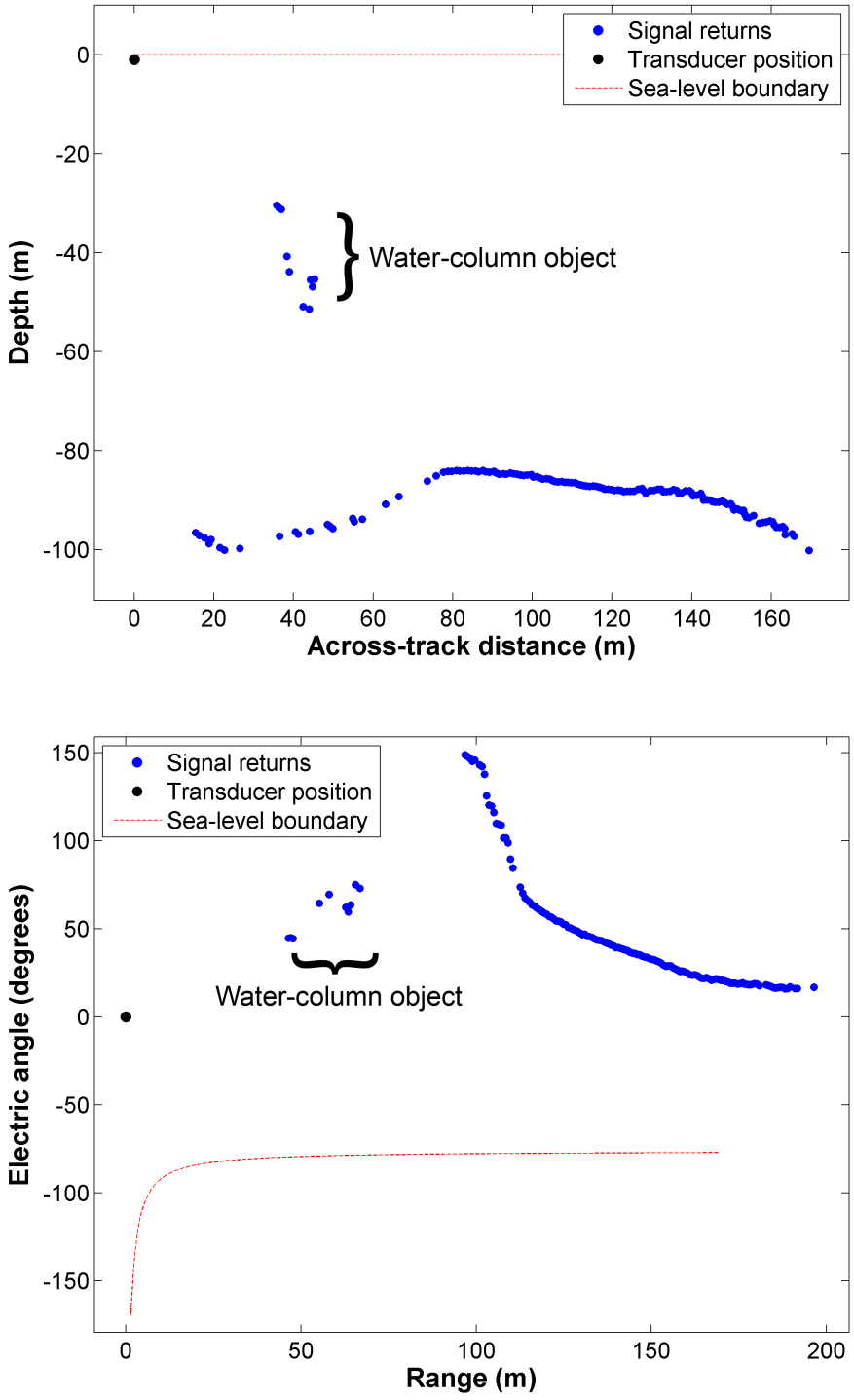

Figure 2.6: Comparing outlier data in cartesian and polar coordinates. The indicated region corresponds to the water-column target, with the remaining data points corresponding to the sea-floor.

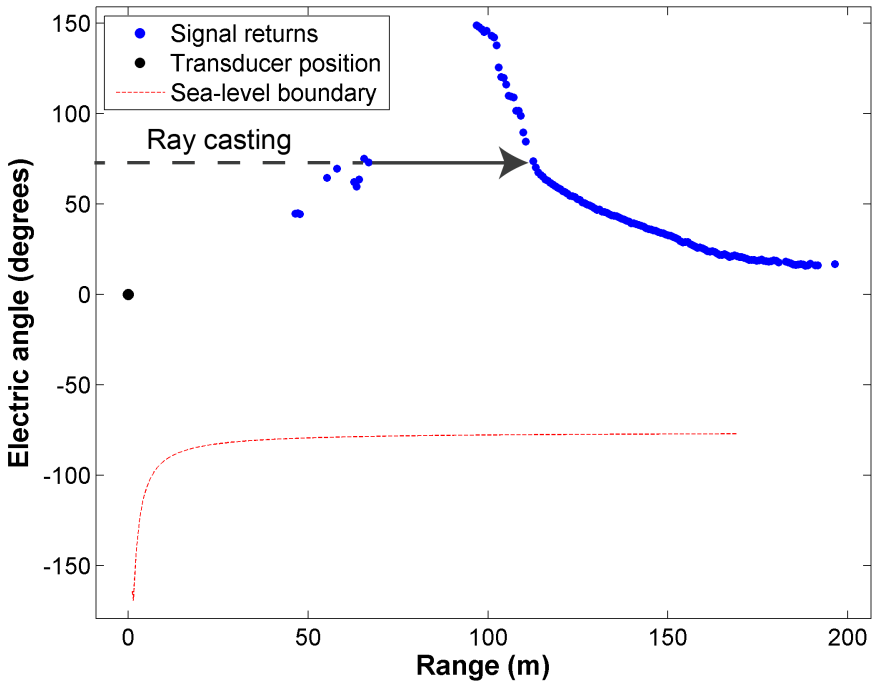

Figure 2.7: Example of ray casting along a particular angle that intersects with a water-column data point and a sea-floor data point. Because there are multiple intersections along this ray, and they are well separated, they will be removed.

(angles) of the grid, the grid cells must be stored in-order by angle and then range. Although a discrete and uniformly spaced dataset with multiple dimensions (sorted by angle, then range) is an ideal case for a linear-time-complex radix sort method, there are generally less than one hundred grid cells to sort so the benefit of a radix sort would not be realized. For this method, a simple $O(n\log(n))$ sort has no noticeable impact on efficiency. After sorting, the implicit method allows the system to operate in linear $O(C)$ complexity.

With a sorted array, the procedure of removing outlier data begins by "casting a ray" along grid rows containing grid cells. Because of the implicit data structure, this simply means grouping together the grid cells that correspond to the same row (angle). Along a particular row, the grid cells with the smallest and largest range values are compared. If the difference of range values between the two cells are greater than a specified threshold, it will likely contain outlier data and all the grid cells are removed[1]. In the instance where multiple grid cells along the row are not separated beyond the range threshold, the grid cell with the smallest range is preserved and the rest of the grid cells are removed.

Figure 2.9 demonstrates the process of removing grid cells corresponding to outlier data

[1]The grid cell instances aren't actually removed from the data structure. It's more efficient to fill a new data structure with just the preserved grid cells.

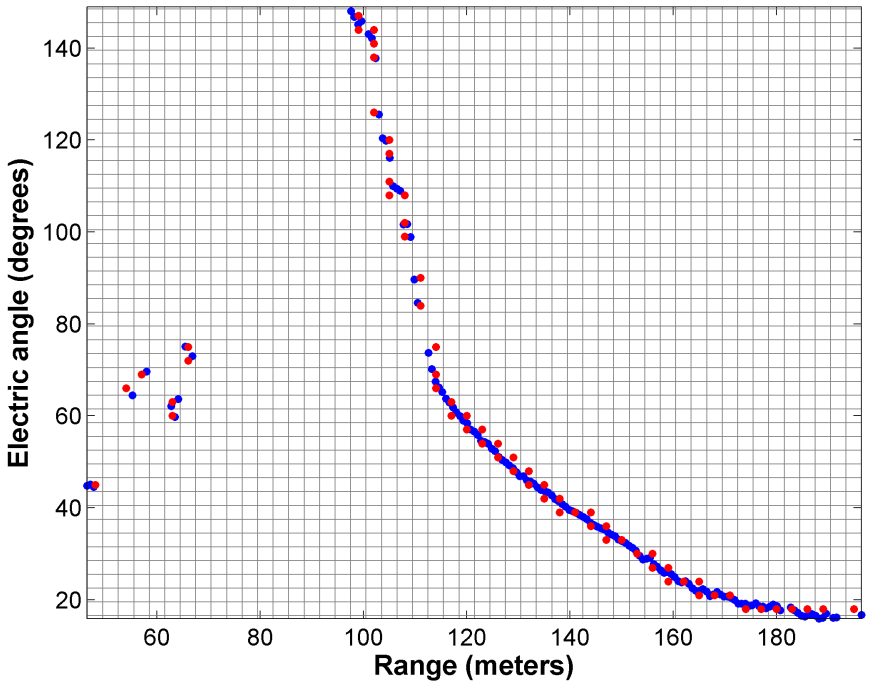

Figure 2.8: Rasterizing grid example. The bathymetry data from pre-processing (*blue dots*) are rasterized onto the 3°x3m grid (*red dots*).

by detecting distant grid cells along the same discrete angle. The top example in the figure shows two instances of grid cells that are separated beyond the threshold of 9m. One example presents the separation between an outlier and a sea-floor grid cell, where both are consequently removed in the resultant example. The second example (bottom right corner) highlights a major drawback of this method; the sea-floor data itself are separated beyond the threshold, causing the entire row of sea-floor data to be removed. In the resulting example (bottom) in figure 2.9, the majority of the outlier data has been removed. The one remaining grid cell corresponding to the water-column object persists because it did not happen to have a sea-floor grid cell rasterized onto the its row. Additionally, the results show there are no remaining sea-floor grid cells in the bottom two rows because the separation of the grid cells was beyond the threshold.

At this point, the remaining implicit grid data structure will consist of a sorted (by angle) set of data points with either zero or one grid cell per discrete-angle row. Because the grid cells are ordered, a large difference in range values between consecutive grid cells in the data structure indicate the likelihood of an outlier grid cell. Figure 2.10 demonstrates the process (*top*) and result (*bottom*) for detecting and removing a residual outlier grid cell. In this example, there is a large jump in range values between grid cells a and b and then

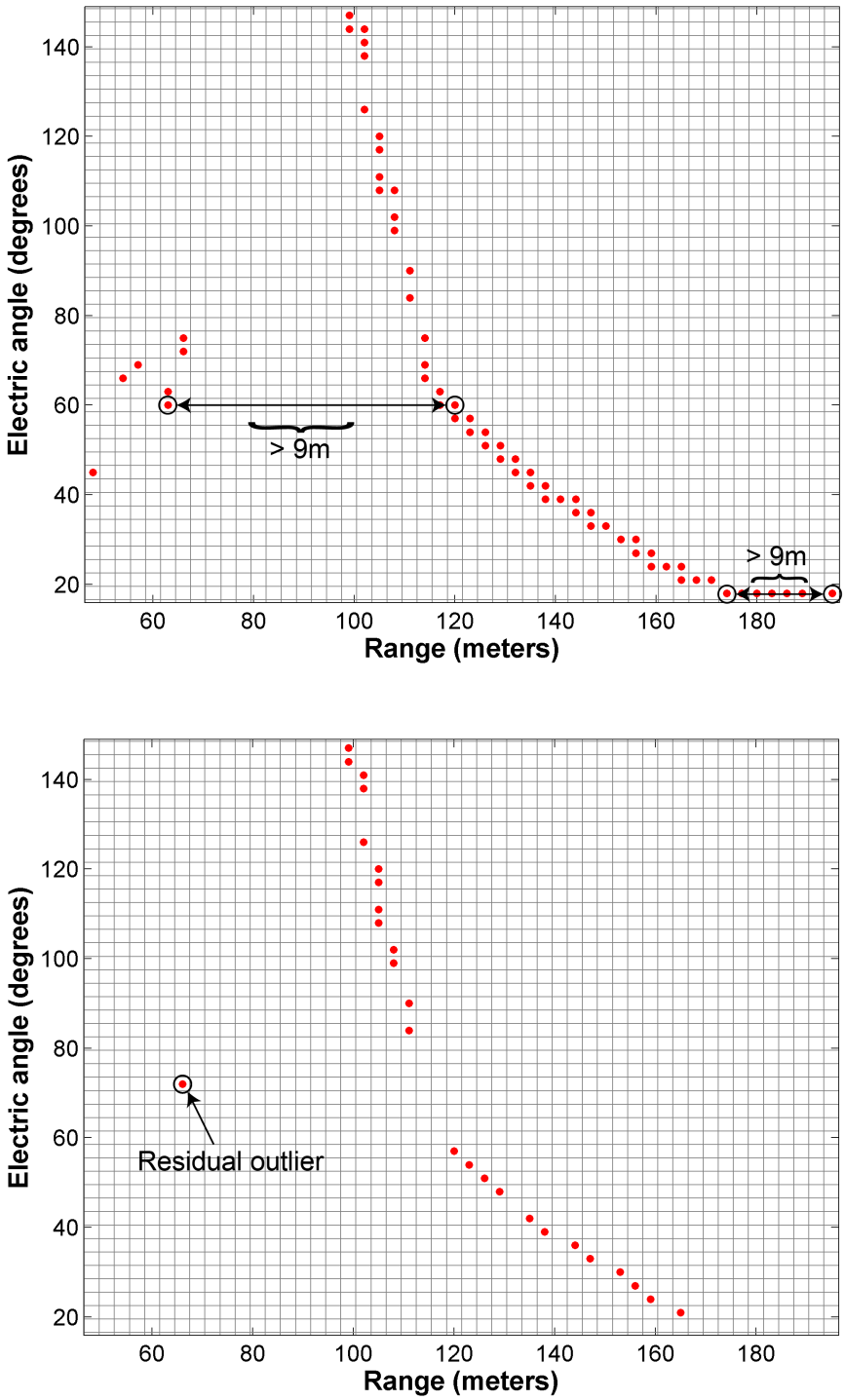

Figure 2.9: Example of the ray casting process (*top*) and the resulting grid (*bottom*). Two instances of grid cell removal are presented - the outlier to sea-floor separation and internal sea-floor separation. The results show the majority of outlier data are removed, along with its corresponding sea-floor data. Additionally, the sea-floor data in the bottom right are removed even with no outliers.

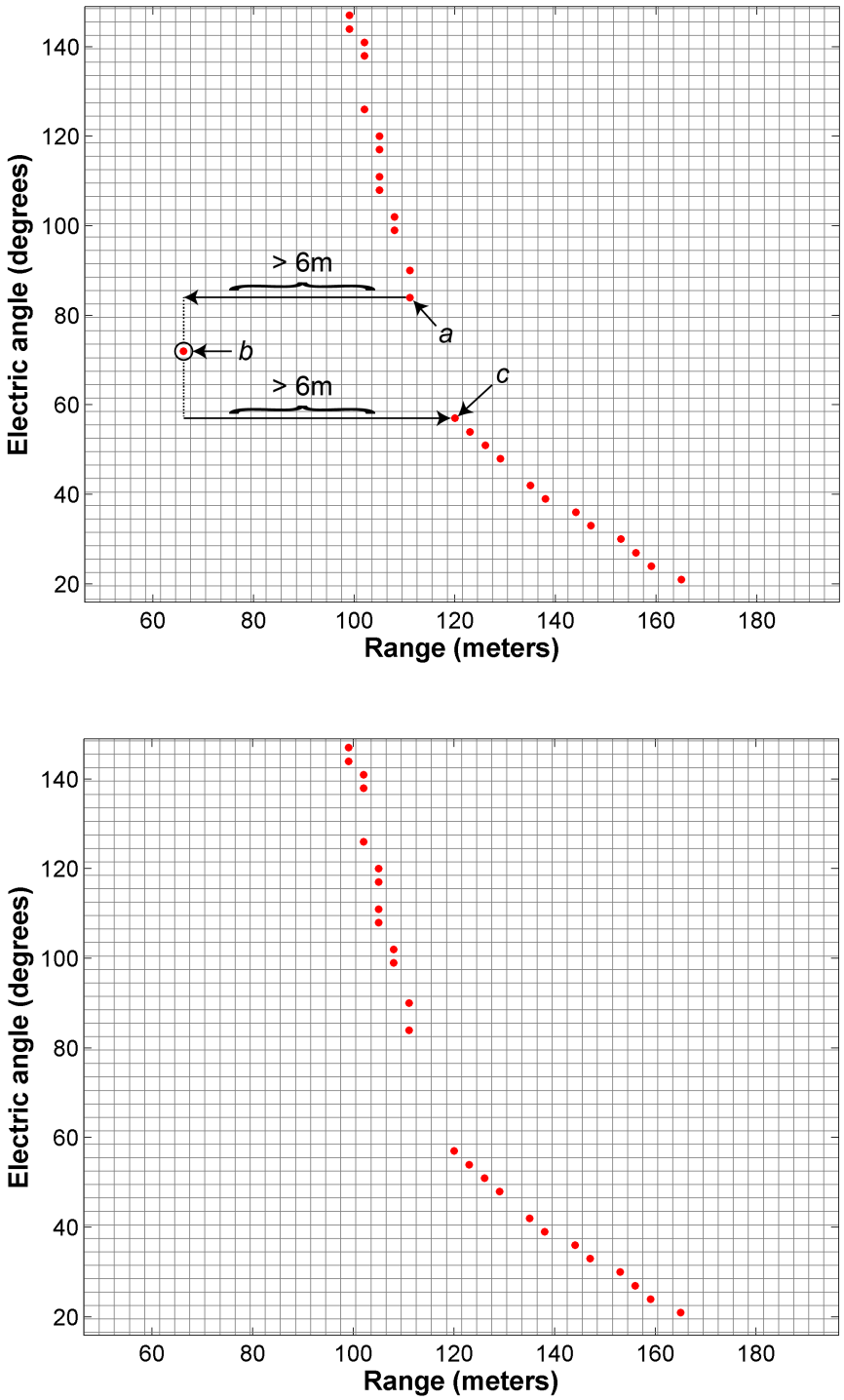

Figure 2.10: Example of the residual outlier detection process (*top*) and the resulting grid (*bottom*). The outlier grid cell jumps, in both directions, a distance that is greater than the allowable threshold. The outlier is removed and the general profile of the sea-floor is intact.

an immediate large jump between b and c in the opposite direction. This feature indicates that b is likely an outlier data point and it is removed from the system. It's also common for residual outliers to exist in pairs, so a second test is performed by extending the operation one index further: if there is a large jump in range values between grid cells a and b, followed by a small range difference in b and c, and then an immediate large jump between c and d in the opposite direction, then b and c are likely outliers and they are removed from the system. The threshold for qualifying a *large* jump in range values is set to 6m. The result in figure 2.10 demonstrates that this method can effectively remove outlier data while still preserving the general profile of the sea-floor.

Figure 2.11 compares the results of the angle ray-casting method from figure 2.10 (as red dots converted to cartesian space) with the original bathymetry data from figure 2.6 (as blue dots). The figure indicates that the signal returns corresponding to the water-column object have been successfully removed. However, two regions of the sea-floor have lost signal return data as result of this method. The first region, centered around 80m in across-track distance, is a result of removing grid cells along rows with outlier data. The second region is a result of a large region of signal returns being positioned along very similar angles. This creates the false impression of outliers because of the separation and all the data is removed. Figure 2.9 already demonstrated this problem, where the lower right corner of the ray casting plot indicates the sea-floor data is separated beyond the specified threshold and therefore removed.

Figure 2.12 presents two additional examples of angle ray-casting results. In both cases the multi-path outlier data are removed from the system with most of the sea-floor data still intact. Contrarily, figure 2.13 presents two examples where a multi-path data point isn't removed (*top*) and a large region of sea-floor data is removed (*bottom*). The multi-path data point isn't removed for two reasons: 1. because the signal returns corresponding to the sea-floor aren't positioned along the same discrete angle as the multi-path, ray casting doesn't remove it and 2. because the multi-path data point is in the last row of the angle-range grid, the test performed in figure 2.10 doesn't catch it. The sea-floor data loss in the bottom example of figure 2.13 is caused by a large separation of data along the same angle, and being incorrectly removed through the ray casting step. In this instance, a low SNR region is the cause of the sea-floor data removal.

This method by itself is not a sufficient solution for sea-floor profiling because it doesn't consistently remove outlier data or preserve the sea-floor data. However, because the entire angle ray-casting method is extremely fast to compute, averaging a 0.009ms completion time per ping, the remaining data points are used to initialize the cluster center positions for the fuzzy c-means clustering algorithm (for the curve tracing method presented in the following section). Normally, the cluster center positions are randomly initialized and then the algorithm converges the cluster centers onto the data set. Because the bathymetry

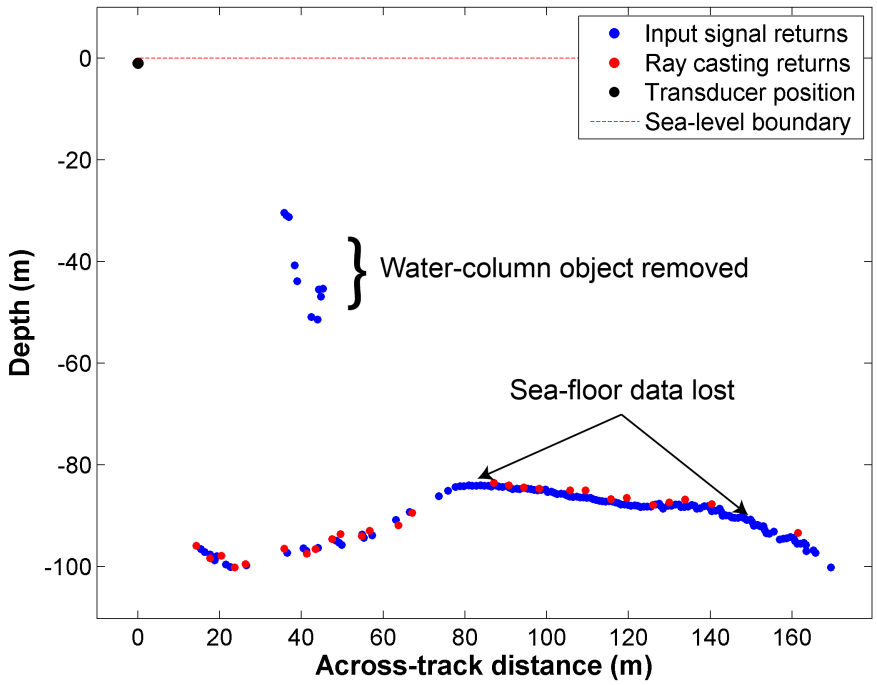

Figure 2.11: Ray cast final in cartesian space with resulting signal returns (*red dots*) superimposed over the original bathymetry from pre-processing (*blue dots*). The outlier data is removed, with the drawback of losing sea-floor data along the same angle. Also, sea-floor regions with shallow grazing angles are removed.

data from the pre-processing module are used as the data set for the curve tracing method, the normal method will cause some cluster center positions to converge onto outlier data. Initializing the cluster center positions from angle ray-casting will create cluster centers that are already well converged onto the sea-floor data. Although figure 2.13 demonstrates that some regions of the sea-floor will not be described by the angle ray-casting output, this problem will be largely solved by converging the cluster centers to the bathymetry data with fuzzy c-means clustering, which will spread the cluster centers out to fill these regions.

2.3. Curve Tracing

This section presents a modified version of the FCT algorithm by Yan [22]. Yan presents a method to trace a path through a point-cloud curve by applying a clustering algorithm to the data, and joining the cluster centers together to form a trace of the desired curve. After the initial trace, he then improves the quality of the trace by adding a penalty term to the clustering algorithm's cost function to reduce the curvature of the trace. This process convincingly traces the profile of the various curves in the data set presented in [22]. However, the data set consists of curves with no breaks, data points that are well distributed

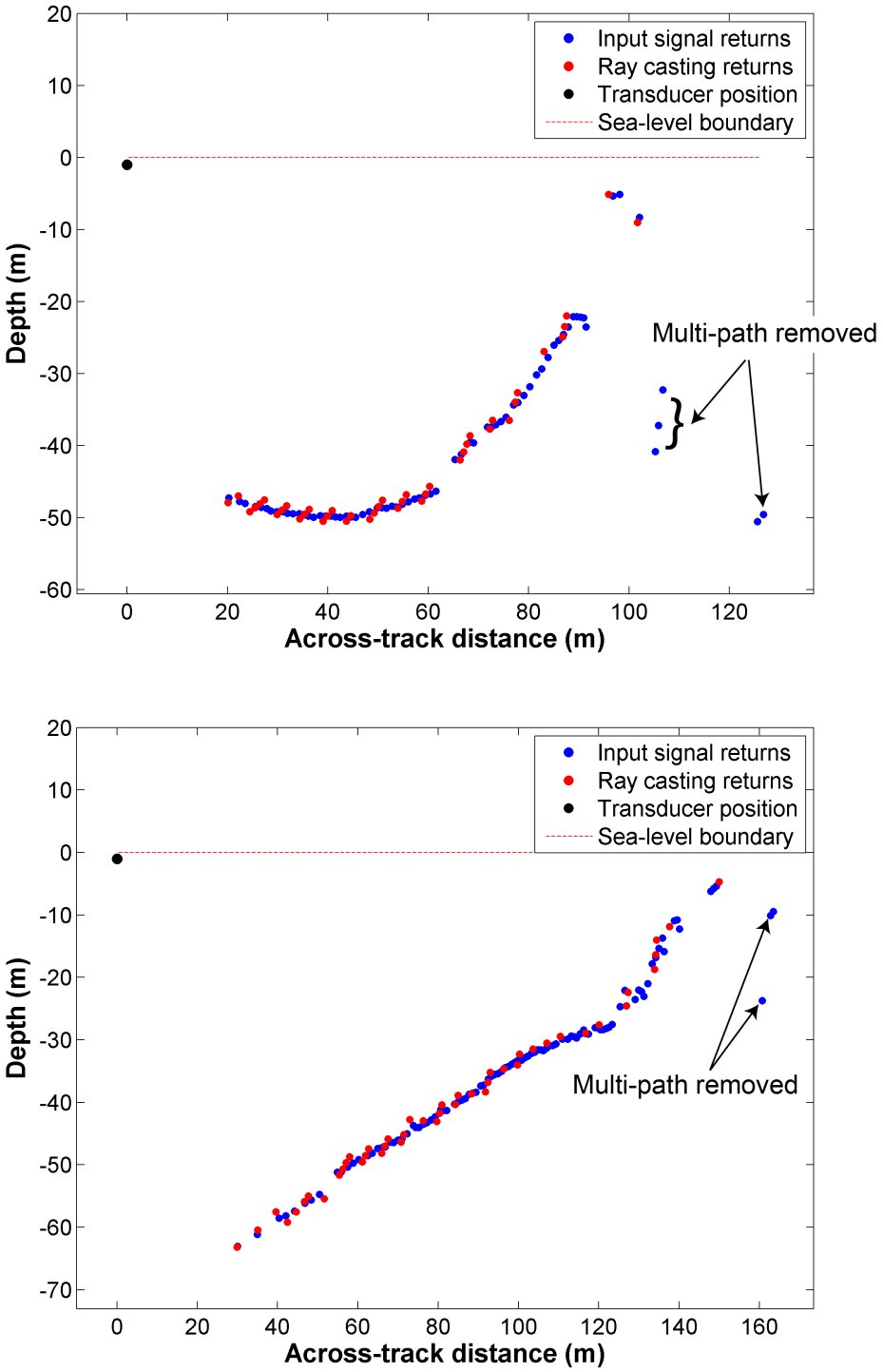

Figure 2.12: Successful angle ray-casting examples in cartesian space with resulting signal returns (*red dots*) superimposed over the original bathymetry from pre-processing (*blue dots*). In both cases, the multi-path is removed and most of the sea-floor regions are preserved.

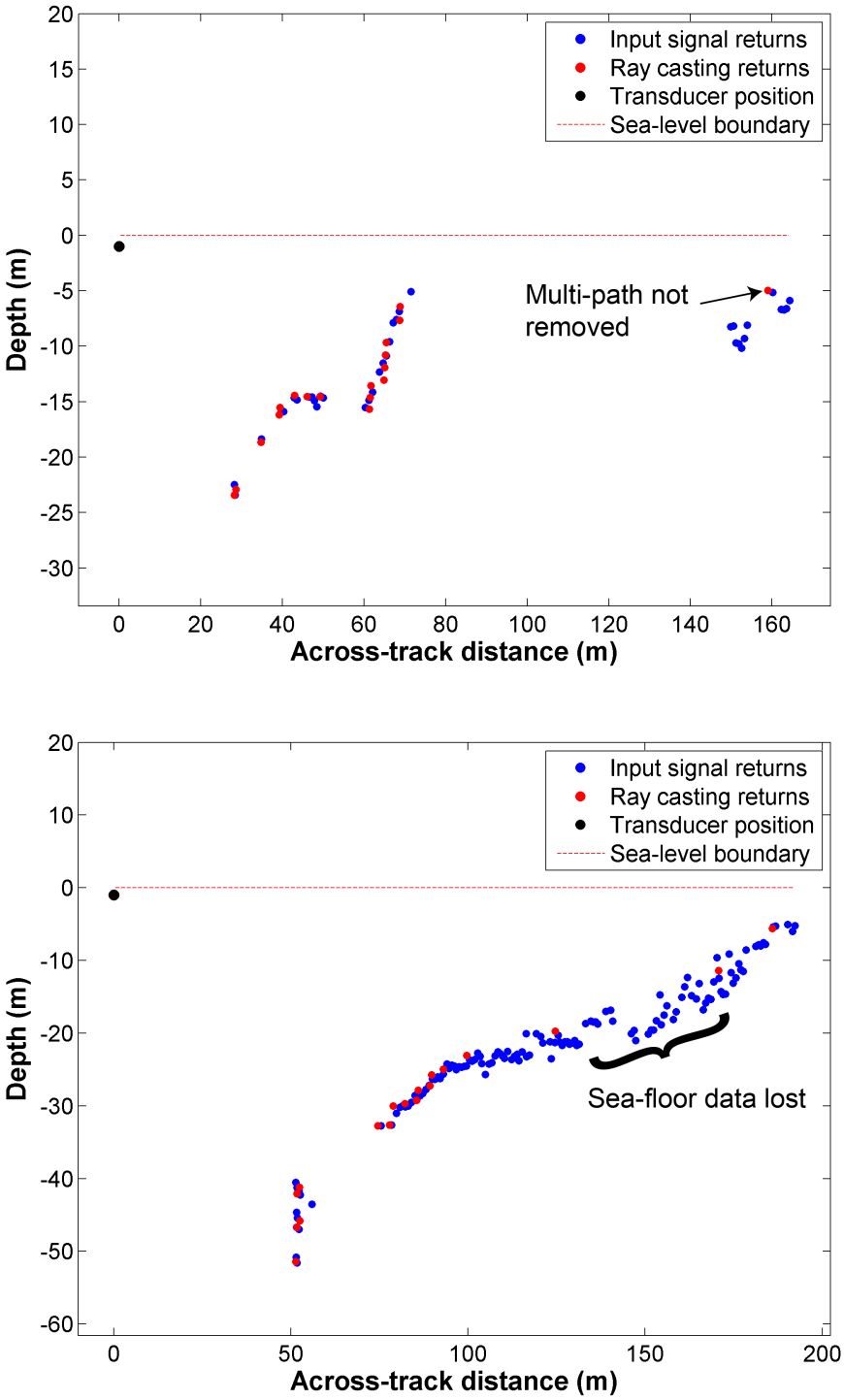

Figure 2.13: Problematic angle ray-casting examples in cartesian space with resulting signal returns (*red dots*) superimposed over the original bathymetry from pre-processing (*blue dots*). In these cases, not all multi-path is removed (*top*) and data for large regions of the sea-floor are lost (*bottom*).

along the curve, and a data set that contains no outliers. These three features only occur in optimal cases in the URL's Pam Rocks data set; most often the bathymetric profiles contain breaks in the sea-floor profile, an uneven distribution of bathymetry data along the sea-floor and many outliers from multi-path and water-column objects. The original FCT algorithm is modified in the following subsections to handle these additional complications. The algorithm presented in this thesis is modified from [22] in three major ways:

1. Adds a repulsion penalty term to the clustering cost function to encourage the cluster centers to be well-distributed even with a poorly distributed data set.

2. Joins curve traces across nadir-region breaks in the sea-floor profile (while avoiding outlier data) by projecting outwards from the end of a curve trace and matching it to the start of another curve trace.

3. Joins curve traces across shadow breaks in the sea-floor profile by projecting along the physical angle (with respect to the transducer) at the end of a curve trace and matching it to the start of another curve trace.

Before discussing the repulsion term it's important to explain how the standard fuzzy c-means algorithm operates.

2.3.1. Fuzzy C-Means Clustering

The unsupervised learning technique of fuzzy c-means (FCM) clustering is the main engine in the curve tracing solution. This method is born from the hard clustering technique of k-means clustering; a method that classifies data points in a data set as belonging to a single cluster center in a cluster set. After classifying the data points, the method moves the cluster center positions by minimizing the overall distance from each data point to the cluster center it is classified as. This process iterates between classifying data points and moving cluster centers until the system's cost function converges within an error tolerance.

The relationship between clusters and data points is described through a "membership" matrix, $\mathbf{U}$, which relates a data point, $\mathbf{x_i}$ (in data set $\mathbf{X}$), to a cluster center, $\mathbf{c_j}$ (in the cluster set $\mathbf{C}$). The membership matrix is an NxC matrix where N is the number of data points and C is the number of cluster centers. For k-means, $\mathbf{U}$ is a binary matrix, meaning a data point either completely belongs to a cluster or it completely does not. For example, when observing data point i and cluster center j, the corresponding membership matrix entry, $u_{i,j}$, will be either 0 or 1 (for no membership or complete membership, respectively). Because a data point can only belong to one cluster center, the entire i-th row of $\mathbf{U}$ can only contain a single 1. The main difference between k-means and FCM is that the data point membership is non-binary (or *fuzzy*) for FCM, meaning for any data point i and cluster center j, the membership value $u_{i,j}$ will be on the interval $[0,1]$. However, because the total

ownership of a data point i must still sum to 1, the i-th row of $\mathbf{U}$ must still sum to 1 and FCM is constrained by [45, 46]:

$$\sum_{j=1}^{C} \mu_{ij} = 1. \tag{2.3}$$

Like optimization techniques, FCM's goal is to minimize its cost function. In this case, FCM is a process of minimizing an error measure based on the squared distance between the cluster centers and the data points. This error method is weighted by the membership of the data point to the cluster center. The minimized error function is represented in the following equation:

$$\min_{\mathbf{U},\mathbf{C}} \{J_{FCM}(\mathbf{U},\mathbf{C};\mathbf{X}) = \sum_{i=1}^{N}\sum_{j=1}^{C} \mu_{ij}^{m} \|\mathbf{x_i} - \mathbf{c_j}\|^2\}, \tag{2.4}$$

where $\mathbf{x_i}$ and $\mathbf{c_j}$ are d-dimensional data point and cluster center locations (respectively). The m parameter in (2.4) is commonly referred to as the fuzzifier parameter, which directly impacts how soft the clustering method is [46, 47][2]. The standard value for FCM methods is $m = 2$ and that's what will be used for the remainder of this document.

Because there's no closed form solution to directly minimize J_{FCM}, it has to be minimized in parts by taking the partial derivatives of J_{FCM}. This forms two separate groups of partial derivatives: all the entries of the membership matrix and all the cluster centers . The minimum of J_{FCM} is approached by iteratively solving for the minima of each group ($\frac{\partial J_{FCM}(U,C;X)}{\partial \mu_{ij}} = 0, \frac{\partial J_{FCM}(U,C;X)}{\partial \mathbf{c_j}} = 0$).

Using the constraint in (2.3) as a lagrange multiplier, the partial derivative of J_{FCM} with respect to μ_{ij} provides a solution for the expectation phase:

$$\mu_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|\mathbf{x_i} - \mathbf{c_j}\|}{\|\mathbf{x_i} - \mathbf{c_k}\|} \right)^{\frac{2}{m-1}}}. \tag{2.5}$$

The updated membership values from (2.5) are then used to update the cluster center locations by taking the derivative of J_{FCM} with respect to $\mathbf{c_j}$ and solving for the maximization phase:

[2]For example, when m approaches 1, the exponent term in (2.5) will approach infinity. In this situation, if the cluster center j is the closest cluster center to data point i, the denominator with approach 1 and μ_{ij} will approach 1. Alternatively, if cluster center j is not the closest cluster center to data point i, the denominator will approach infinity and μ_{ij} will approach 0. This means when $m = 1$, this method becomes the *hard* clustering method of k-means.

$$\mathbf{c_j} = \frac{\sum_{i=1}^{N} \mu_{ij}^{m} \mathbf{x_i}}{\sum_{i=1}^{N} \mu_{ij}^{m}}. \tag{2.6}$$

The system is iterated by updating the membership matrix with (2.5), then using those values to update the cluster center positions by (2.6). However, before the initial update of the membership matrix, the initial positions for the cluster centers need to specified. Normally, the cluster center positions are randomly initialized within the domain of the data set $\mathbf{X}$. Instead, in this thesis, the set of cluster centers, $\mathbf{C}$, are initialized by the output data from the angle ray-casting method. The data set, $\mathbf{X}$, is initialized by the bathymetry data from the pre-processing module, so this adaptation to cluster center initialization provides a nearly-converged set of initial positions that approximate the sea-floor reasonably well. Using this initialization also reduces the likelihood that a cluster center will converge onto outlier data because they are already well positioned to be influenced by the sea-floor bathymetry.

After each iteration, the system updates the J_{FCM} value from (2.4) and compares it to the value from the previous iteration with the follow function:

$$\|J_{FCM}^{t} - J_{FCM}^{t-1}\| < \epsilon, \tag{2.7}$$

where t symbolizes the current iteration. Once the J_{FCM} difference between iterations is smaller than the error bound, ϵ, the clustering process is considered converged and the process is completed.

It is important to note that FCM generally does not guarantee convergence to the global minimum. For the typical use case of classification, local minima with FCM can introduce serious mismatches. However, this application is not a direct classification problem, rather an exploited use to trace curves through cluster centers. Local minima have not been observed to cause problems with curve tracing because the cluster centers consistently approach the general profile of the sea-floor. For this reason, a larger-than-typical ϵ is acceptable (and, for performance reasons, highly encouraged). Using $\epsilon = 10$, the cluster centers converge extremely quickly and the algorithm usually finishes in fewer than 10 iterations. However, a hard cap of 20 iterations is set to prevent any serious runaway iterations from impacting the performance.

Figure 2.14 presents two examples of the FCM algorithm results with cluster center positions initialized by angle ray-casting and the bathymetry data initialized by the pre-processing output. The two examples correspond to two of the output results from the angle ray-casting: top example from figure 2.12 and bottom example from figure 2.13. In the top

example of figure 2.14, many cluster centers converge along the horizontal region of the sea-floor, while the sloped sea-floor region contains only sparse cluster centers. Additionally, a single cluster center has converged onto outlier data. The bottom example shows that cluster centers have moved into the region of the sea-floor where the signal returns were removed by the angle ray-casting method (from figure 2.13). This situation is desirable because it indicates the drawback of losing sea-floor data from angle ray-casting is not a substantial hindrance. However, once again, a dense grouping of cluster centers is observed along the sea-floor profile. These unevenly distributed cluster centers are completely undesirable.

The number of cluster centers, C, present in the system can significantly impact the system's ability to trace curves. If C is too small, the system will not have enough cluster centers to accurately approximate local curvatures and features present in the profile. However, if there are too many clusters, the system will become so localized that it begins to over-fit noisy regions. Additionally, because the method is $O(NC)$ complex (N is the number of signal returns in the bathymetry), minimizing the number of cluster centers helps control efficiency. The number of clusters will also impact performance in proceeding subsections because their efficiency also depends on the number of cluster centers. Although the initial C parameter is determined by the output from angle ray-casting, the following subsection presents a solution for reducing C when necessary.

This section demonstrates the ability to use the traditional fuzzy c-means algorithm to fit cluster centers to the point-cloud sea-floor profile. However, at the moment the clusters centers are just points along the profile. The clusters still need to be joined together to form chains that approximate the curves of the sea-floor profile. This is not entirely straight forward because outlier data, low SNR and breaks in the sea-floor can complicate this process. Also, before joining the cluster centers together, the issue of dense cluster center regions from figure 2.14 must be handled. To address this problem, the FCM algorithm is modified to enforce cluster center repulsion by introducing a penalty term to the FCM cost function, (2.4).

2.3.2. Fuzzy C-Means Clustering With Repulsion Constraints

Although, by design, the FCM cost function, (2.4), encourages cluster separation, the attraction towards data points generally overrides this (as shown in figure 2.14). To explicitly enforce spatial separation of cluster centers, a repulsion term presented by Timm et al. [48, 49] is added to the cost function from (2.4):

$$\min_{\mathbf{U},\mathbf{C}}\{J_{FCM}(\mathbf{U},\mathbf{C};\mathbf{X}) = \sum_{i=1}^{N}\sum_{j=1}^{C}\mu_{ij}^{m}\|\mathbf{x_i}-\mathbf{c_j}\|^2 + \alpha\sum_{j=1}^{C}\sum_{k=1,k\neq j}^{C}\sigma(\|\mathbf{c_j}-\mathbf{c_k}\|^2)\}, \tag{2.8}$$

where α is the coefficient to control the strength of the repulsion.

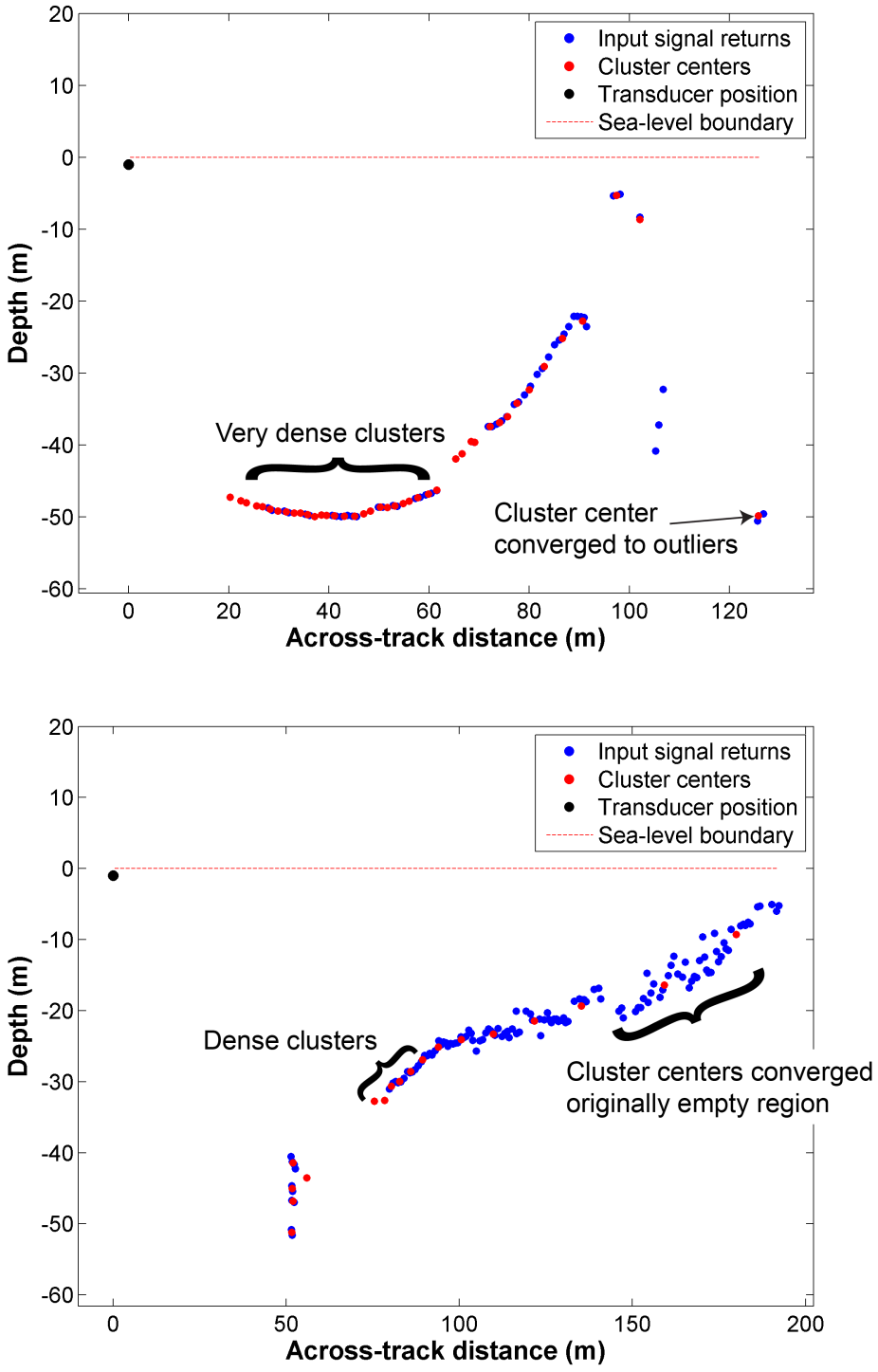

Figure 2.14: Fuzzy c-means clustering results. The cluster centers (*red dots*) converge onto the signal returns (*blue dots*). However, both examples display dense cluster center regions that complicate the curve tracing method and increase computation time.

This new cost function now also minimizes a kernel function $\sigma(x)$ based on cluster separation. Because the repulsion term should enforce the cluster center separation only at very local distances (and provide no influence for well separated cluster centers), a chosen kernel function should minimize the inverse of a distance function. The Gaussian kernel, $\sigma(r) = e^{(-r^2)}$, is ideal choice because it is stable when cluster centers are very close together.[3]

$$\min_{\mathbf{U},\mathbf{C}}\{J_{FCM}(\mathbf{U},\mathbf{C};\mathbf{X}) = \sum_{i=1}^{N}\sum_{j=1}^{C}\mu_{ij}^{m}\|\mathbf{x_i}-\mathbf{c_j}\|^2 + \alpha\sum_{j=1}^{C}\sum_{k=1,k\neq j}^{C} e^{(-\|\mathbf{c_j}-\mathbf{c_k}\|^2)},$$

Because there is no $\mathbf{x_i}$ term in the added penalty, the expectation step is not altered by this constraint and remains the same as (2.5). However, the partial derivatives for the maximization step differ from (2.6) because of the cluster centers present in the constraint. The new maximization step is:

$$\mathbf{c_j} = \frac{\sum_{i=1}^{N}\mu_{ij}^{m}\mathbf{x_i} - \alpha\sum_{k=1,k\neq j}^{C}\mathbf{c_k}e^{(-\|\mathbf{c_j}-\mathbf{c_k}\|^2)}}{\sum_{i=1}^{N}\mu_{ij}^{m} - \alpha\sum_{k=1,k\neq j}^{C}e^{(-\|\mathbf{c_j}-\mathbf{c_k}\|^2)}}. \tag{2.9}$$

However, Timm et al. [49] later propose a variation to penalty term coefficient α by using an adaptive weight for each cluster center. This weight is based on the how strongly a cluster center j influences a data point i, for all i in the data set:

$$\alpha_j = \alpha\sum_{i=1}^{N}\mu_{ij}^{m}. \tag{2.10}$$

Using (2.10), maximization step becomes:

$$\mathbf{c_j} = \frac{\sum_{i=1}^{N}\mu_{ij}^{m}\mathbf{x_i} - \alpha_j\sum_{k=1,k\neq j}^{C}\mathbf{c_k}e^{(-\|\mathbf{c_j}-\mathbf{c_k}\|^2)}}{\sum_{i=1}^{N}\mu_{ij}^{m} - \alpha_j\sum_{k=1,k\neq j}^{C}e^{(-\|\mathbf{c_j}-\mathbf{c_k}\|^2)}}. \tag{2.11}$$

The adaptive weight provides greater repulsion response in regions of more dense data points because there will be more points with high membership to a cluster center j values to scale α_j higher. The feature is very desirable for both the URL's test cases, because dense data point regions are the typical locations of poorly distributed cluster centers. However,

[3]Please note that the exp() function is not actually used in the C++ code, rather $e^{(-r^2)}$ is approximated by $\frac{1}{r^2+1}$. The exp() function is extremely time consuming to compute, and it quickly becomes a bottleneck in the system. The approximation is sufficient because it is still stable for small r values and attenuates for large r values in the same manner as the Gaussian kernel. However, the Gaussian kernel notation is used throughout this subsection for clarity.

Timm et al. do not address the instability of (2.11) when the denominator approaches zero. These instances of instability were observed to occur when a cluster center has little influence on any of the data points and it is sufficiently far away from any other cluster (in other words, both terms in the denominator each approach zero). The authors' test cases consist of large and distributed data sets and a small number of clusters. For such test cases, the cluster centers always have a large influence on a set of data points so instabilities were never encountered. The test cases from the URL's Pam Rocks data set are quite the opposite and, as the following equations demonstrate, the instabilities in this approach cannot be suppressed with the α parameter.

By combining (2.10) and (2.11), the denominator approaches zero in the following equation:

$$\sum_{i=1}^{N} \mu_{ij}^{m} (1 - \alpha \sum_{k=1, k \neq j}^{C} e^{(-\|\mathbf{c_j} - \mathbf{c_k}\|^2)}) = 0. \tag{2.12}$$

The instability encountered in (2.12) is observed in the following conditions:

$$\sum_{i=1}^{N} \mu_{ij}^{m} \approx 0, \tag{2.13}$$

$$\sum_{k=1, k \neq j}^{C} e^{(-\|\mathbf{c_j} - \mathbf{c_k}\|^2)} \approx \frac{1}{\alpha}. \tag{2.14}$$

As consequence of (2.10), the α term is de-coupled from the membership matrix. Regardless of the chosen α, the instability due to the membership matrix cannot be prevented in a system with a high cluster-to-data-point ratio. This scenario is unstable because data points will assign very small membership values to many cluster centers, so the summation in (2.13) of the all memberships assigned to a cluster center, j, from the data points will be extremely small. A high cluster-to-data-point ratio is observed in the dense cluster region in figure 2.14. In this case, the instabilities in the system will eject cluster centers away from the sea-floor profile during (2.11), leaving a reduced number of cluster centers in the domain to fit the profile.

Despite the drawbacks of the repulsion penalty, the unavoidable ejection of cluster centers from the sea-floor profile is actually very advantageous. Figure 2.14 demonstrates that there can be too many cluster centers initialized by angle ray-casting output. Since the instabilities launch cluster centers away from the sea-floor profile, the result can reduce the number of clusters remaining to trace the profile curves. However, at each iteration, the launched cluster centers must be detected and removed from the system. These launched cluster centers will be very distant from the bathymetry data, so they will receive approx-

imately zero membership from any data point (a consequence of 2.13). On the contrary, cluster centers positioned close to bathymetry data will receive substantial membership from the surrounding data points. This difference is used to identify and remove (indicated by $\emptyset$) likely *ejected* cluster centers by applying a threshold, k, to the sum total membership from all the data points to a cluster center, j:

$$\mathbf{c_j} = \begin{cases} \mathbf{c_j}, & \text{if } \sum_{i=1}^{N} \mu_{ij}^m \leq k \\ \emptyset, & \text{if } \sum_{i=1}^{N} \mu_{ij}^m > k \end{cases} \tag{2.15}$$

The threshold (2.15) is applied at the end of each iteration of (2.5 and (2.11). Experimental observations determined that the parameter settings $\alpha = 0.5$ and $k = 0.1$ successfully reduced and distributed the cluster centers in the algorithm. Figure 2.15 presents the result of applying the FCM algorithm with the repulsion penalty to figure 2.14. In both examples, there has been a reduction in cluster centers as well as a substantial improvement in the cluster center distribution. However, the top example indicates that there are more cluster centers converged onto outliers than in figure 2.14. This is an unfortunate consequence of the repulsion instability that has been observed to happen on rare occasions. The proceeding subsections present a step-by-step procedure to connect the cluster centers together in a way that avoids the impact that these additional outlier cluster centers have on the overall tracing of the sea-floor profile.

The addition of (2.11) and (2.15) provides a pseudo-adaptive solution for the number of cluster centers and enforces the spatial distribution of the remaining cluster centers. These improvements greatly increase both computational efficiency and curve tracing effectiveness.

2.3.3. Fuzzy Curve Tracing Algorithm

Once the FCM process has converged, the next step in the algorithm is to connect neighbouring cluster centers together to form a chain that traces the curves of a point cloud profile. The basic fuzzy curve tracing (FCT) algorithm joins local cluster centers, handles any loops formed in the joined chain, and, to avoid over-fitting the curve, constrains the smoothness of the chain [22]. To handle breaks in the sea-floor profile, the chains of cluster centers must be attached in a way that connects them along the sea-floor profiles but avoids connecting to cluster centers corresponding to outlier data (like the outlier cluster center in figure 2.15).

The initial step in the FCT algorithm is to connect the cluster centers together to form a chain. Yan presents a solution for connecting the cluster centers together based on how strongly one cluster center influences the data points nearest to another cluster center. For example, if cluster j and cluster k are being compared, Yan builds a set, S_j, of all the data

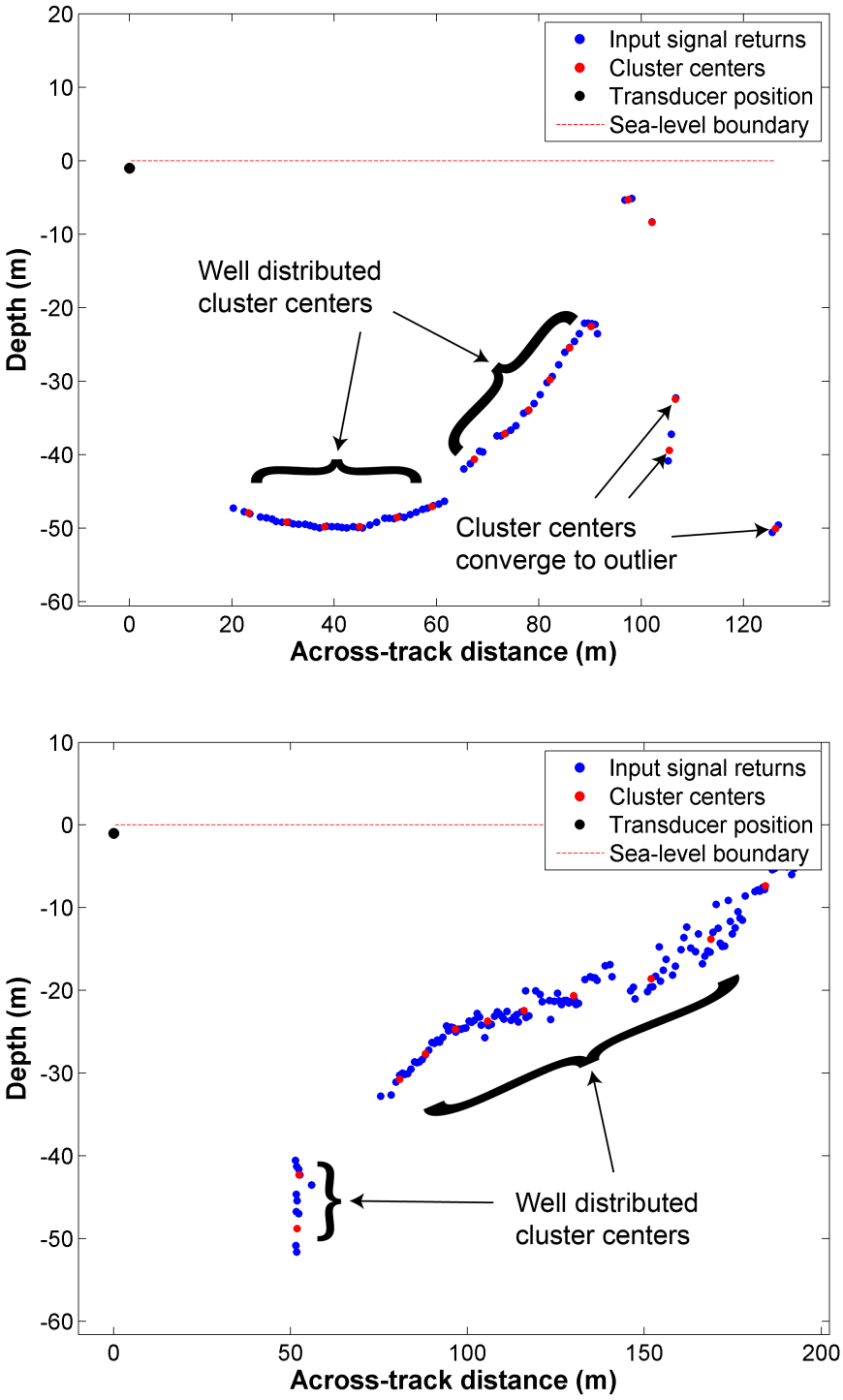

Figure 2.15: Cluster centers with repulsion penalty. The examples demonstrate an improved distribution along the profile compared to figure 2.14. Unfortunately, the instabilities in the repulsion sometimes converge cluster centers onto outlier data (*top*).

points with the largest membership value corresponding to cluster j. He then sums the membership values that each data point in set S_j assigns to cluster k. The summation is normalized, as to be consistent for different S_j lengths, by dividing it by the number of data points in S_j, represented by N_j. The final value, $f_{j,k}$ describes the influence that cluster k has on cluster j. After all the cluster centers are compared, the k cluster center that has the largest influence will be connected to cluster j. The process of comparing cluster center k's influence on cluster center j is described in the following equation:

$$f_{j,k} = \frac{100}{N_j} \sum_{i \in S_j} \mu_{ik}, \tag{2.16}$$

where $f_{j,k}$ is an entry in the influence matrix $\mathbf{F}$.

The influence matrix is used to identify the closest cluster centers and create a spatial relationship between clusters. While generally accurate, the approach is biased towards cluster centers in dense data point regions because the influence value tends to be greater there than for sparse data points regions, even with closer cluster centers. More importantly, Yan's proposed approach for joining cluster centers is deceptively costly. Each cluster center must be compared to every other cluster center, which initially appears to be an $O(C^2)$ complex method. However, because each comparison requires building its own S_j set, the entire membership matrix, $\mathbf{U}$, must be traversed per cluster center comparison. Building S_j is $O(NC)$ complex because $\mathbf{U}$ is an NxC matrix (N is the number of data points and C is the number of clusters) and the set is built separately for every cluster center comparison. In total, building the entire influence matrix $\mathbf{F}$ from (2.16) is $O(NC^3)$ complex and extremely time consuming as a result.

Instead of Yan's influence matrix, a more spatially consistent and efficient method is proposed. In this thesis, the cluster centers are linked to their two closest clusters by their squared Euclidean distance. The revised influence factor is described in the following equation:

$$\hat{f_{j,k}} = \|\mathbf{c_j} - \mathbf{c_k}\|^2,$$

where $\hat{f_{j,k}}$ is an entry in the new influence matrix $\hat{\mathbf{F}}$.

There are three motivations for this change:

- The Euclidean distance method is bounded by $O(C^2)$ complexity instead of $O(NC^3)$ from (2.16).
- In practice, N is generally much greater than C so the elimination of N in the asymptotic bound provides great savings.
- The new influence matrix, $\hat{\mathbf{F}}$, is symmetric, so only the upper triangular of $\hat{\mathbf{F}}$ must

be computed to link the cluster centers.

Since the cluster centers are linked to their closest two neighbours, the indices of the two smallest squared distances along row j of $\hat{\mathbf{F}}$ provide the corresponding connections for cluster center j. Table 2.1 presents an example of the two smallest squared distances (f_{j,k_1} and f_{j,k_2}) from cluster j to the rest of the cluster centers. Table 2.2 presents the corresponding cluster indices (k_1 and k_2) of the two closest cluster centers to cluster center j. Figure 2.16 presents visual representation of table 2.2 by tracing a red line from the cluster center corresponding to column j to the cluster centers corresponding to the indices k_1 and k_2 of the j-th column, for every j in the cluster set.

Table 2.1: Cluster distance matrix - sorted and condensed to the two smallest squared distances

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
f_{j,k_1}	30.7	30.7	50.2	541.9	25.6	33.7	32.5	47.6	32.5	50.2	48.4	46.7	25.6	48.4	46.7	55.6	56.5
f_{j,k_2}	350.4	342.6	370.7	692.4	33.9	33.9	33.7	109.4	47.6	519.7	55.9	55.9	116.9	109.4	55.6	56.5	223.8

Table 2.2: Cluster connection matrix - cluster indices corresponding to distances in table 2.1

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
k_1	2	1	10	10	13	7	9	9	7	3	14	15	5	11	12	15	16
k_2	13	13	13	3	6	5	6	14	8	13	12	11	6	8	16	17	15

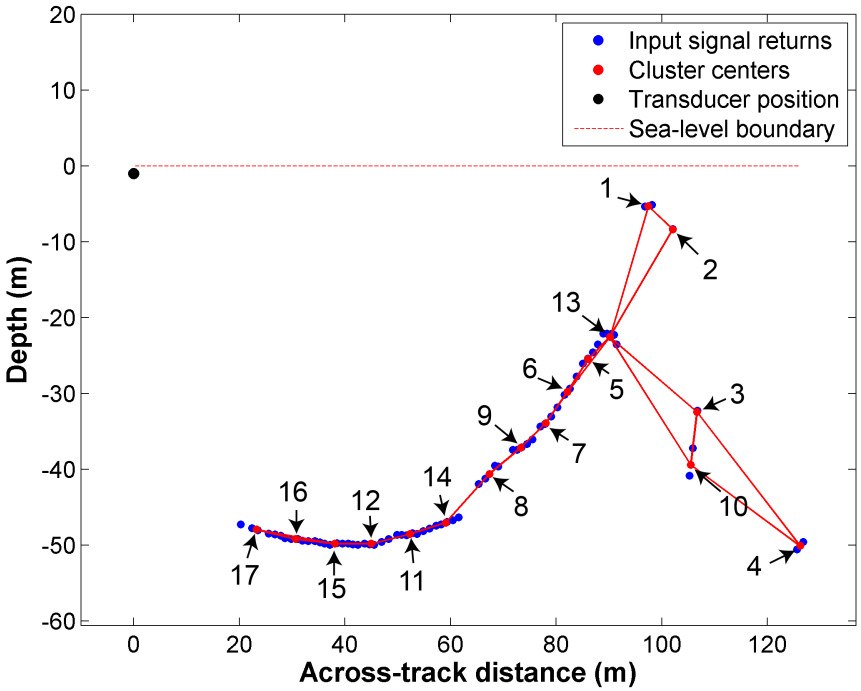

Figure 2.16: Linked cluster centers example with labelled indices. The connections between cluster centers are specified by the k_1 and k_2 indices for each column in table 2.2

The connections between cluster centers 17 and 13 in figure 2.16 trace the sea-floor profile quite accurately. However, once at 13, there are connections from one region near cluster centers 1 and 2 and another region near 3 and 10. In this example, 1 and 2 likely corresponding to the sea-floor on the other side of a nadir region from 13. Cluster centers 3 and 10 correspond to multi-path outlier data. Figure 2.16 does not indicate the direction of the connection; there's no indication if there is a red line originating from 13 outwards to 3 (meaning, in table 2.2, the column $j = 13$ contains either $k_1 = 3$ or $k_2 = 3$) or vice versa. However, table 2.2 does explicitly indicate which cluster center creates the connection. There are five cluster centers in table 2.2 that connection to cluster center 13, however there are only two cluster centers that 13 connects to. In fact, the only connection to 13 that is reciprocated by 13 is cluster center 5.

The connection reciprocation is more than just an interesting observation. This distinction is used to break connections between cluster centers in an effort to remove the complex web surrounding cluster center 13 in figure 2.16. Breaking these connections is necessary to manage complex webs because loops form at the end of every chain and sometimes in the middle of the chain. Figure 2.17 presents two profiles with loops at the end of the chain, a loop in the middle of the chain, and a complex web within the chain.

In the proceeding subsections of the FCT method, the cluster chains are processed in a way that requires the loops within the chains to be removed. The processes in these subsections constrain the curvature of the chains and join chains that profile sections of the sea-floor. If loops persist in the chain, it can be difficult to accurately complete those processes. The following subsection presents a process to break the loops and complex webs (see figure 2.17) based on connection reciprocation.

Breaking Chain Loops

The process of creating loop-free chains of cluster centers is similar to a minimum spanning tree algorithm. In essence, the loops are broken by maintaining the shortest connection and removing the longest. Removing end-loops in this fashion is simple and effective, but more complicated connections (see figure 2.17) prove to be more challenging.

Yan [22] presents a graph-rule system to break loops based on the angles of a chain loop and influence factors, from (2.16), between the cluster centers within the loop. However, Yan's data set only consists of continuous point clouds with very localized noise and limited outliers. As a result, his cluster centers typically connect together as a sequential chain with only a few 3-vertice loops. Contrarily, the URL's Pam Rocks data set contain discontinuous regions and outlier noise and result in multi-vertice webs (see figure 2.17).

Yan's method for breaking loops will provide an appropriate solution for trivial cluster chains. However, this method cannot resolve complicated connections and often results in non-manifold connections (a vertex with a rank greater than 2 and not in a loop). Instead,

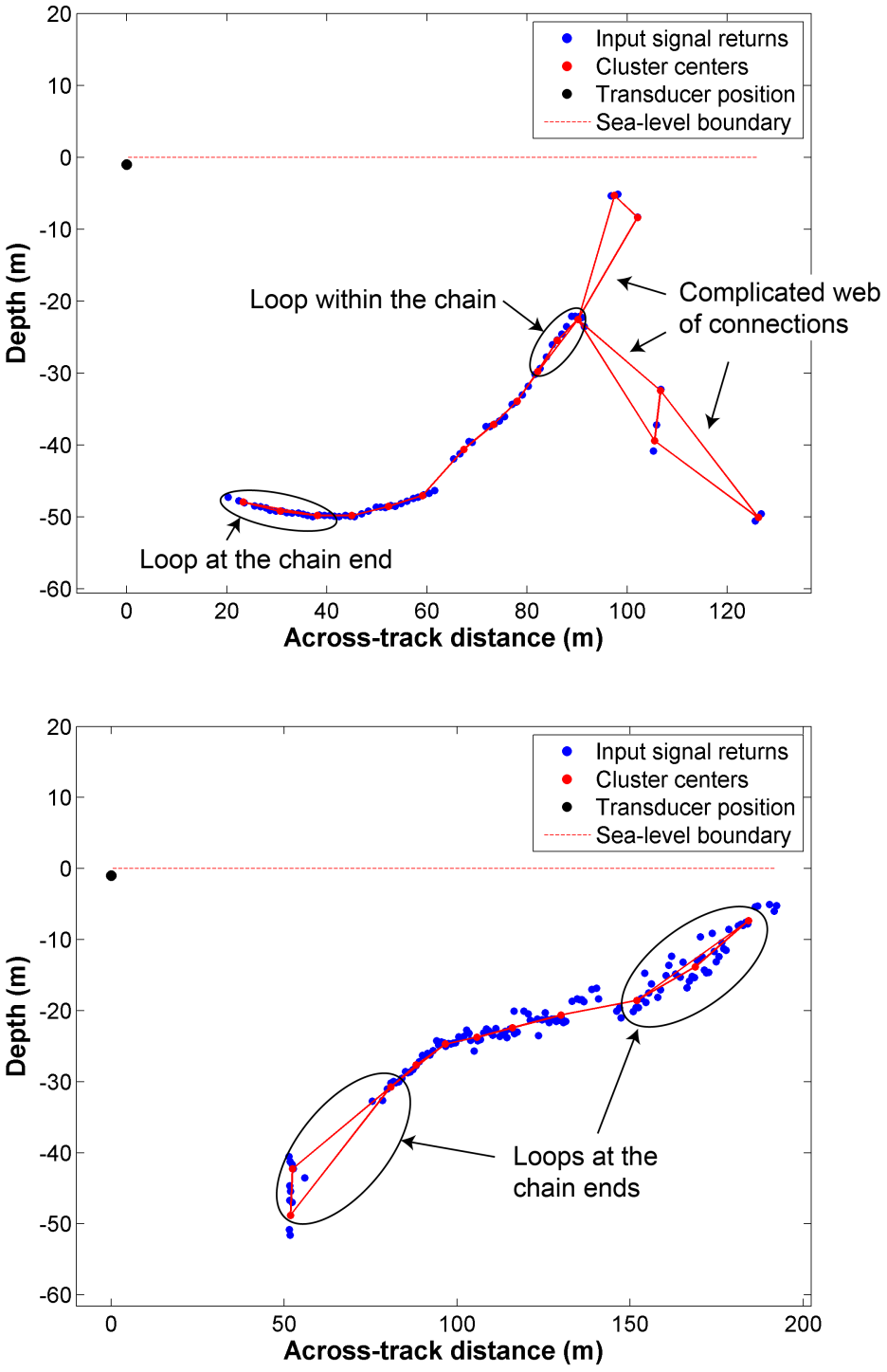

Figure 2.17: Linked cluster centers examples. The examples demonstrate the existence of complex webs (*top*), loops within the chain (*top*), and loops at the ends of the chains (*top and bottom*).

this thesis presents a more direct and simpler solution for breaking loops in cluster chains.

In this thesis, a link between two cluster centers is broken if the two cluster centers do not reciprocate a connection. Meaning that, if a cluster center i contains cluster center j as its k_1 or k_2 indices, then cluster center j must also have i as one of its k_1 or k_2 indices to maintain a link between the two. If no such reciprocating connection exists, then the index is removed; indicated by an $\emptyset$ in the location of the j index in the i-th column in table 2.3.

Figures 2.16 and 2.17 present an example of a complicated web centered around cluster center 13. Table 2.2 indicates that there are several cluster centers that connect to 13 but only cluster center 5 is reciprocated by 13. According to the method used in this thesis for breaking loops, all the connections from cluster centers that connect to 13 without reciprocation are be broken. In this example, the connections to cluster center 13 from cluster centers 1, 2, 3, and 10 are be broken. Additionally, the connection from 13 to 6 is broken because there is no reciprocation from 6 to 13. Figure 2.18 presents a visual explanation of this process. The connection direction from a cluster center j to its k_1 and k_2 cluster centers are indicated by a black arrow. In this figure, when a connection is not reciprocated, the connection is broken and indicated with a red "X".

Table 2.3 presents the result of the loop-breaking process by replacing non-reciprocating indices from table 2.2 with $\emptyset$ flags. Based on the example in figure 2.18, the $\emptyset$ flags are set in the columns corresponding to cluster centers 1, 2, 3, 10, and 13. Additionally, table 2.3 contains an $\emptyset$ flag in column 17 because, as figure 2.17 indicates, there is a loop at the end of the chain that is broken. Although cluster centers 1 and 2 are positioned on the far end of a nadir region, along the sea-floor, from cluster center 13, the connection across the gap is broken by the loop-breaking method. A strategy for re-connecting cluster chains across gaps in the sea-floor is presented in section 2.3.3.

Table 2.3: Cluster connection matrix after loop breaking

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
k_1	2	1	10	$\emptyset$	13	7	9	9	7	3	14	15	5	11	12	15	16
k_2	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	6	5	6	14	8	$\emptyset$	12	11	$\emptyset$	8	16	17	$\emptyset$

The loop-breaking method reliably removes complex webs and loops at the ends and in the middle of the chain. However, this approach can potentially create chains with closed loops. Detecting and handling these remaining loops requires traversing each chain in the system and testing if cluster center indices are repeated. Unfortunately this is not completely straight forward because table 2.3 does not provide direct access to a particular chain. Coincidentally, the remaining steps in the system also require the ability to directly access a specific chain, so an array structure for each cluster chain is built from table 2.3.

The cluster chain is an array where each element references a particular cluster center in the sequence of the chain. This provides direct access to the start and end links of every

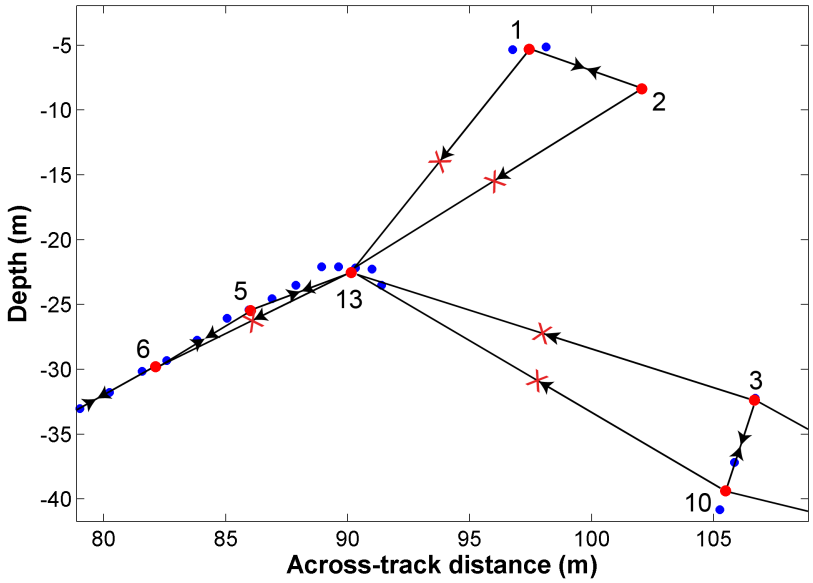

Figure 2.18: Breaking loops in a cluster chain. An enlarged version of the top example from figure 2.17. The connections formed from a cluster to its k_1 and k_2 indices (see table 2.2) are indicated with arrows, which point in the direction the connection is formed. A connection that is not reciprocated is indicated with a red "X" and it is removed. The removed connection are identified by an $\emptyset$ in table 2.3.

cluster chain. The data structure is built by parsing table 2.3 and appending cluster centers that link together into each cluster chain array.

The middle column of table 2.3 is chosen as the initial cluster center index to build a cluster chain with. The middle column might not correspond to the end-point of a cluster chain, so the chain will be build in two directions and appended together at the end. In the case of building an array from table 2.3, the starting index is 9, so the first link in this chain is set to the position of cluster center 9. The next cluster center, found by the k_1 index in column 9, is index 7, so the second link in this chain is set to the position of cluster center 7. The 7-th column in table 2.3 contains a 9 for k_1, which is already stored in the chain, so the k_2 index is used instead for the third link and the process is repeated. This chain building continues by traversing through table 2.3 and appending cluster centers until it reaches column 13, where there is an $\emptyset$ flag. This flag indicates that cluster center 13 is one of the ends of the chain. The remaining section of the chain is built by repeating the chain building process but starting with the k_2 index instead from cluster center 9. This second traversal will terminate at index 17, according to table 2.3, and the chain's array will be complete. Once both sections of the chain is built, they are appended together to start at cluster center 17, continue through cluster center 9, and end at cluster center 13. All the indices that are used to build the chain 17-to-13 are removed from table 2.3 and the process

is repeated until there are no indices remaining in the table.

This method for building the chain data structure completes a set of chain arrays for open-loop chains, but there must be an additional process during the traversal to prevent closed-loops in the chain. Every time a new cluster center is added to a chain, there is a check to verify that the cluster center's index doesn't match the cluster center index at the beginning of the chain. Matching index values indicates the existence of a closed loop. A closed-loop chain, once identified, is broken by finding the index in the chain with the smallest angle created by its two neighbours. The longest connection formed between the index with the smallest angle and its two neighbours is the one that is removed. The order of the chain in the data structure might need to adjusted to reflect this connection break but the result creates an open-loop chain array.

Figure 2.19 presents the results of the loop-breaking method and creation of the chain data structure from figure 2.17. This solution removes the loops and web problems presented in figure 2.17 with the consequence of breaking some of the chain connections that actually traced the sea-floor profile. These breaks in the chain, along the profile, are re-connected by projecting outwards from the ends of each chain and searching for ideal chains to connect to. However, before re-connecting the breaks in the chain and other discontinuities, the curvature of the chains need to be smoothed out. Large changes in curvature tend to be introduced by low SNR regions during clustering and by breaking loops after connecting cluster centers. Figure 2.20 present two examples of large curvature regions that make it difficult to accurately project outwards from the end of the chain to connect with other chains along the sea-floor profile. The following method smooths these fluctuations out so that the process for joining cluster chains will be more effective.

Chain Curvature Constraint

The examples presented in figure 2.20 indicate the need for a curvature constraint of the cluster chains to better approximate the sea-floor profile. This improvement will also aid the process of connecting chains that trace the sea-floor profile because projecting out from the chain to create connections will not be influenced by large fluctuations in the chains. The curvature in the chains are reduced by applying a modified FCM clustering method to the chains of cluster centers. The modification to the clustering method introduces a curvature penalty term (suggested by [22]) applied to the traditional FCM cost function (2.4). The piecewise linear second order difference of the chain is used in the modified cost function to minimize the curvature at each cluster center. Using β as the constraint coefficient, the new objective function is as follows:

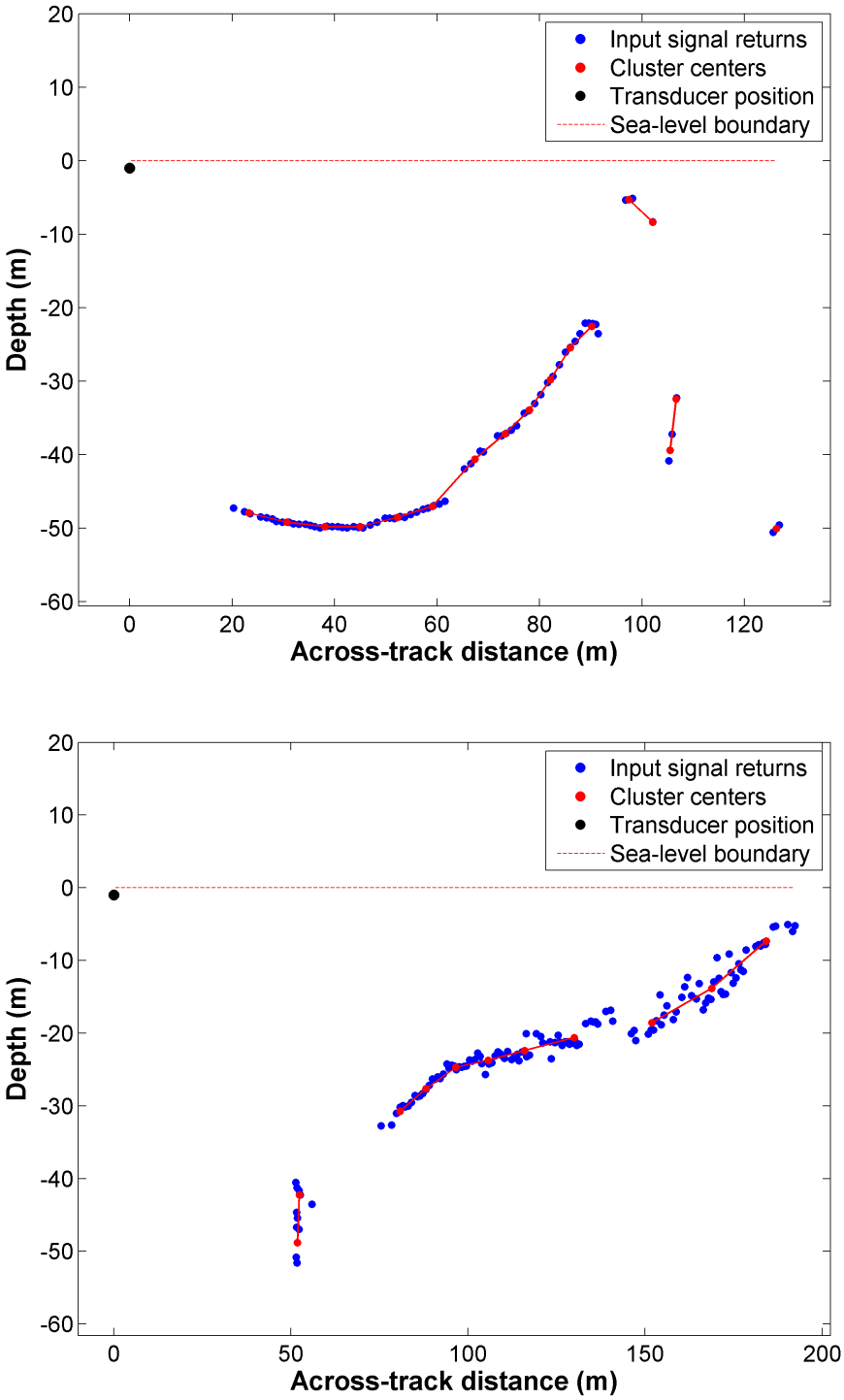

Figure 2.19: Breaking chain loops examples with complex webs (*top*), loops within the chain (*top*), and loops at the ends of the chains (*top and bottom*) removed from figure 2.17. Breaking the loops through the non-reciprocating connection method tends to introduce breaks in chains that trace the sea-floor profile.

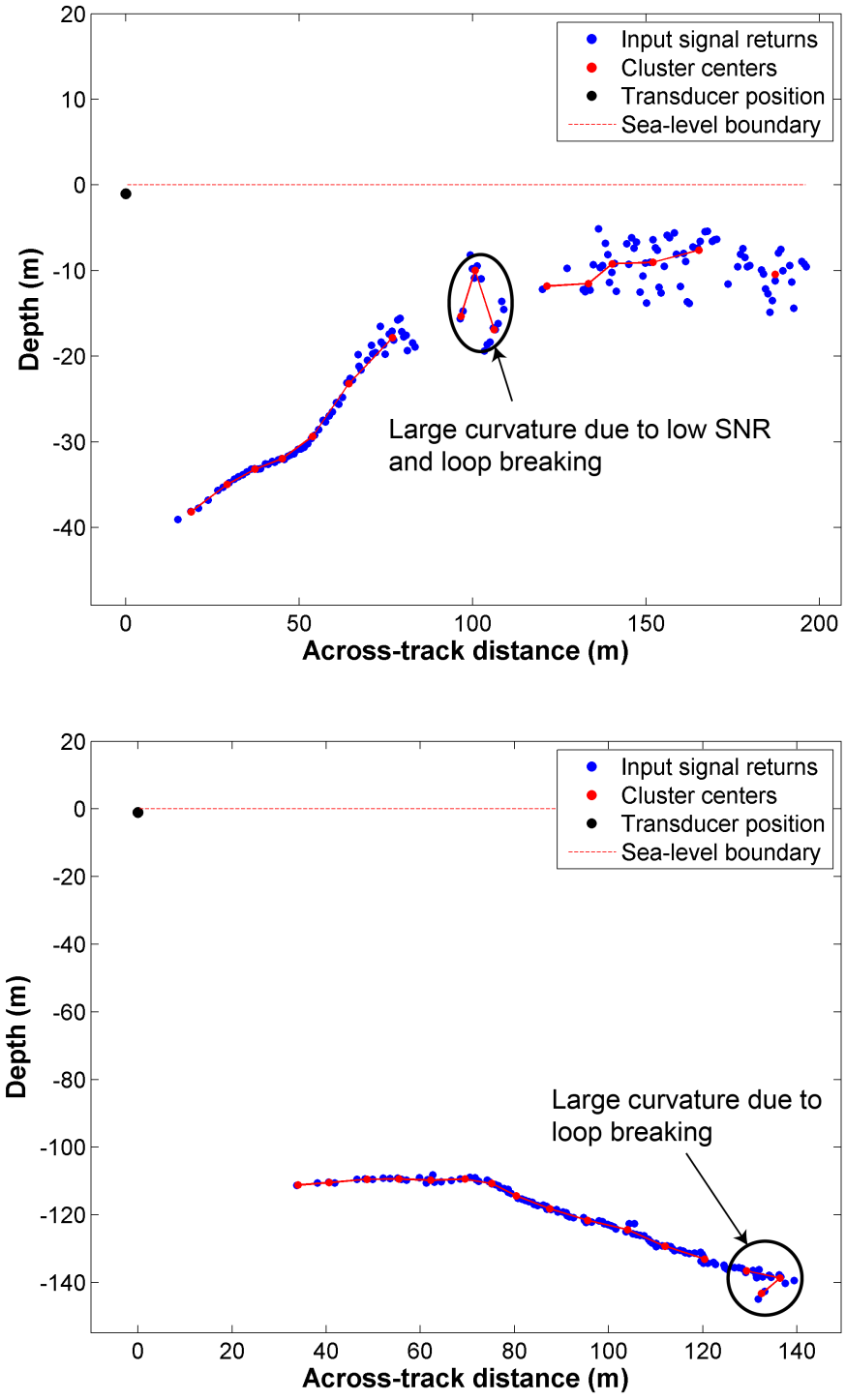

Figure 2.20: Cluster chains with large curvature regions. They are caused by low SNR regions and the breaking chain loop process. The curvature of the chains must be constrained before the chains can be connected along the sea-floor profile.

$$\min_{\mathbf{U},\mathbf{C}}\{J_{FCM}(\mathbf{U},\mathbf{C};\mathbf{X}) = \sum_{i=1}^{N}\sum_{j=1}^{C}\mu_{ij}^{m}\|\mathbf{x_i}-\mathbf{c_j}\|^2 + \beta\sum_{j=2}^{C-1}\|\mathbf{c_{j-1}}-2\mathbf{c_j}+\mathbf{c_{j+1}}\|^2\}. \tag{2.17}$$

For the sake of clarity, (2.17) only describes the FCM with a curvature penalty term for a single cluster chain. Because of this simplification with (2.17), the constrained FCM must be applied to every cluster chain that exists in the system. Additionally, it is important to note that the constraint summation in (2.17) ranges from $j = [2, C-1]$ because a second order difference doesn't exist for the end points of the chains. Because there is no membership matrix, $\mathbf{U}$, in the penalty term, the expectation-step ($\frac{\partial J_{FCM}(U,C;X)}{\partial \mu_{ij}} = 0$) equation remains the same as (2.5). However, the maximization step ($\frac{\partial J_{FCM}(U,C;X)}{\partial \mathbf{c_j}} = 0$) will be heavily influenced by the penalty. The solution for the maximization step will differ depending on the location of the cluster center in the chain because the partial derivatives near chain ends contain fewer cluster points. Additionally, Yan didn't address the special cases with only one, two, or three cluster centers in a chain because his data set never produced short enough chains. In the same way that (2.17) only describes the constraint for a single chain, the following partial derivative equations only apply to a single chain.

The following equations describe the partial derivative cases of (2.17) with respect to the cluster centers in a chain, where C is the number of clusters in the chain:

- $C = 1, 2$:

$$\sum_{i=1}^{N}\mu_{ij}^{m}(\mathbf{c_j}-\mathbf{x_i}) = 0$$

- $C = 3$:

$$\begin{aligned}
\beta(\mathbf{c_{j+2}}-2\mathbf{c_{j+1}}+\mathbf{c_j}) + \sum_{i=1}^{N}\mu_{ij}^{m}(\mathbf{c_j}-\mathbf{x_i}) &= 0 \quad for \quad j=1\\
-2\beta(\mathbf{c_{j+2}}-2\mathbf{c_{j+1}}+\mathbf{c_j}) + \sum_{i=1}^{N}\mu_{ij}^{m}(\mathbf{c_j}-\mathbf{x_i}) &= 0 \quad for \quad j=2\\
\beta(\mathbf{c_j}-2\mathbf{c_{j-1}}+\mathbf{c_{j-2}}) + \sum_{i=1}^{N}\mu_{ij}^{m}(\mathbf{c_j}-\mathbf{x_i}) &= 0 \quad for \quad j=3
\end{aligned}$$

- $C > 3$:

$$\begin{aligned}
\beta(\mathbf{c_{j+2}} - 2\mathbf{c_{j+1}} + \mathbf{c_j}) + \sum_{i=1}^{N} \mu_{ij}^m (\mathbf{c_j} - \mathbf{x_i}) &= 0 \quad for \quad j = 1 \\
\beta(\mathbf{c_{j+2}} - 4\mathbf{c_{j+1}} + 5\mathbf{c_j} - 2\mathbf{c_{j-1}}) + \sum_{i=1}^{N} \mu_{ij}^m (\mathbf{c_j} - \mathbf{x_i}) &= 0 \quad for \quad j = 2 \\
\beta(\mathbf{c_{j+2}} - 4\mathbf{c_{j+1}} + 6\mathbf{c_j} - 4\mathbf{c_{j-1}} - \mathbf{c_{j-2}}) + \sum_{i=1}^{N} \mu_{ij}^m (\mathbf{c_j} - \mathbf{x_i}) &= 0 \quad for \quad 3 \leq j \leq C-2 \\
\beta(-2\mathbf{c_{j+1}} - 5\mathbf{c_j} - 4\mathbf{c_{j-1}} - \mathbf{c_{j-2}}) + \sum_{i=1}^{N} \mu_{ij}^m (\mathbf{c_j} - \mathbf{x_i}) &= 0 \quad for \quad j = C-1 \\
\beta(\mathbf{c_j} - 2\mathbf{c_{j-1}} + \mathbf{c_{j-1}}) + \sum_{i=1}^{N} \mu_{ij}^m (\mathbf{c_j} - \mathbf{x_i}) &= 0 \quad for \quad j = C
\end{aligned}$$

Solving for $\mathbf{c_j}$, the cluster center positions are updated using the following equations:

- $C = 1, 2$:

$$\mathbf{c_j} = \frac{\sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{\sum_{i=1}^{N} \mu_{ij}^m}.$$

- $C = 3$:

$$\begin{aligned}
\mathbf{c_j} &= \frac{\beta(-\mathbf{c_{j+2}} + 2\mathbf{c_{j+1}}) \sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{\beta + \sum_{i=1}^{N} \mu_{ij}^m} \quad for \quad j = 1 \\
\mathbf{c_j} &= \frac{2\beta(\mathbf{c_{j+1}} + \mathbf{c_{j-1}}) \sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{4\beta + \sum_{i=1}^{N} \mu_{ij}^m} \quad for \quad j = 2 \\
\mathbf{c_j} &= \frac{\beta(2\mathbf{c_{j-1}} - \mathbf{c_{j-2}}) \sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{\beta + \sum_{i=1}^{N} \mu_{ij}^m} \quad for \quad j = 3
\end{aligned}$$

- $C > 3$:

$$\mathbf{c_j} = \frac{\beta(-\mathbf{c_{j+2}} + 2\mathbf{c_{j+1}}) \sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{\beta + \sum_{i=1}^{N} \mu_{ij}^m} \qquad for \quad j = 1$$

$$\mathbf{c_j} = \frac{\beta(-\mathbf{c_{j+2}} + 4\mathbf{c_{j+1}} + 2\mathbf{c_{j-1}}) \sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{5\beta + \sum_{i=1}^{N} \mu_{ij}^m} \qquad for \quad j = 2$$

$$\mathbf{c_j} = \frac{\beta(-\mathbf{c_{j+2}} + 4\mathbf{c_{j+1}} + 4\mathbf{c_{j-1}} - 4\mathbf{c_{j-2}}) \sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{6\beta + \sum_{i=1}^{N} \mu_{ij}^m} \qquad for \quad 3 \leq j \leq C-2$$

$$\mathbf{c_j} = \frac{\beta(2\mathbf{c_{j+1}} + 4\mathbf{c_{j-1}} - \mathbf{c_{j-2}}) \sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{5\beta + \sum_{i=1}^{N} \mu_{ij}^m} \qquad for \quad j = C-1$$

$$\mathbf{c_j} = \frac{\beta(2\mathbf{c_{j-1}} - \mathbf{c_{j-2}}) \sum_{i=1}^{N} \mu_{ij}^m \mathbf{x_i}}{\beta + \sum_{i=1}^{N} \mu_{ij}^m} \qquad for \quad j = C$$

Figure 2.21 presents the improvements to the large curvature regions in figure 2.20 by applying (2.17), with $\beta = 4.0$, to the cluster chains. The large curvature region in the top example from figure 2.21 shows substantial improvement, however the cluster chain is not consistent with the sea-floor profile. Instead, this cluster chain traces the noise in the region and makes it more difficult to connect this cluster chain to the others along the sea-floor profile. The large curvature region in the bottom example is also improved and traces the sea-floor profile very well. Connecting this improved cluster chain to the other chain along the sea-floor profile is now very straight forward.

This improvement in cluster chain curvature will greatly improve the method of connecting the chains because the joining algorithm projects outwards from the end of the chains to find ideal chains to connect to. If there is a large curvature fluctuation, there will be no certainty that the projection outwards from the chain has any relevance to the actual sea-floor profile.

Although the complexity of each iteration in this process is $O(CN)$, (as with (2.4)), the computational bandwidth is significantly larger with the added constraint. This curvature

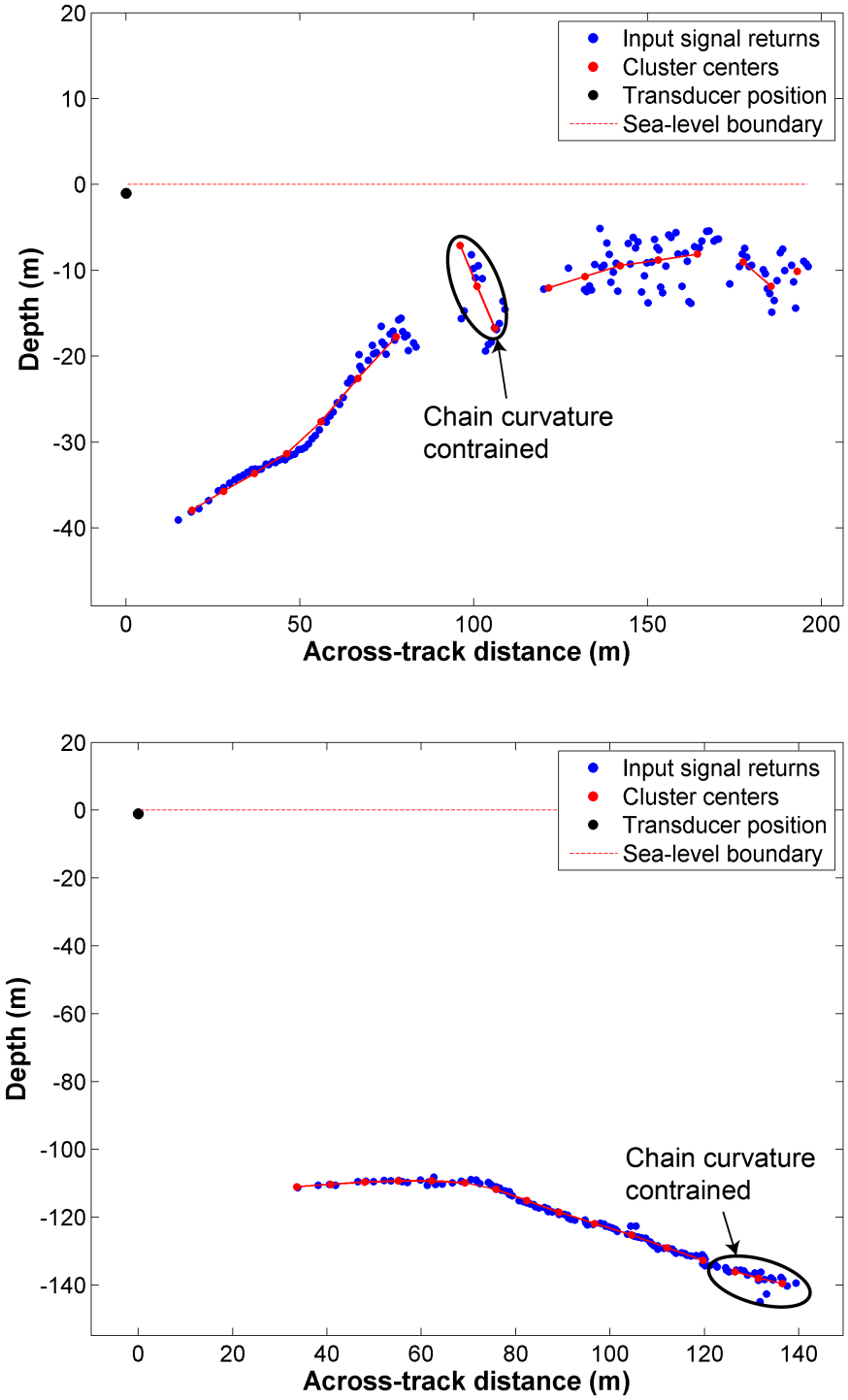

Figure 2.21: Curvature-constrained cluster center chain examples. The regions of large curvature indicated in figure 2.20 are largely reduced to produce smooth curved chains.

constraint process can easily become the largest bottleneck in the system if the data set or cluster numbers grow too large. However, because the cluster centers have already converged to local minimum from the FCM with repulsion algorithm, the constrained FCM process largely focuses on locally optimal chain curvature and it was observed to converge four times faster than the FCM with repulsion algorithm. The preconditioned state from the FCM algorithm makes the curvature constraint improvement to the cluster chain desirable and computationally efficient.

Joining Chains

At this point in the FCT algorithm, the cluster centers are usually well positioned along the sea-floor profile and grouped into chains that trace along the curves. However the sea-floor profile tends to be traced by multiple chains of clusters with breaks between them. These chains must be connected together in a principled way to create a single chain that traces the sea-floor profile. This problem, was not addressed in [22], because the data set was continuous and well distributed breaks in the chain were never encountered.

Raghupathy et al. present a method using Radon transforms from image processing to extract a line's edge and direction [44]. With this information, they project outwards from a break, or intersection, in a line to recover the continuation of the line afterwards. Similar to [44], the solution presented in this thesis connects the cluster chains by projecting outwards from the end of a chain to a sample point and testing if the start of any other chain is positioned within an acceptable range of horizontal and perpendicular distances to the sample point.

Before projecting outwards from the cluster chains, the "start" and "end" links of the chain must be assigned. Each chain is re-arranged so the starting link in the array has the smallest across-track distance – labeled as the "start". The opposite end of the chain is labelled as the "end" and should be the furthest cluster center in across-track distance. All intermediate points in the chain are sequential links.

After the chain re-arranging is complete, the distances between each chain's starting link to all remaining chains' ending links are calculated. The pair of chains with the shortest distance between one chain's starting link and the other's ending link is tested first. Subsequent tests are executed in order of shortest remaining distances.

The test for connecting chain pairs begins by projecting outwards from the end-link of a chain along its best-fit line. The best-fit line is determined from the last three links in the chain-end using least squares, with $\mathbf{c_3}$, $\mathbf{c_2}$, $\mathbf{c_1}$ as the last, second last and third last cluster centers in the chain, respectively. The chain links, $\mathbf{c_i} = \langle c_{x,i}, c_{z,i} \rangle$ for $i = 1, 2, 3$, are projected onto the best-fit line to give the new points $\tilde{\mathbf{c}}_\mathbf{i} = \langle c_{x,i}, \tilde{c_{z,i}} \rangle$ as follows:[4]:

[4]Since the matrix inversion in the pseudo-inverse is always 2x2, the inverse is hard-coded in the C++ code to avoid any performance issues with numerical matrix inversions.

$$\Phi(\mathbf{c_x}) = \begin{bmatrix} c_{x,1} & 1 \\ c_{x,2} & 1 \\ c_{x,3} & 1 \end{bmatrix} \quad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \quad \mathbf{c_z} = \begin{bmatrix} c_{z,1} \\ c_{z,2} \\ c_{z,3} \end{bmatrix}$$
$$\mathbf{w} = (\Phi^T\Phi)^{-1}\Phi^T\mathbf{c_z} \tag{2.18}$$
$$\tilde{\mathbf{c_z}} = \Phi(\mathbf{c_x})\mathbf{w}$$

Using the best fit line from (2.18), the algorithm projects outward from the end of the chain, in the most likely direction the chain might proceed in, represented as unit vector $\hat{\mathbf{v}}$:

$$\hat{\mathbf{v}} = \frac{\tilde{\mathbf{c_3}} - \tilde{\mathbf{c_1}}}{\|\tilde{\mathbf{c_3}} - \tilde{\mathbf{c_1}}\|}.$$

A point along the best-fit line from (2.18) is created by projecting out in the direction of $\hat{\mathbf{v}}$, labelled as $\mathbf{c_s}$. This "sample" point is used to test if the start of the second chain in the comparison will be an acceptable connection to the end of the first chain. Point $\mathbf{c_s}$ is projected out by a distance d_s from the projected end point $\tilde{\mathbf{c_3}}$ by:

$$\mathbf{c_s} = \tilde{\mathbf{c_3}} + d_s\hat{\mathbf{v}}.$$

The starting link of the second chain, $\mathbf{c_4}$, is compared to the sample point $\mathbf{c_s}$ by the vector $\mathbf{u}$. This $\mathbf{u}$ vector is projected onto $\hat{\mathbf{v}}$ to provide parallel and perpendicular vectors between $\mathbf{c_4}$ and $\mathbf{c_s}$ in (2.19). The magnitude of both vectors determine whether the two chains should be connected.

$$\begin{aligned} \mathbf{u} &= \mathbf{c_4} - \mathbf{c_s} \\ proj_{\hat{\mathbf{v}}}(\mathbf{u}) &= \tfrac{\mathbf{u}\cdot\hat{\mathbf{v}}}{\hat{\mathbf{v}}\cdot\hat{\mathbf{v}}}\hat{\mathbf{v}} \\ perp_{\hat{\mathbf{v}}}(\mathbf{u}) &= \mathbf{u} - proj_{\hat{\mathbf{v}}}(\mathbf{u}) \end{aligned} \tag{2.19}$$

If the magnitudes of vectors $proj_{\hat{\mathbf{v}}}(\mathbf{u})$, and $perp_{\hat{\mathbf{v}}}(\mathbf{u})$ are less than their associated threshold values, d_{proj} and d_{perp} respectively, then the two chains are connected by joining $\mathbf{c_3}$ to $\mathbf{c_4}$. If the pair of chains are joined together, the entire new system of cluster chains are re-processed with the constrained FCM method from (2.17) and the whole joining process is repeated. If the chains are not joined, then another pair of chains are compared until all the combinations of chains are compared.

Figure 2.22 demonstrates the process of comparing two chains using the method described above. The first chain, consisting of $\mathbf{c_1}$, $\mathbf{c_2}$, $\mathbf{c_3}$, is projected onto the best fit line (black dotted line) specified by $\tilde{\mathbf{c_1}}$ and $\tilde{\mathbf{c_3}}$. The sample point, $\mathbf{c_s}$, is then determined by projecting outwards from $\tilde{\mathbf{c_3}}$ along the best fit line's unit vector, $\hat{\mathbf{v}}$. The vector between the

sample point and the starting link of the second chain, $\mathbf{c_4}$, is represented by $\mathbf{u}$ and used to find the projection and perpendicular vectors between $\mathbf{c_s}$ and $\mathbf{c_4}$. The projection, $proj_{\hat{\mathbf{v}}}(\mathbf{u})$, and perpendicular, $perp_{\hat{\mathbf{v}}}(\mathbf{u})$, vectors are compared against the threshold values d_{proj} and d_{perp}, respectively. The region around $\mathbf{c_s}$ that satisfy both threshold values is indicated with a dotted red line. In figure 2.22, the starting link of the second chain ($\mathbf{c_4}$) lies outside of the boundary and the two chains will not be connected.

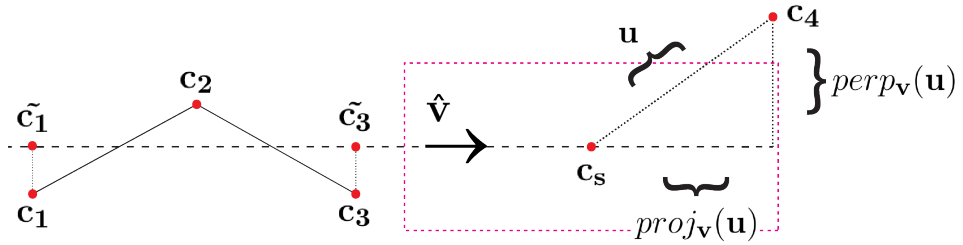

Figure 2.22: Projection diagram for joining cluster chains. A sample point, $\mathbf{c_s}$ is create by projecting out from $\mathbf{c_3}$ along the best fit line (black hashed line). The start link of the second chain, $\mathbf{c_4}$, does not fall within the red dotted-line boundary (specified by the project and perpendicular vector magnitudes) and the chains will not be connected.

Figures 2.23 and 2.24 demonstrate that breaks in the chains from figures 2.19 and 2.21 are connected by this joining method to trace the sea-floor profile with a single chain. In all four examples, the joining method is executed with the parameters $d_s = 10m$, $d_{proj} = 20m$, $d_{perp} = 10m$.

The chain joining method reliably connects chains together that are separated due to nadir regions and loop-breaking. However, separations due to shadows are not resolved. The top example in figure 2.25 highlights a chain separation due to a shadow region even after the joining method is complete. An additional step is required to properly trace a sea-floor profile with shadows.

Additionally, the top example in figure 2.23 demonstrates an instance where a second, non-sea-floor, chain also exists. These additional chains are due to cluster centers that converge on outlier data. Recognizing which chain accurately describes the sea-floor is an important step for identifying sea-floor bathymetry, but resolving chain separations due to shadows must be handled first.

Casting Shadows

Because sonar systems are based on line-of-sight, there will be no echo from the sea-floor beyond the horizon from the transducer. Therefore, there is no signal return information gathered by the sonar system to describe the sea-floor in that region. This lack of information creates large gaps between chains during the FCT algorithm and the chains on either end of the gap will likely not be connected during the joining method.

The remaining cluster chains are tested against each other and connected by comparing

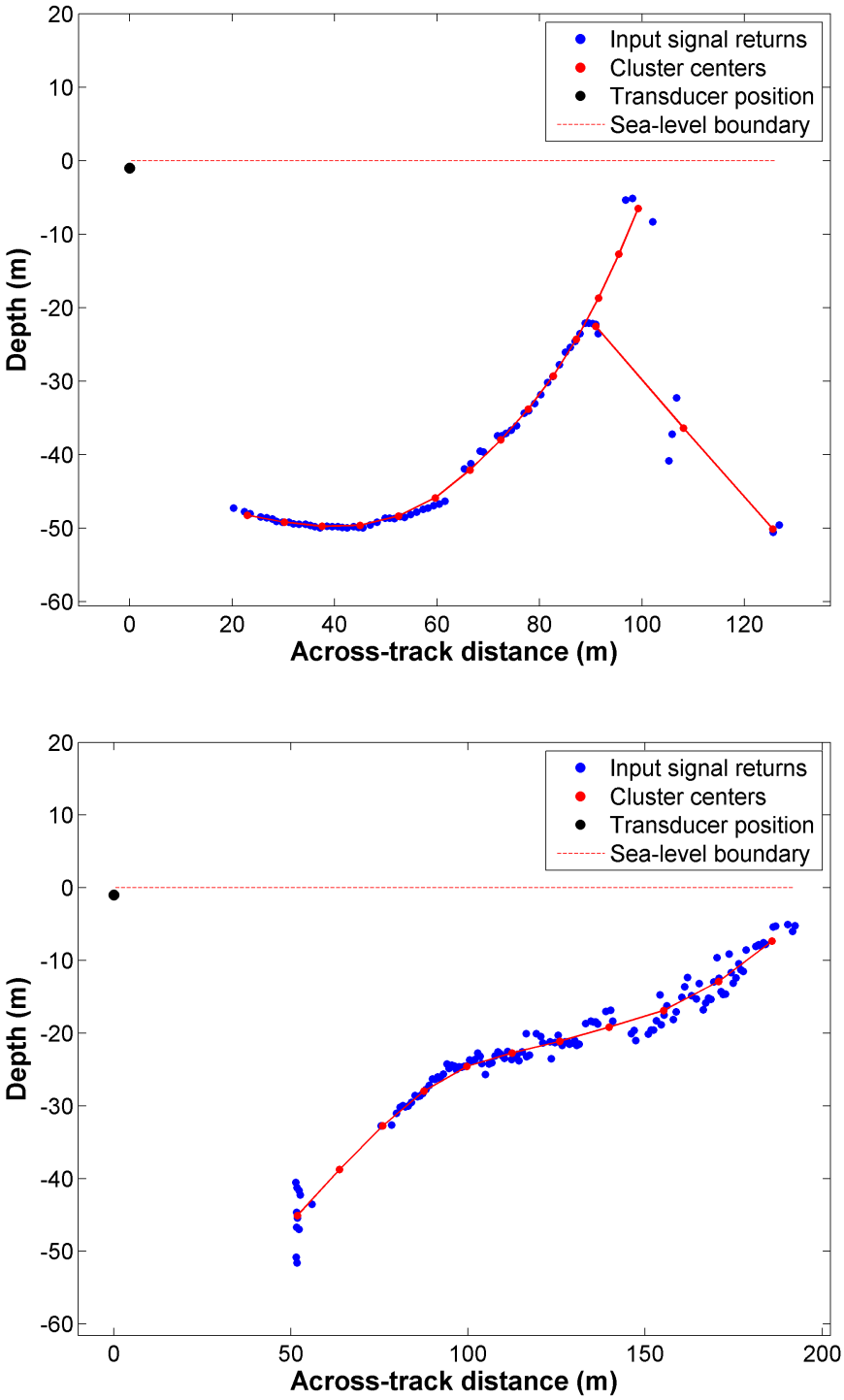

Figure 2.23: Examples of joined cluster chains from figure 2.19. The chain joining method result for both examples trace their respective sea-floor profile with a single cluster chain. A second chain (*top*) persists because it traces through outlier data and wasn't connected to the sea-floor chain.

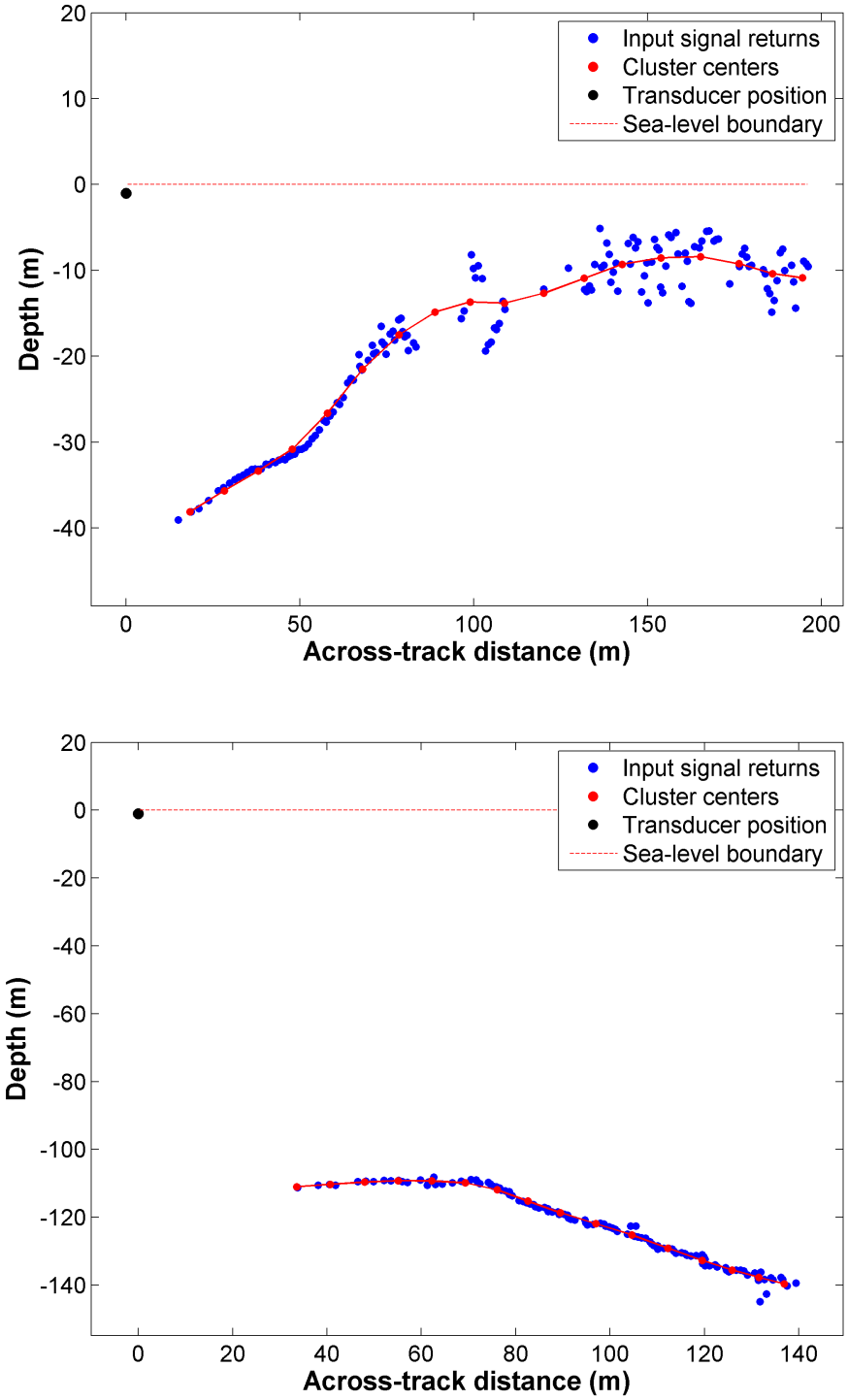

Figure 2.24: Examples of joined cluster chains from figure 2.21. The chain joining method result for both examples trace their respective sea-floor profile with a single cluster chain.

the physical angles (with respect to the transducer) for the end-link of the first chain and the start-link of the second chain. The position of each link in the chain is currently stored in cartesian space, so the physical angle must be computed before the links can be compared. Equations 1.2, and 1.3 describe the process to convert a data point from physical angle and range space to cartesian space, so determining the physical angle from cartesian data for each link is just the inverse of these equations. If a data point is positioned at x and z in the across-track and depth dimensions, and the transducer is tilted at an angle t_ϕ and at a depth d_t relative to sea-level, then the physical angle, ϕ, is determined by:

$$\phi = \tan^{-1}(\frac{d_t - z}{x}) - t_\phi.$$

With the physical angle for the end-link and start-link of the chain pair, the chains are joined together if the end-link of the first chain and the start-link of the second chain differ by less than $\Delta\phi = 20°$. This parameter setting was chosen because it was observed to connect the shadow regions between cluster chains along the sea-floor without accidentally connecting them to chains in outlier regions. If a chain pair is connected by shadow casting, the new set of cluster chains is processed with the curvature contrained FCM method to smooth out any large jumps in the chain that are introduced by the new connection.

Figure 2.25 presents an example (*top*) of a shadow region between two sea-floor chains that remain separated after the joining method. The chains are connected (see *bottom* example of figure 2.25) because they pass the test that compares the difference in physical angle between the two links (which differ by less than 20°). Finally, with the shadow regions resolved, the system is now capable of tracing the curves of the sea-floor profile in situations of nadir and shadow regions, outlier data and low SNR regions. However, instances of a cluster chain tracing outlier data (see *top* example of figure 2.23) still exist. To complete the entire sea-floor profiling algorithm by isolating the sea-floor bathymetry, the cluster chain corresponding to the sea-floor profile must be identified.

Isolating Sea-floor Cluster Chain and Bathymetry

The final process in the the FCT algorithm isolates the cluster chain that traces the sea-floor profile and the corresponding bathymetry data for FCT cases that contain multiple cluster chains. Figure 2.26 presents two cases with a cluster chain that describes the sea-floor profile and a cluster chain that describes outlier data. The top example contains a cluster chain that traces through a water-column object. The bottom example contains a cluster chain that traces through multi-path and a small section of the sea-floor profile.

Ideally, the cluster chain that traces the sea-floor profile will be considerably longer and contain more cluster centers than cluster chains that span outlier data regions. While this is generally observed to be true, there is no reason why an outlier region couldn't contain a long cluster chain with many cluster centers. However, it is far more likely that the average

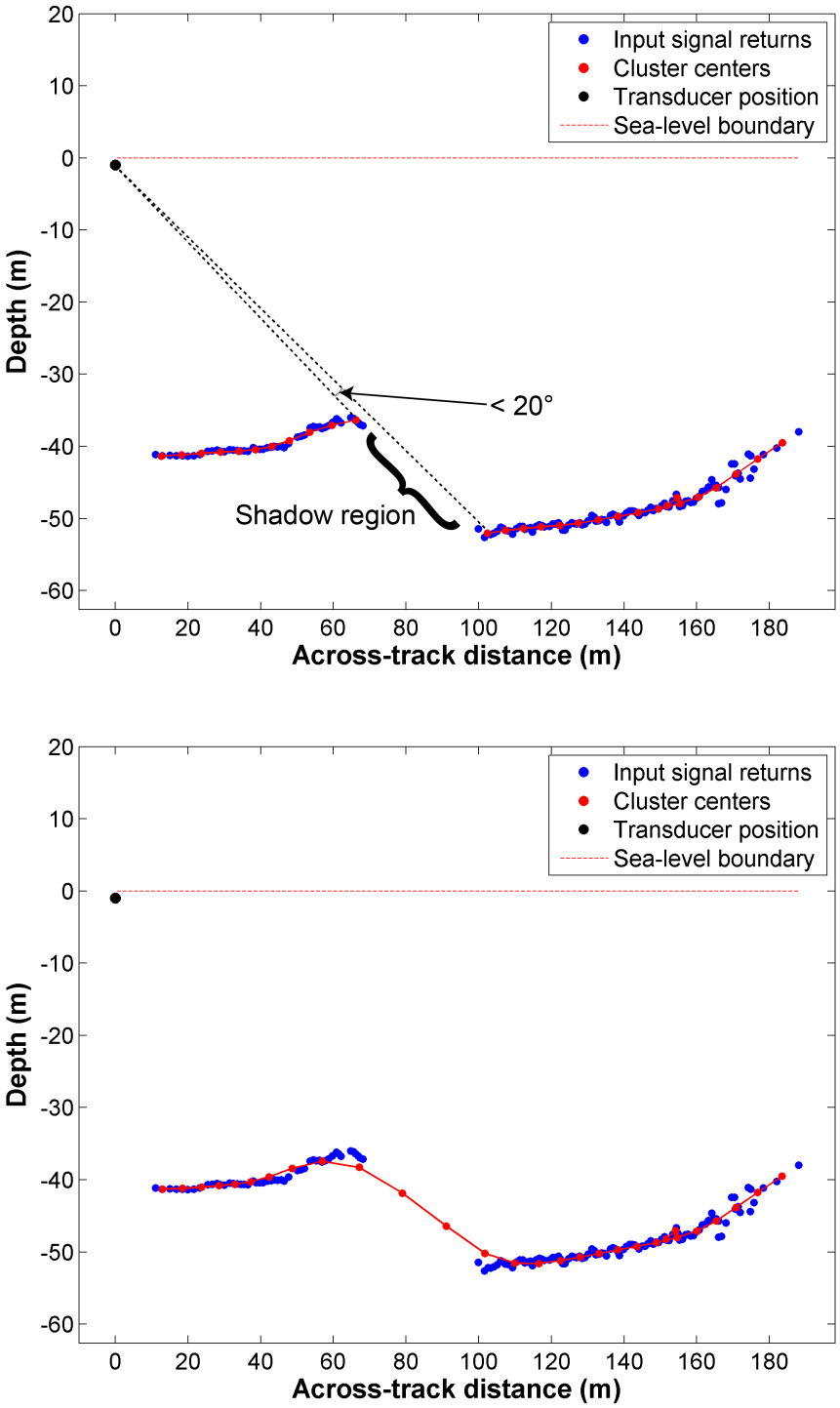

Figure 2.25: Shadow casting to join cluster chains. The two cluster chains between the shadow region are separated in their physical angles by less than 20° (*top*). They are connected across the shadow region to create a final cluster chain that traces the sea-floor profile (*bottom*).

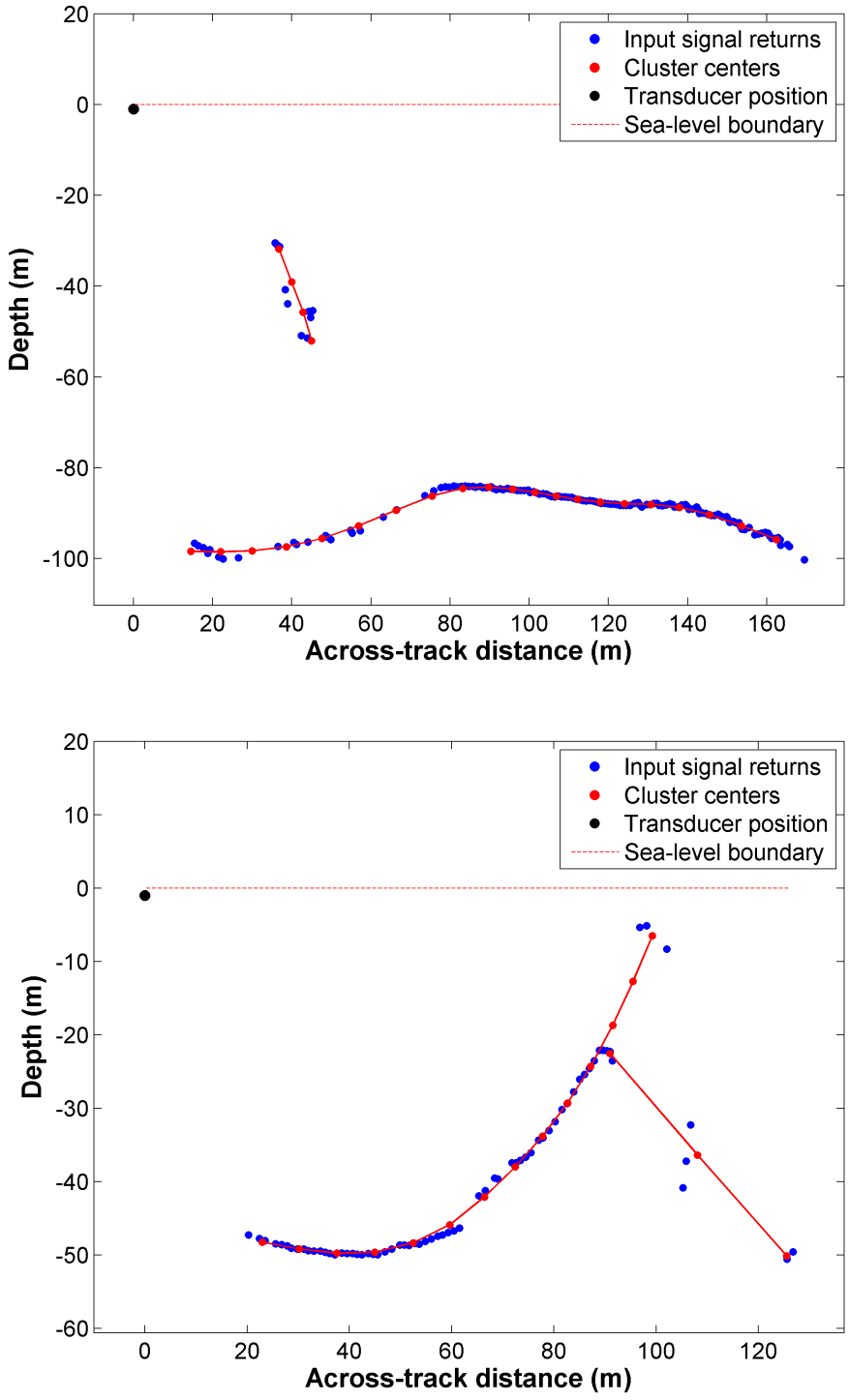

Figure 2.26: Cluster chains that trace the sea-floor and outliers. In both cases, a single cluster chain traces the sea-floor profile while another chain traces a water-column object (*top*) or outlier data (*bottom*).

distance from a bathymetry data point to the cluster chain will be sufficiently smaller for a sea-floor profiling chain than an outlier chain. The chain isolation approach combines: the average distances from bathymetry data points to the chain, the number of clusters in the chain and the length of the chain to determine the cluster chain that most likely traces the sea-floor profile.

It's important to clarify that only bathymetry data points reasonably close to a chain should count in the average distance, not the entire set of data points. Also, because the goal of this whole chapter is to return bathymetry data associated with the sea-floor, this is the perfect time to group bathymetry data to cluster chains and return the bathymetry data that associate with the ideal cluster chain.

A bathymetry data point is assigned to a cluster chain by comparing its projected and orthogonal distance to each vector of cluster center pairs in a cluster chain. This projection approach is very similar to the approach described in figure 2.22. In this case: $\mathbf{c_1}$ and $\mathbf{c_2}$ are two neighbouring clusters in a chain and $\mathbf{x}$ is a bathymetry data point. The vectors between the three points are labelled as $\mathbf{v}$, $\mathbf{u_1}$, and $\mathbf{u_2}$ and specified by:

$$\begin{aligned}\mathbf{v} &= \mathbf{c_1} - \mathbf{c_2},\\ \mathbf{u_1} &= \mathbf{x} - \mathbf{c_1},\\ \mathbf{u_2} &= \mathbf{x} - \mathbf{c_2}.\end{aligned}$$

The projection vectors between the data point and the two cluster centers are calculated as follows:

$$\begin{aligned}proj_{\mathbf{v}}(\mathbf{u_1}) &= \tfrac{\mathbf{u_1}\cdot\mathbf{v}}{\mathbf{v}\cdot\mathbf{v}}\mathbf{v},\\ proj_{\mathbf{v}}(\mathbf{u_2}) &= \tfrac{\mathbf{u_2}\cdot\mathbf{v}}{\mathbf{v}\cdot\mathbf{v}}\mathbf{v}.\end{aligned}$$

Calculating both projection vectors is important because it allows the method to determine if the data point's projection is between the two cluster centers. For example, if $\|proj_{\mathbf{v}}(\mathbf{u_1})\| + \|proj_{\mathbf{v}}(\mathbf{u_2})\| > \|\mathbf{v}\|$, then the data point's projection is not located between the cluster centers.

The perpendicular vector from vector $\mathbf{v}$ to point $\mathbf{x_3}$, labelled as $perp_{\mathbf{v}}(\mathbf{u_1})$ is determined by:

$$perp_{\mathbf{v}}(\mathbf{u_1}) = \mathbf{u_1} - proj_{\mathbf{v}}(\mathbf{u_1}). \tag{2.20}$$

Only one perpendicular vector was calculated in this instance because $perp_{\mathbf{v}}(\mathbf{u_2})$ would be identical.

The projection and perpendicular vector magnitudes provide a reasonable criteria to determine if the bathymetry data point should be associated with the cluster chain in question. The data point must pass the following tests:

$$\begin{gathered} \|proj_{\mathbf{v}}(\mathbf{u_1})\| + \|proj_{\mathbf{v}}(\mathbf{u_2})\| \leq \|\mathbf{v}\| + d_{proj} \\ and \\ \|perp_{\mathbf{v}}(\mathbf{u_1})\| \leq d_{perp}. \end{gathered} \tag{2.21}$$

where d_{proj} is a tolerance distance on either side of the cluster center pairs and d_{perp} is a tolerance distance off of the chain pair vector $\mathbf{v}$. The parameter settings used in this thesis are $d_{proj} = 10m$ and $d_{perp} = 5m$.

The first test in (2.21) ensures that the bathymetry data point projection falls inside, or very close to, the line segment between the cluster pair. The second test ensures the bathymetry data point is positioned reasonably close to the line segment. Figure 2.27 provides a visual explanation of the method, where the red dotted-line boundary represents the region that a data point, $\mathbf{x_3}$ must be positioned within to be considered acceptable. The boundary is formed based on the region around a cluster center pair that would pass the tests in (2.21). In the example presented in figure 2.27, the bathymetry data point is just slightly outside of the red dotted-line boundary. For this case, the projection test passes, but the perpendicular test fails.

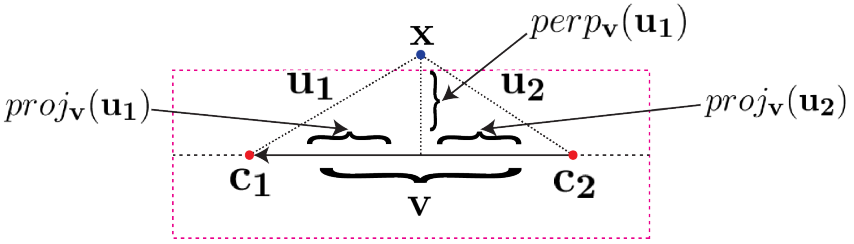

Figure 2.27: Projection diagram for assigning bathymetry data points to cluster chains. The data point, $\mathbf{x}$ is tested against the line segment created by $\mathbf{c_1}$ and $\mathbf{c_2}$. The data point does not fall within the red dotted-line boundary (based on the magnitude of both projection vectors and perpendicular vectors) and will not be assigned to the chain based on this cluster pair, $\mathbf{c_1}$ and $\mathbf{c_2}$.

If both tests in (2.21) are satisfied, the bathymetry data point, $\mathbf{x}$, is labelled as bathymetry data associated with the cluster chain formed by $\mathbf{c_1}$ and $\mathbf{c_2}$. If the test fails, then the next cluster pair in the chain is used to test the bathymetry data point. This process repeats for all bathymetry data against all cluster chains.

The average squared distance from the bathymetry data associated with a cluster chain is used as partial criteria for determining the ideal chain. The perpendicular vector for a bathymetry data point, $\mathbf{x_i}$, to its associated cluster chain, $\mathbf{C_j}$, from (2.20) will be now re-labelled as $\mathbf{d_{\perp,i}}$ for ease of notation. The average squared perpendicular distance (or error), ϵ_j, for all the data points assigned to $\mathbf{C_j}$ is specified by:

$$\epsilon_j = \frac{1}{N_j} \sum_{\{\mathbf{x_i} \in \mathbf{C_j}\}}^{N_j} \|\mathbf{d}_{\perp,\mathbf{i}}\|^2, \tag{2.22}$$

where $\{\mathbf{x_i} \in \mathbf{C_j}\}$ is the set of data points assigned to $\mathbf{C_j}$ and N_j is the length of that set.

The average error between the bathymetry data points and the cluster chains provide a reasonable measure of curve tracing accuracy, but the length of the chain and number of clusters is also important. The method for determining the optimal chain should find the cluster chain that maximizes the number of clusters in a chain, C_i, and the squared length of the chain, $\sum_{j=1}^{C_i-1} \|\mathbf{c_{j+1}} - \mathbf{c_j}\|^2$, and minimizes the average error (or maximizes the inverse error with $\frac{1}{\epsilon_i^2+1}$). The following equation returns the sea-floor profiling cluster chain, labelled as $\mathbf{C_{FCT}}$, according to the three conditions specified:

$$\mathbf{C_{FCT}} = \max_i \{\frac{C_i}{\epsilon_i + 1} \sum_{j=1}^{C_i-1} \|\mathbf{c_{j+1}} - \mathbf{c_j}\|^2\}. \tag{2.23}$$

Figure 2.28 presents the $\mathbf{C_{FCT}}$ results by applying (2.23) to the examples in figure 2.26. In both cases, the cluster chain corresponding to the sea-floor is determined to be the ideal chain by (2.23) while the water-column and multi-path (top and bottom respectively) cluster chains are removed. Additionally, the bathymetry data corresponding to the sea-floor are captured by the chain that traces the sea-floor profile.

In figure 2.28, the signal returns from the initial input are represented by blue dots, while the signal returns that correspond to the sea-floor (i.e. the data points that fall within the red dotted-line boundary from figure 2.27) are represented by green dots. The green dots are intentionally smaller than the blue dots to visualize that they are from the FCT input data set. Both examples in figure 2.28 successfully identify and isolate the sea-floor signal returns.

2.4. Sea-floor Profiling Conclusion

This chapter presented an algorithm to isolate the sea-floor profile in bathymetry data with outliers. Given a dataset with non-sea-floor returns from the ocean surface and water column objects, as well as gaps in the sea-floor data due to shadows and nadir regions, the presented algorithm can still approximate the sea-floor profile from the dataset. In addition to the accuracy of the algorithm, it is computationally efficient with an average computation time of 7.40ms per ping.

The initial filtering methods in section 2.1 were only successful at isolating the sea-floor in trivial cases. However, these methods are computationally efficient and removing the trivial data points through the sea-level boundary and signal amplitude filters will significantly improve the overall computation time of the system. Based on efficiency alone, the

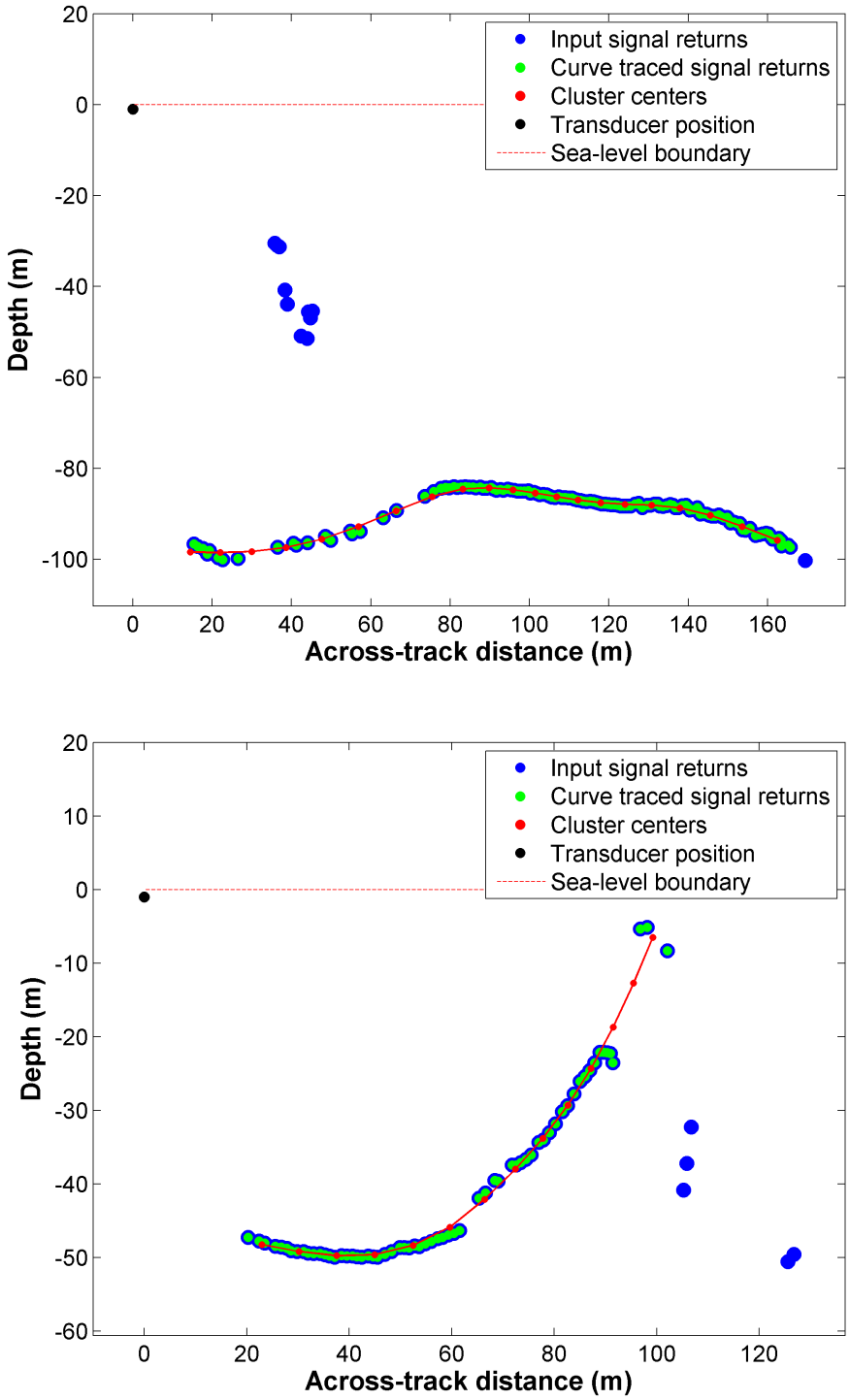

Figure 2.28: Examples of the isolated cluster chain and associated bathymetry data that traces the sea-floor from the examples of multiple cluster chains in figure 2.26. The original bathymetry data are indicated with blue dots, while the data that is associated with the sea-floor is indicated with smaller green dots (which overlap the original data).

two filters act as good initializing tools for the more complex sea-floor profiling algorithm.

The angle ray-casting method demonstrates the ability to remove outliers from the bathymetry data but it also tends to remove sea-floor signal returns. The removal of sea-floor data arises from rasterizing multiple signal returns into a single grid cell, thus losing data that is more localized than the grid cell spacing. Additionally, this data loss in sea-floor regions that have a shallow grazing angle with respect to the transducer. This data loss occurs because the signal returns tend to rasterize along the same row of the grid, where largely separated data points are removed. Because of the inability to preserve sea-floor detail while removing outliers and noise, this method is not a sufficient solution for extracting the sea-floor profile. However, enough sea-floor data is preserved from the angle-ray casting method that the output will coarsely approximate the sea-floor profile (with the possibility of a few remaining outlier data points). It is reasonable to say that the output from angle ray-casting approximates a converged solution to a local minimum for a fuzzy c-means clustering method. In addition, angle ray-casting is $O(N)$ complex where as fuzzy c-means is $O(kCN)$ complex (k iterations over N data points and C cluster centers). The efficiency of angle ray-casting and its approximation to the sea-floor profile make it a good initializer for (2.8).

The fuzzy c-means curve tracing method by Yan demonstrates a flexible foundation to trace the curve of point cloud data [22]. However, the data set presented in this thesis often comprises of point cloud data with varying distribution, gaps between curves and outlier data. These additional complications were not handled by [22]. With the additional of a cluster repulsion term, chain joining methods and algorithm engineering, the curve tracing method presented in this thesis provides an efficient and promising solution for sea-floor profiling within bathymetry data. Figure 2.28 clearly indicates that this method is capable of tracing the broad strokes of the sea-floor profile without being influenced by outlier data or truncated by gaps in the sea-floor bathymetry.

The computational bottleneck in the system is the FCM clustering methods, both with repulsion and curvature constraints. The repulsion is generally initialized with more cluster centers than the final output because of cluster center ejection. Having a larger initial set, having to handle ejections and having to reposition cluster centers after ejection make FCM with repulsion the most time consuming portion of the sea-floor profiling algorithm. The FCM with curvature constraints algorithm generally begins with chains that are already closely converged, so the number of iterations to reach convergence is typically small and the resulting computational bandwidth is smaller than the repulsion method. For the goal of this thesis, the entire sea-floor profiling algorithm is very efficient, but those interested in further performance should focus optimization effort on the curvature constrained FCM (e.g. multi-threading, etc.).

3. Sea-floor Classification using Neural Networks

This chapter presents a technique to classify different composition regions of the sea-floor by processing side-scan sonar images. The classification method processes a side-scan sonar image based on local variations in a sample region centered around each pixel in the image. For this thesis, the local variations are computed as an image's texture within the sample region and the computed values are classified using a machine learning technique.

There are many existing computer vision techniques that describe an image's texture information. Although each technique differs in the way it extracts the texture information, they all build a numerical value for each type of texture feature. These features are combined to form a vector that describes the texture of a particular sample region of an image. This process is iterated over the entire image by shifting the sample region by one pixel in the image until every pixel in the image has been processed by at least one sample region. The pixel at the center of each sample region is assigned the computed image texture feature vector of that particular sample region. The full image is then segmented by grouping pixels together based on the position of their feature vectors in some n-dimensional space (where n is the length of the feature vector). A supervised machine learning technique is used in this thesis to group and classify these feature vectors.

Segmenting a side-scan sonar image for this thesis requires two important steps: building the texture feature space for the full image and applying a machine learning process to group similar texture vectors to create the segmented regions. This thesis uses a hybrid set of image texture features, presented by Stewart et al. [31], to build the sample region's feature vectors and an artificial neural network (ANN) method to assign classification values to each sample region's feature vectors.

The weights between the connections of the ANN (explained in section 3.2) are trained a priori by cross validating hand-picked training data from side-scan images. A priori training of the machine learning technique is referred to as supervised learning, whereas the fuzzy c-means method used in chapter 2 is referred to as unsupervised learning. Figure 3.1 (reprint of figure 1.4) presents an example of side-sonar image classification based on image texture, where the RGB colour values represent the probability that a pixel belongs to a particular

class (the regions are segmented into rocky outcroppings (*red*), sandy regions (*green*), and shadow/no-signal regions (*blue*)).

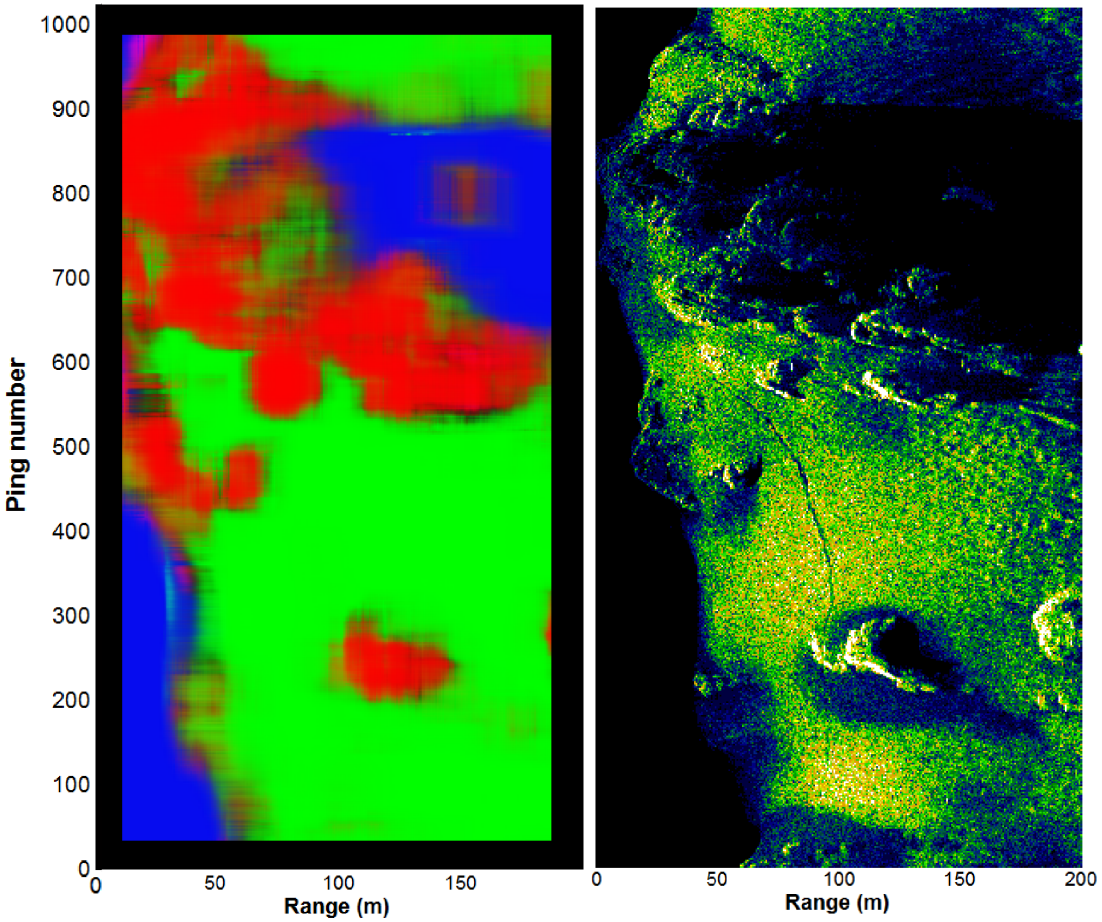

Figure 3.1: Classified side-scan image example from 1.2. The regions are segmented into rocky outcroppings (*red*), sandy regions (*green*), and shadow/no-signal regions (*blue*).

A major drawback in image classification is the large computation time required to build the feature vectors for every sample region to cover the full image. Because the feature vector is constructed for each individual pixel based on processing the sample region over neighbouring pixels, the entire classification process is at least $O(N^2M^2)$ complex (where N is the height of a square sample region and M is the height of a square side-scan image). Depending on the image texture method used, additional hidden complexity may also exist for building a particular feature space. For example, the grey-level run-length (GLRL) image texture method builds a histogram by traversing the sample region and then traverses the entire histogram to build the feature space. The hidden cost of the histogram processing makes building the feature vectors with GLRL for the entire image $O(N^2M^2H^2)$ complex (where H is the height of a square 2-D histogram).

The complexity of image classification makes the processing of a single side-scan image a very time consuming application and it limits the usability of the method. For example, the author's initial investigation of [31]'s proposed hybrid feature spaces took 12 hours to process a single side-scan image using readily available Matlab libraries [50, 51, 52]. This computation time is a very limiting factor even for post-processing after a sonar survey, where dozens of side-scan images are likely collected in a single survey. By porting [50, 51, 52] to C++, a performance improvement of two orders-of-magnitude was achieved. Although processing time of ten minutes per side-scan image might be acceptable for post-processing, it is still several orders of magnitude away from being usable as a real-time process during a sonar survey.

This thesis presents the contribution of creating a GPGPU implementation to compute the image texture feature space in an efficient manner to provide real-time classification information about a side-scan sonar image. To the best of the author's knowledge, there is no publicly available GPGPU library to compute these texture feature spaces, and readily available libraries are not efficient enough to provide real-time information. Although the main contribution for this chapter is a technological solution of an existing method, the image classification method is presented below to allow interested readers to repeat the results from this thesis without having to reference related work.

3.1. Image Texture Feature Space

The image processing method used to produce the results in figure 1.4 is a hybrid image texture feature space, consisting of several well-established image texture processing techniques. There are two hybrid feature spaces presented by Stewart et al., referred to as Hybrid R and Hybrid S [31]. The Hybrid R method emphasizes grey-level run-length features with a minority of grey-level difference matrix features. The Hybrid S method emphasizes spatial grey-level dependence matrix features and a minority of grey-level difference matrix features.

Although newer and more recent methods exist, the literature has demonstrated that for performance, accuracy and parallelize-ability, these traditional methods presented below are still competitive solutions. Also, the image texture classification literature tend to present algorithms as a general solution for all image texture applications, so efficacy of each method applied specifically to side-scan sonar images is unknown. Alternatively, [31]'s hybrid feature space was demonstrated to be $> 85\%$ accurate at classifying side-scan sonar images using a side-scan sonar system of a similar frequency used by URL. In addition, Stewart et al. tested their system with a survey of an area that is geographically similar to the URL's Pam Rocks data set in the Pacific Northwest (they surveyed the Juan de Fuca Ridge). The authors demonstrate that both of their hybrid image texture feature spaces outperform the standard methods from which the hybrid methods were derived.

Although the current literature contains a large number of image texture algorithms, determining the optimal method for side-scan sonar image classification is beyond the scope of this thesis. For the reasons stated above, [31]'s hybrid methods were chosen as reasonably accurate algorithms to investigate as candidates for a GPGPU implementation to achieve real-time side-scan sonar classification.

3.1.1. Grey-Level Run-Length Features

The GLRL feature space is derived from a specialized histogram that accumulates the number of consecutive pixels of the same grey-level for each run-length and each grey-level in a sample region of a monochrome image. The histogram is represented by a matrix $\mathbf{P}$, where matrix element $p(i, j)$ corresponds to number of runs in the image at grey-level i for a run-length of j. To reduce the variation caused by image rotations, classification methods generally build a separate $\mathbf{P_\theta}$ matrix which separately computes grey-level runs in the four major directions: $\theta = 0°, 45°, 90°, 135°$.

Figure 3.2 presents an example sample region of an image with arrows pointing in the direction that the process travels along to build each corresponding $\mathbf{P_\theta}$. One obvious problem with this approach is the diagonal arrows are biased against long run-lengths because only the arrow along the matrix diagonals correspond to the longest possible run-length. Although this can be problematic in the contrived example presented in figure 3.2, in practice the image size is usually much larger than the longest grey-level run present in the image and the impact of this bias quickly disappears.

The GLRL histogram is built from figure 3.2 by travelling along the arrow directions and accumulating information at every change in pixel grey-level. At the beginning of each arrow, the pixel's grey-level is stored and the method traverses the image pixel-by-pixel in the direction of the arrow. The method checks for changes in grey-level during the traversal and increments the element $p_{i,j}$ in $\mathbf{P_\theta}$ corresponding to the stored grey-level and the number of pixels traversed before the grey-level change (run-length). Once a change in grey-level is handled, the new grey-level value is stored and the process is repeated until the entire sample region has been traversed. Figure 3.3 presents the resulting grey-level run-length histogram created by processing the sample region in the four directions presented in figure 3.2.

The GLRL feature space of a sample region is created by processing $\mathbf{P_\theta}$ matrix with different emphasis values to create individual feature values. Each feature in the space processes the histogram in a different way to provide a numerical representation of the image. For example, if there are many short run-lengths in the image, the *short run-length*

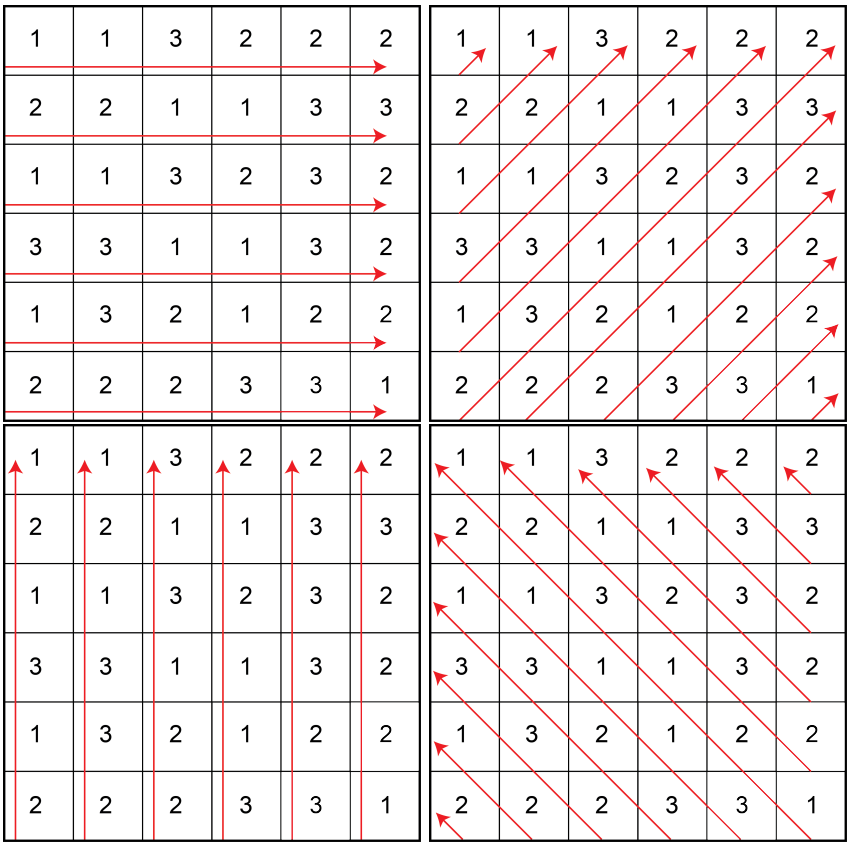

Figure 3.2: Examples of traversing an image to build the GLRL histogram, P_θ, for each direction ($0°, 45°, 90°, 135°$ respectively).

emphasis feature will contain a high score[1].

The initial GLRL method presented in [33] creates the following feature set:

Grey-level distribution:

$$GLD = \frac{1}{n_r} \sum_{i=1}^{M} \left(\sum_{j=1}^{N} p(i,j) \right)^2 .$$

Run-length distribution:

$$RLD = \frac{1}{n_r} \sum_{j=1}^{N} \left(\sum_{i=1}^{M} p(i,j) \right)^2 .$$

Run percentage:

[1] Contrarily, the *long run-length emphasis* feature will contain a small score. This implies many feature values are highly correlated to another within the feature space and only a subset of feature values are needed to accurately approximate the whole space.

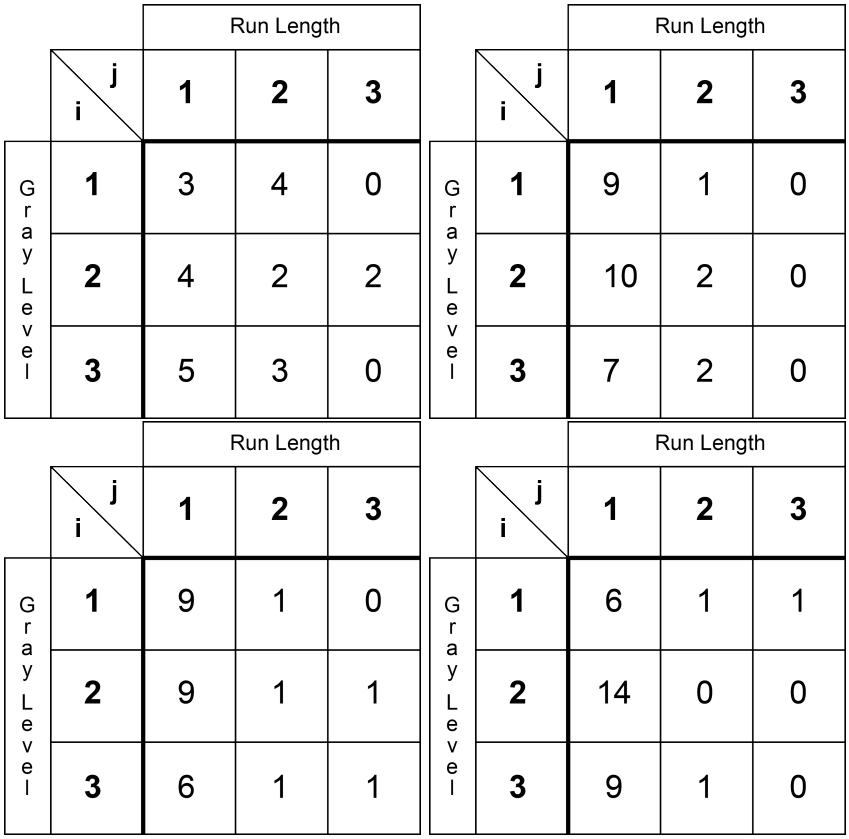

Figure 3.3: Examples of the GLRL histograms created for each direction from the examples in figure 3.2

$$RPC = \frac{n_r}{n_p}.$$

Short run-length emphasis:

$$SRE = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{p(i,j)}{j^2}.$$

Long run-length emphasis:

$$LRE = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} p(i,j) j^2.$$

In all the above cases, n_r is the total number of runs and n_p is the total number of pixels in the sample window.

Chu et al. present two additional features based on grey-level emphasis instead of run-length in the previous two features [53].

Low grey-level run emphasis:

$$LGRE = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{p(i,j)}{i^2}$$

High grey-level run emphasis:

$$HGRE = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} p(i,j) i^2$$

Darasathy et al. complete the modern grey-level run-length feature space by cross-emphasizing grey-level and run-length features [34]:

Short run-length low grey-level emphasis:

$$SRLGE = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{p(i,j)}{i^2 j^2}$$

Short run-length high grey-level emphasis:

$$SRHGE = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{p(i,j) i^2}{j^2}$$

Long run-length low grey-level emphasis:

$$LRLGE = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} \frac{p(i,j) j^2}{i^2}$$

Long run-length high grey-level emphasis:

$$LRHGE = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} p(i,j) i^2 j^2$$

All GLRL features except the first three (GLD, RLD, RPC) can be described by the general run-length grey-level feature equation:

$$GRLGLF = \frac{1}{n_r} \sum_{i=1}^{M} \sum_{j=1}^{N} p(i,j) i^{k_i} j^{k_j}$$

where $k_i, k_j = -2, 0, 2$ according to the emphasis placed by the particular feature. For the features where i or j are numerators, they emphasize large grey-level or run-length values and when they are denominators, they emphasize small grey-level or run-length values, respectively.

As a side note, there is a discrepancy in the [31] appendix that describes LRLGE and LRHGE. The features are listed as $LRLGE = GRLGLF|_{k_i=2,k_j=2}$ and $LRHGE = GRLGLF|_{k_i=-2,k_j=2}$ which will emphasize high grey-levels for $LRLGE$ and low grey-levels

for $LRHGE$. To the k_i values should be switched for LRLGE and LRHGE in [31]. In fact, the [Grey] Level Run Length Matrix Toolbox [50] used to initially test the classification method confirms this errata.

3.1.2. Grey-Level Difference Features

The GLDR method builds a first-order statistical feature space of a sample region. The feature space is built upon a probability mass function (pmf) that describes the likelihood that two pixels in the sample region, separated by distance r in a direction θ, differ in magnitude by a particular grey-level value, i [28, 31]. The method begins by creating an initial matrix, $\mathbf{F}$, which specifies the magnitude of grey-level difference between a pixel at position (x,y) and a pixel at some distance r in direction θ. If a grey-level value of a pixel in the sample region is represented by the function $I(x,y)$, then $\mathbf{F}$ is constructed as follows:

$$\mathbf{F}_{(r,\theta)}(x,y) = |I(x,y) - I(x + rcos(\theta), y + rsin(\theta))|. \tag{3.1}$$

Matrix $\mathbf{F}$ is constructed by updating the (x,y) coordinates in (3.1) while traversing along each pixel in the sample region (similar to figure 3.2). Pixel pairs that fall outside of the sample region are simply ignored. The pmf, $\mathbf{p}$, is then built from F such that $\mathbf{p}(i) = P(F(x,y) = i)$ for every grey-level difference magnitude value, i. The pmf is used to describe the image texture because highly homogeneous images will contain a pmf with an expected value around $i = 0$, while a highly varying images will create a pmf with a concentration around larger i values. By processing $\mathbf{p}$ with a set of statistical descriptors, the GLDR feature set is created with the following features:

Expected value:

$$\mu = \sum_{i=1}^{G} iP(i)$$

Variance:

$$\sigma^2 = \sum_{i=1}^{G} i^2 P(i) - \mu^2$$

Contrast:

$$CON = \sum_{i=1}^{G} i^2 P(i)$$

Angular second moment:

$$ASM = \sum_{i=1}^{G} P(i)^2$$

Entropy:

$$ENT = -\sum_{i=1}^{G} P(i)\log(P(i))$$

Like the GLRL method, to account for rotation, a pmf is constructed for the four major directions $\theta = 0°, 45°, 90°, 135°$. The examples presented in this thesis uses a radius value of 10 pixels (with $(x + rcos(\theta), y + rsin(\theta))$ coordinates rounded to the nearest integer for $\theta = 45°, 135°$). Although using multiple radius values will likely add discriminatory power to the method, it comes with a major cost to performance.

It should be noted that [31] presents two additional features – cluster shade and cluster prominence – that are not presented [28], or mentioned by the [31] as novel. These two features are also never followed up with in subsequent publications and for these reasons they are not discussed in this thesis.

3.1.3. Spatial Grey-Level Dependence Matrix Features

The SGLDM method builds a second-order statical feature space similar to the GLDR method [27, 28, 32, 31]. Like GLDR, the SGLDM method builds a matrix by comparing the grey-levels of pixels separated by a distance r in a direction θ. However, unlike GLDR, the SGLDM method doesn't store grey-level differences, but rather a histogram with the indices specified by the grey-levels of the two separated pixels. To describe the feature space, a grey-level co-occurence matrix, M, is created by traversing an image and incrementing element $m_{i,j}$ for each pixel, where i is the grey-level of a pixel at (x, y) and j is the grey-level of a pixel at $(x + rcos(\theta), y + rsin(\theta))$.

Figures 3.4 and 3.5 provide a visual description of this method. In these examples, the grey-level co-occurrence matrices are constructed by comparing the grey-level of pixel pairs at a radius of $r = 1$ apart, in the directions $\theta = 0°, 45°, 90°, 135°$. For each (i, j) grey-level pair, the corresponding grey-level co-occurrence matrix element $m_{i,j}$ is incremented. Figure 3.5 presents the complete grey-level co-occurrence matrices that were created by traversing the sample regions from figure 3.4

To visualize the histogram-building process, a *red* ellipse is used to collect set of a particular grey-level pairs in the sample regions in figure 3.4. A *red* circle indicates the corresponding m_i, j value of the M_θ matrices in figure 3.5.

With a completed grey-level co-occurence matrix, M, the SGLDM method creates a seven dimensional vector consisting of the following features [27, 32]:

Expected value:

$$\mu = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} iM(i,j).$$

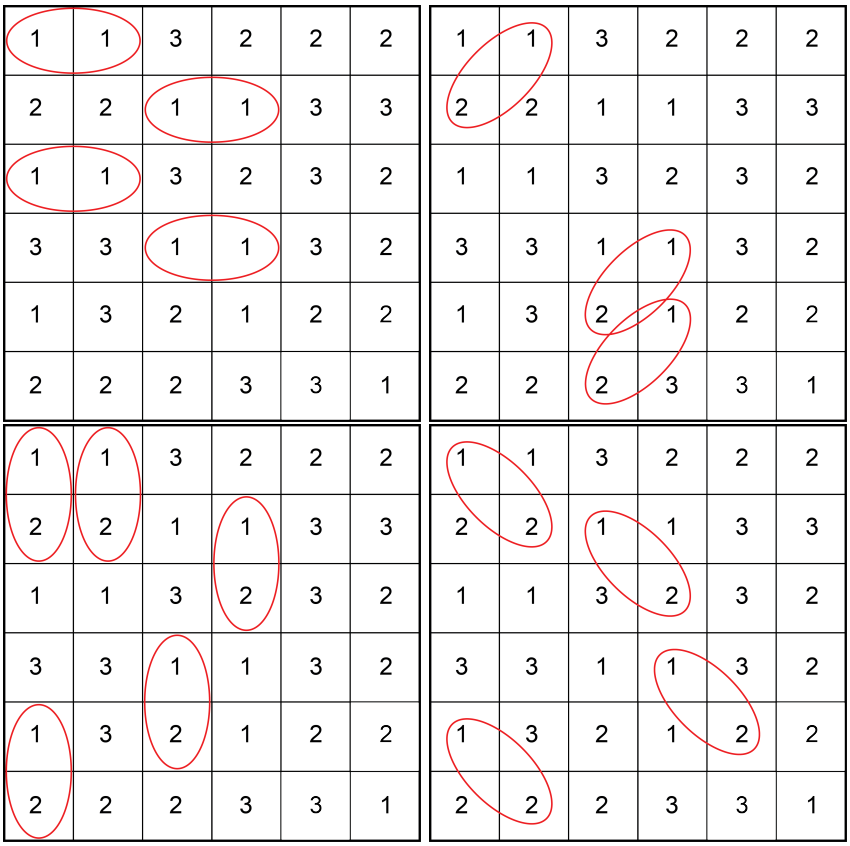

Figure 3.4: Examples of traversing an image to build the SGLDM histogram M_θ for each direction ($0°, 45°, 90°, 135°$ respectively) with a radius $r = 1$. The *red* ellipses represent an particular i,j grey-level pair that is used to build the histogram in figure 3.5.

Variance:

$$\sigma^2 = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N} i^2 M(i,j) - \mu^2.$$

Correlation:

$$C = \frac{1}{\sigma^2 N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}(i-\mu)(j-\mu)M(i,j).$$

Energy:

$$E = \frac{1}{N^4}\sum_{i=1}^{N}\sum_{j=1}^{N} M(i,j)^2.$$

Entropy:

$$H = \frac{1}{N^2} - \sum_{i=1}^{N}\sum_{j=1}^{N} M(i,j)\log(\frac{M(i,j)}{N^2}).$$

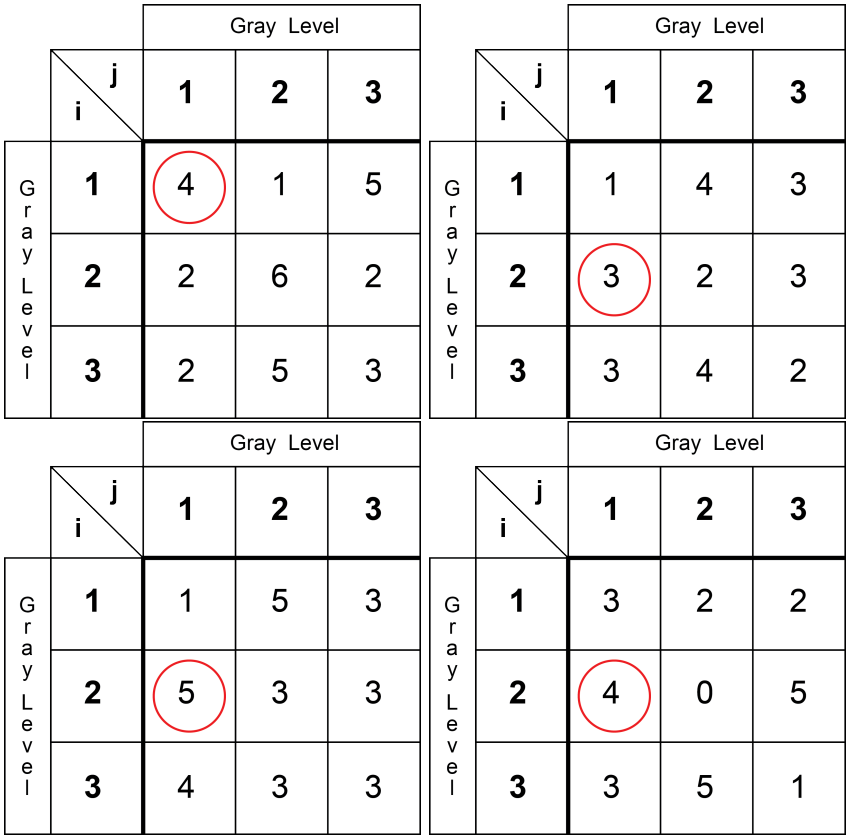

Figure 3.5: Examples of the SGLDM histograms, or grey-level co-occurrence matrices, created for each direction from the examples in figure 3.2. The *red* circle corresponds to the number of i,j grey-level pairs from figure 3.2.

Contrast:

$$T = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}(i-j)^2 M(i,j).$$

Homogeneity:

$$I = \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{M(i,j)}{(1+(i-j)^2)}.$$

As with the previous feature spaces, a separate M_θ matrix is created for the four directions $\theta = 0°, 45°, 90°, 135°$. In this thesis, a radius of $r = 1$ is used to build the feature space. Although this only describes very local information, it is computationally expensive to process multiple radii feature sets.

3.1.4. Hybrid Feature Spaces

Stewart et al. present two hybrid feature spaces, referred to a *Hybrid R* and *Hybrid S*, in an attempt to combine the benefits of the different feature vectors presented above [31]. Because many feature values within the GLRL feature space are highly correlated

to another, selecting a subset of several feature spaces could likely yield better accuracy than any individual feature space mentioned above. Additionally, the authors assert that the GLRL method is best used for discriminating textures with linear components, where as GLDR is best used for discriminating uniformity and density. Since different regions in side-scan images contain different texture features, using a single feature space is not ideal. Blending useful features from a mosaic of methods creates a more robust image texture feature space.

The Hybrid R method creates a feature space that heavily emphasizes GLRL features with a minority of GLDR features. Alternatively, the Hybrid S method heavily emphasizes SGLDM features, also with a minority of GLDR features. To account for image rotation, the four directions ($\theta = 0°, 45°, 90°, 135°$) are averaged for each GLRL, GLDR and SGLDM feature. In addition, the variance of the four directions is also calculated to account for any axis-aligned features.

Both hybrid feature spaces are designed as follows:

Hybrid R:

1. Grey-level mean of image
2. Grey-level standard deviation of image
3. Four-direction mean of GLRL LRE
4. Four-direction variance of GLRL LRE
5. Four-direction mean of GLRL HGRE
6. Four-direction variance of GLRL HGRE
7. Four-direction mean of GLRL LRLGE
8. Four-direction variance of GLRL LRLGE
9. Four-direction mean of GLRL LRHGE
10. Four-direction variance of GLRL LRHGE
11. Four-direction mean of GLRL RLD
12. Four-direction variance of GLRL RLD
13. Four-direction mean of GLRL GLD
14. Four-direction variance of GLRL GLD

15. Four-direction mean of GLRL RPC
16. Four-direction variance of GLRL RPC
17. Four-direction mean of GLDR expected value
18. Four-direction variance of GLDR expected value
19. Four-direction mean of GLDR entropy
20. Four-direction variance of GLDR entropy

Hybrid S:

1. Grey-level mean of image
2. Grey-level standard deviation of image
3. Four-direction mean of SGLDM correlation
4. Four-direction variance of SGLDM correlation
5. Four-direction mean of SGLDM energy
6. Four-direction variance SGLDM energy
7. Four-direction mean of SGLDM entropy
8. Four-direction variance of SGLDM entropy
9. Four-direction mean of SGLDM contrast
10. Four-direction variance of SGLDM contrast
11. Four-direction mean of SGLDM homogeneity
12. Four-direction variance of SGLDM homogeneity
13. Four-direction mean of SGLDM expected value
14. Four-direction variance of SGLDM expected value
15. Four-direction mean of SGLDM variance
16. Four-direction variance of SGLDM variance
17. Four-direction mean of GLRL RPC
18. Four-direction variance of GLRL RPC

19. Four-direction mean of GLDR mean

20. Four-direction variance of GLDR mean

21. Four-direction mean of GLDR entropy

22. Four-direction variance of GLDR entropy

The cross-validation error, presented in subsection 4.2.5, indicates that the Hybrid R feature space is the more accurate solution for classifying side-scan sonar images. For this reason, the Hybrid R was chosen as the preferred feature space technique to implement into a GPGPU method.

3.2. Artificial Neural Network Classifier

Once an image texture feature vector is computed for a sample image, it needs to be processed by a machine learning method in order to assign a particular class to the sample region [31]. In this thesis, an artificial neural network (ANN) system is used as the machine learning method to classify the sample regions of a side-scan sonar image. This method was chosen because it proved to be a very accurate and, while the training process is slow, the classification process is fast and easily parallelizable[2].

An ANN is a system that is built on layers of neurons and weighted connections between neurons of consecutive layers [54]. Figure 3.6 provides a visual explanation, where the input data (from the feature vector) is passed through to the neurons in the *input layer*. The neurons in the *input layer* pass on their information to the neurons in the *hidden layer* through the connections between layers[3]. Each connection between neurons contains a weight that controls the amount of influence one neuron has on the next consecutive neuron.

Figure 3.7 presents an example of a single connection between two neurons, i and j, in their corresponding layers, $k-1$ and k respectively. The output, or activation, of neuron j is represented by a_j^k and it is influenced by the activation of neuron i in the previous layer, represented by a_i^{k-1}. The influence of activation a_i^{k-1} on a_j^k is weighted by w_{ij}^k. Although the connection between neurons in figure 3.7 appear rather straight forward, the full network example in figure 3.6 show that the activation a_j^k is influenced by all the neurons in the previous layer.

Figure 3.7 only examines one connection between neurons of different layers. Because a neuron is connected to all the neurons of the previous layer, the activation of that neuron is derived from the weighted sum of all activation values of neurons from the previous layer. To approximate the true activation of a biological neuron, the weighted sum of activation

[2]However, parallelizing the ANN is left as future work.

[3]Although figure 3.6 only demonstrates a single *hidden layer*, an ANN design can easily extend to multiple *hidden layers*.

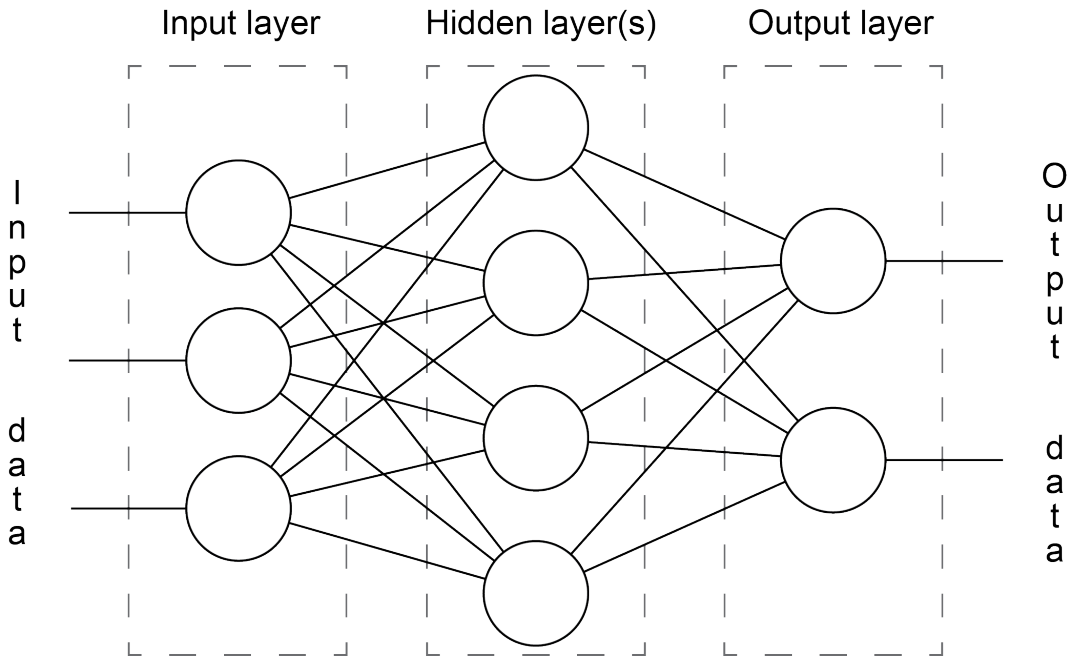

Figure 3.6: Example of an artificial neural network. The input data is passed through the layers the system by weighted connections. The output data is a function of the input data and the weighted connections between neurons of each layer.

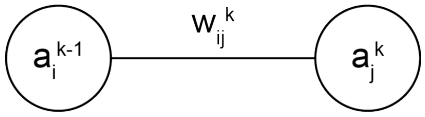

Figure 3.7: Example of a single connection between two neurons. The activation, a_j^k, of neuron j in layer k is influenced by the activation, a_i^{k-1}, of neuron i in layer $k-1$. The influence of this connection is weighted by w_{ij}^k.

levels from the previous layer is passed into a sigmoid function. Equations 3.2 and 3.3 demonstrate how the activation, a_j^k, of a neuron is determined based on neurons from the previous layer.

$$a_j^k = \sigma(\sum_i w_{ij}^k a_i^{k-1}) \tag{3.2}$$

$$\sigma(z) = \frac{1}{1+e^{-z}} \tag{3.3}$$

Computing output data (classification values in this case) by propagating input values through the sigmoid functions and weighted connections is quite fast and simple. However, the process of determining the optimal weights for the connections is more complicated. Like most optimization methods in machine learning, determining the optimal values for the degrees of freedom (e.g. the connections weights) is achieved by minimizing some cost

function. For an ANN, this usually implies minimizing some variation of the following:

$$C = \frac{1}{2} \sum_{n} \sum_{j} \| y_{n,j} - a_{n,j}^{output} \|^2, \tag{3.4}$$

where $y_{n,j}$ is a known output training value for test case n and for output dimension j, and $a_{n,j}^{output}$ is the activation at the output from the ANN based on $y_{n,j}$'s corresponding input training values. If the network output activation matches the training data, the cost function will be zero and the network will be in an optimal state. Unfortunately, finding the ideal parameters for the connection weights is challenging because $a_{n,j}^{output}$ is a non-linear function (due to the sigmoid function in (3.3). The non-linearity means that solving $\frac{\partial C}{\partial w_i j^k} = 0$ to find the optimal connection weights is a non-trivial problem.

A method called backwards propagation of errors, or backpropagation, was introduced to approximate the partial derivatives of the cost function within the network [55]. This technique initially assigns random values to the connection weights of the ANN and computes output activation values from training data. The error between the ANN output and the training data is passed backwards into the network from the outputs and the back-propagation error is used to approximate the partial derivatives of the cost function at each connection weight. Using this method, the weight values are usually updated using an optimization method like a gradient descent approach. This process is repeated until the system converges onto a local minimum.

To alleviate the problems of encountering local minima, the entire process of back-propagation is repeated many times, each with a new set of randomized initial connection weights. A cross-validation scheme is used to determine the set of connection weights that achieves the lowest activation output error with respect to the training data. Once the ANN is constructed, new image texture feature vectors are be passed into the system and a classification value is represented by activation output for each *output layer* neuron. As an example, a *sandy* sample region of a side-scan sonar image should yield an activation output close to 1.0 for the *sandy* output neuron and activation outputs close to 0.0 for all other output neurons in the network. In fact, the colour coding of the side-scan image in figure 1.4 is taken directly from the output values of the ANN and converted into RGB values as a visual representation.

In general, because many ANN libraries exist (e.g. MATLAB's Neural Network Toolbox), training the connection weights with backpropagation can be treated as a black box method. A user only has to choose the network architecture (number of hidden layers and neurons in the each layer) and provide the training data inputs and outputs.

3.3. GPGPU Conversion

The GLRL, GLDR and SGLDM feature spaces presented in section 3.1 can all be trivially computed using the [50, 51, 52] libraries. Even the two hybrid feature spaces can be easily calculated be some additional processing on the output of the libraries. These libraries are extremely valuable because they simplify testing and verifying the accuracy of these features spaces with any arbitrary data set. However, the computation time required to build the feature vectors as inputs to a classifier is too substantial to be useful as a real-time classification system.

To solve this problem, a hardware solution is used to substantially reduce the computation time to build the feature vectors. The Hybrid R feature space from subsection 3.1.4 was implemented in a GPGPU programming language called OpenCL [56]. This *C*-based programming language allows a developer to exploit the massively parallel hardware architecture of a GPU to execute computations not related to graphics rendering. The main challenge when converting a CPU-based algorithm into a GPU-based is designing with their hardware differences in mind.

GPU hardware is designed with a SIMD (single instruction, multiple data) architecture, which means a single instruction is executed on a many computation threads. This is a historical design that was created to rasterize and render a large number of triangles at once; a process that is identical for every triangle in the batch. Unlike multi-threading on a CPU, the SIMD architecture is not well equipped to handle branch decisions for particular threads and they can easily become a bottleneck.

With OpenCL, single programs, or *kernels*, are executed one at a time on a GPU. Like a CPU, a GPU contains many cores however, with a GPU, the same kernel is executed on every core. Each core in the GPU contains a group of threads, referred to as a *work group*, and each thread is referred to as a *work item*. Each work item in a work group executes the same line of code at the same time, hence the *single instruction* part of SIMD [57]. The *multiple data* portion of SIMD implies that each work item may contain different local variables.

The OpenCL implementation of the Hybrid R feature space begins by loading an entire image into global memory on the GPU. A kernel is then passed into the GPU to process the loaded image. A different work item created in the kernel for every accessible pixel in the image. Each of these work items is initialized to correspond to the center of its own unique sample region in the loaded image. Although the starting position for each work item is different, the *single instruction* aspect dictates that the process of looping over a sample region must be the same.

For the GLRL features, each work item also contains local grey-level and run-length values. The kernel instructs all the work items to traverse their sample region, and check

if the grey-level changed and update their local histogram if so. The process of checking for grey-levels and updating the histogram introduce small branch conditions because some work items will experience a change in grey-level and others will not. Branching in a SIMD system is allowed, but it forces the kernel to handle the work items that do branch separately form the work items that do not branch. If many nested or time-consuming branches occur frequently, the savings from parallelization quickly disappear. The use of ternary operators can help reduce the use conditional branches, but in practice there was no decrease in performance for small, localized branch conditions.

A GPU can typically process tens of thousands of work items at once; many more work items than necessary to compute pixels from a single ping (and neighbouring pixels to fill the sample regions). To maximize the vacant work items, the GLRL and GLDR kernels compute the four direction ($\theta = 0°, 45°, 90°, 135°0$) feature values at once. Although the method for traversing a sample region is different for each direction, if a work group in a kernel only contains work items that are traversing in a certain direction, there will not be any complications. As mentioned above, each work item in a work group should ideally execute the same operations, but they don't have to be consistent between work groups.

With the histogram (or pmf) constructed for each work item in the GLRL (or GLDR) kernel, computing the actual feature values for each work item is a simple implementation with no branch conditions or directions to manage. The feature values for each direction is stored in memory on the GPU, and a second kernel is executed that accesses the stored values and computes the various four-direction mean and variance values for each work item to create the Hybrid R feature space. The final feature values are passed back to the CPU and used as inputs to the ANN classifier.

3.4. Sea-floor Classification Conclusion

Based on Stewart et al.'s original method, this chapter describes the various pieces necessary to build a real-time sea-floor classification method from side-scan sonar images. Traditional image texture methods are generally less accurate than the hybrid methods because they tend to discriminate only certain types of texture information [31]. In addition, many feature values in the GLRL feature space are redundant and highly correlate with another feature value in the space. The most accurate method from [31], the Hybrid R feature space, was chosen as the desired method to convert to OpenCL.

Implementing the GLRL and GLDR methods in OpenCL was the most challenging section in this chapter because the algorithms must be re-designed in a SIMD environment. This requires substantial investment to learn both the OpenCL API and the design limitations with GPGPU programming. However, the performance benefits of such a massively parallel system make re-implementing a fruitful endeavour. Finally, OpenCL was chosen over CUDA because OpenCL can be executed on a wide range of GPUs and CPUs, whereas

CUDA is only compatible with nVidia GPUs.

The ANN classifier was not ported to the GPU because computing the activation output values for a set of feature vectors is quite simple and efficient. However, given a set of weights, computing activation outputs for an ANN is easily parallelizable because it can be reduced to a series of matrix multiplication processes. The most time consuming and challenging part of an ANN method is the backpropagation portion and that is handled a priori during the training phase [58].

The final result of this chapter is highlighted in figure 3.1, where each the pixel data in each row corresponds to a single ping. A sample window is created for each accessible pixel in the side-scan sonar image and a colour value is assigned based on the classification value output from the ANN classifier. A real-time system should be able to provide this classification data for each image row at the same rate or faster than the sonar system creates acoustic pings.

Although the current system processes classification information for each pixel, complexity savings could be achieved by only processing classification values for a coarse sampling of pixels. If there was a composition change between pixels (based on classification values), the system could compute a refined region between the two different composition values. However, adaptive refinement was not investigated in this thesis, and highly heterogeneous side-scan images will not benefit from the adaptive refinement method (e.g. worst-case complexity).

Alternatively, multiple side-scan image rows could be passed to the GPU to be computed at the same time. By allowing multiple pings to be processed at once, the feature space computation cost would be amortized over several pings. However, as indicated in chapter 4, the current design is sufficient to achieve the computation time restrictions imposed for this thesis.

4. Experimental Analysis

4.1. Sea-floor Profiling Results

4.1.1. Error Analysis

The method presented in chapter 2 demonstrates an effective and efficient algorithm to identify the sea-floor profile within 3-D side-scan sonar bathymetry data. Although the examples in chapter 2 already covered a wide range of complicated bathymetric profile, this subsection analyzes the accuracy of the method (when compared to a human operator) with a larger set of test cases. The test data set is a hand-picked selection of 74 pings from the URL's Pam Rocks data set that consist of bathymetry data with multi-path, water-column objects, nadir region gaps and shadow region gaps. This collection was chosen to contain a variety of complicated bathymetry data to properly assess the over-all accuracy of the sea-floor profiling method.

For each ping in the test data set, an operator identified the signal returns that best corresponding to the sea-floor. These selected signal returns serve as a baseline to compare with, and assess the accuracy of, the sea-floor profiling algorithm. Figure 4.1 presents two example pings with the operator-identified test data superimposed onto the original bathymetry data.

Figure 4.2 presents an example of mismatches between the sea-floor data identified by the operator and by the sea-floor profiling algorithm. The instances where a signal return is identified as sea-floor data by the operator but not by the sea-floor profiling algorithm are referred to as *false negatives*. Alternatively, the instances where a signal return is identified as sea-floor data by the sea-floor profiling algorithm and not by the operator are referred to as *false positives*. Both types of errors are important to minimize because an ideal sea-floor profiling system should avoid adding incorrect data while preserving the true sea-floor data.

Table 4.1 presents false positive and false negative error rates between operator-selected sea-floor data and the sea-floor profiling algorithm data for the 74 test cases. The average error rate is presented on a per-ping basis, which means the sum of the error rate for each ping is divided by total number of pings in the data set. The average error (per-ping) is considerably low and suggests that, on average, the sea-floor profiling algorithm is almost as accurate as a human operator at identifying signal returns corresponding to the sea-floor.

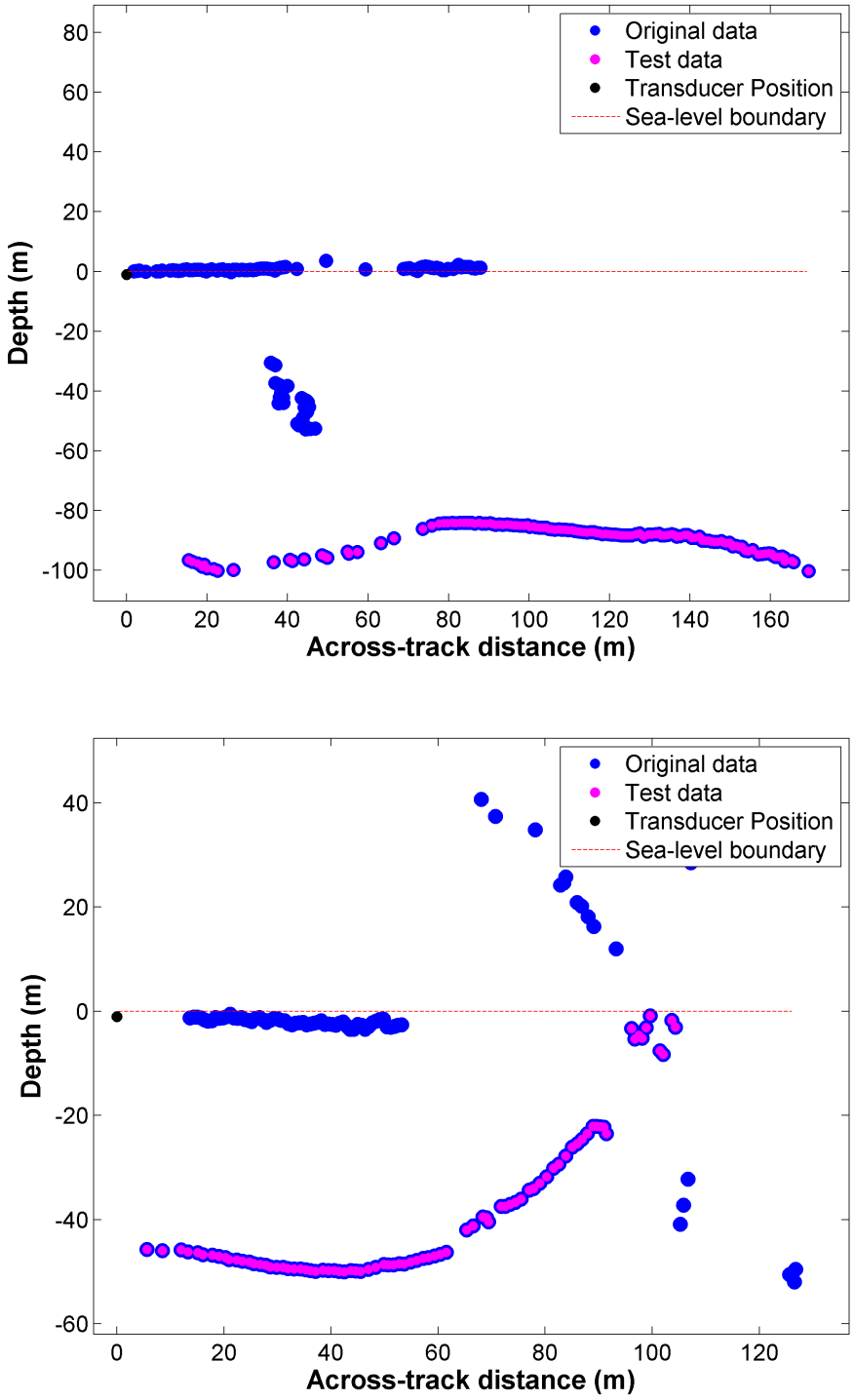

Figure 4.1: Examples of operator-selected sea-floor data points. The original bathymetry data are indicated with blue dots, while the data that was hand-selected as corresponding to the sea-floor are indicated with smaller magenta dots (which overlap the original data).

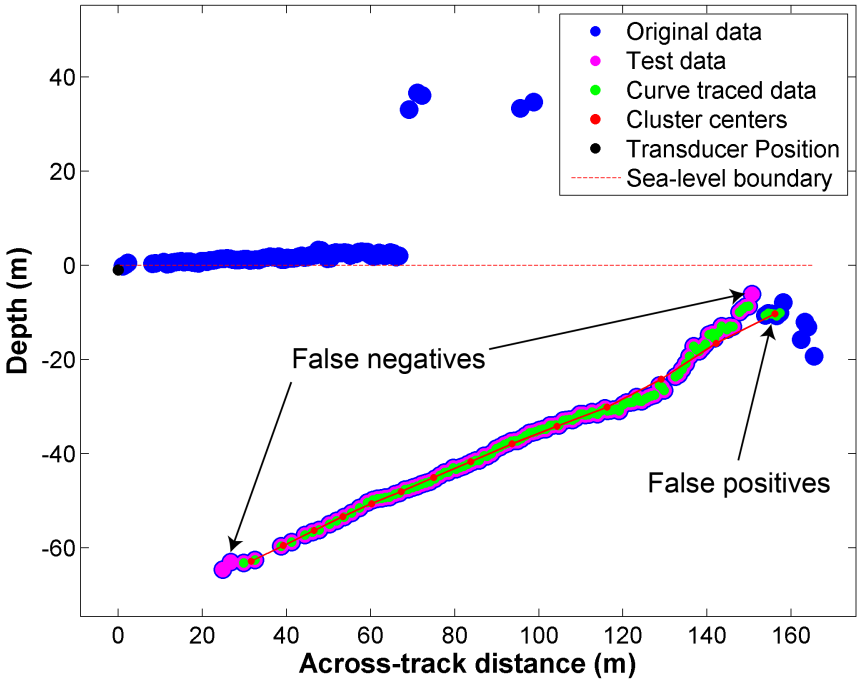

Figure 4.2: Example of errors between sea-floor profiling and operator-selected data points. The original bathymetry data are indicated with blue dots, the operator-identified sea-floor data are indicated with smaller magenta dots, and the sea-floor profiled sea-floor data are indicated with green dots. The incorrectly identified sea-floor data (*false positives*) and incorrectly missed sea-floor data (*false negatives*) are also indicated.

Although the average error is quite low, it is important to investigate the worst-case examples. In addition to average error, table 4.1 presents two worst-case measures for both the false positive and false negative categories. The first case is the ping with the largest error as a percentage of total number of signal returns in the ping and the second being the ping with the largest total number of errors (although a lower percentage of overall error). Although the worst case values in table 4.1 might be cause for concern, a visual inspection demonstrates that the worst cases are not as catastrophic as the numbers suggest.

	False positive rate	False negative rate
Average (per-ping)	2.5%	2.6%
Worst case (by percentage)	47.6%	10.9%
Worst case (by total error)	23.7%	10.9%

Table 4.1: Error measure between operator and sea-floor profiling algorithm

Figure 4.3 presents the pings corresponding to the largest percentage (*top*) and largest total of false positive (*bottom*) error. In both examples, the cluster center chains created by the sea-floor profiling algorithm extend beyond the furthest operator-identified sea-floor

signal return. Although the errors are mismatches between the baseline of operator-selected data and the sea-floor profiling algorithm, the exact location of the sea-floor in both examples is rather ambiguous. In situations like those in figure 4.3, even the operator has to make a judgement call when identifying the signal returns that correspond to the sea-floor. Additional processing of the sea-floor profiling algorithm might be necessary to reduce the false positive error. However, given that the operator is not completely confident about which signal returns correspond to the sea-floor in figure 4.3 and that the sea-floor profiling algorithm still traces the general profile of the sea-floor signal returns, the method presented in chapter 2 can still be considered a reliable solution for identifying sea-floor signal returns.

Figure 4.4 presents the ping that corresponds to both the largest percentage and largest total false negative error. This example demonstrates that the false negative error was cause by a region of sea-floor signal returns incorrectly filtered out by the sea-floor profiling algorithm. This region was removed during the amplitude filtering stage (subsection 2.1.2) because the amplitude filtering parameter, d_a, was set too high. Unlike the examples in figure 4.3, this type of error can be reduced by parameter tuning.

4.1.2. Performance Analysis

The computational complexity of the sea-floor profiling algorithm is polynomial due to the underlying cost-function minimization from the FCM algorithm (equation 2.4). The minimization process iterates over all the data points and all the cluster centers until the cost-function converges onto a local minimum. While this complexity can quickly make the algorithm too expensive to compute in real-time during a sonar survey, the collected data used in the thesis was never large enough to experience this problem. The entire average computation time, per ping, for the sea-floor profiling algorithm is an order of magnitude smaller than the initial completion-time goal of 270 ms (specified in subsection 1.3.1). Regardless, computational bottlenecks are important to address because future applications might encounter larger data sets or more stringent computation time restrictions.

Table 4.2 provides a breakdown of the average computation time for each major function in the sea-floor profiling algorithm. The three pre-curve-tracing functions, *boundary filter*, *amplitude filter* and *angle ray-casting*, are trivially fast (compared to the overall computation time). These three functions are very efficient because they are linear or log linearly complex (due to sorting) with a small computational bandwidth for each item in the data set. Similarly, the *fuzzy curve tracing*, *breaking chain loops* and *isolating sea-floor bathymetry* functions are also extremely fast. The *fuzzy curve tracing* function is $O(n^2)$ complex, however the number of cluster centers is so low that the impact of polynomial growth on computation time is never experienced. The *breaking chain loops* function is a simple linearly complex process that traverses each cluster chain and checks for (and corrects) loops in the chains. Interestingly, the *isolating sea-floor bathymetry* function is the

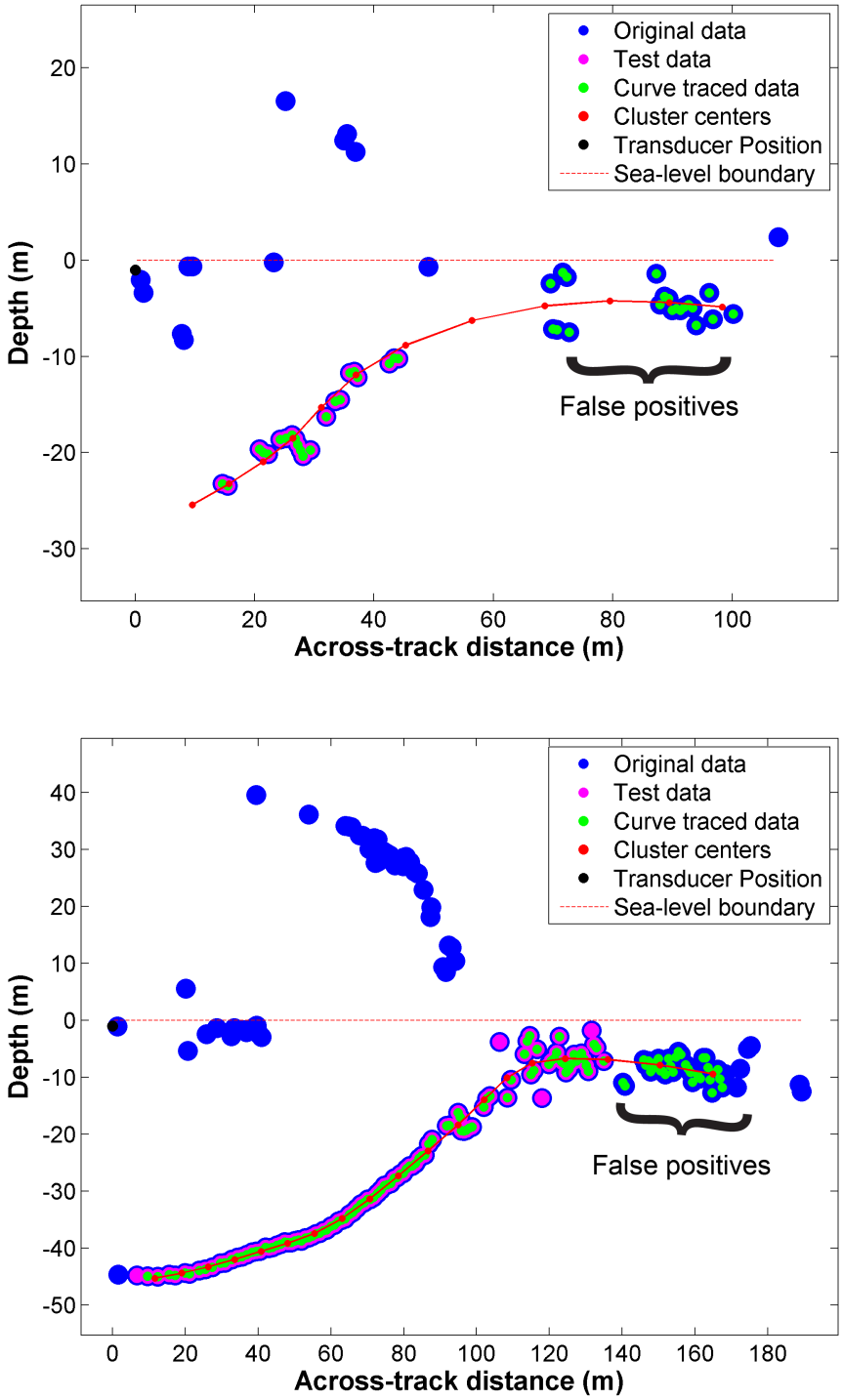

Figure 4.3: Pings with the largest percentage (*top*) and largest gross (*bottom*) false positive error. In both cases, the sea-floor profile created by the chain of cluster centers extends beyond the operator-selected sea-floor data and traces through additional bathymetry data. This continuation accounts for the *47.6%* and *23.7%* false positive error, respectively.

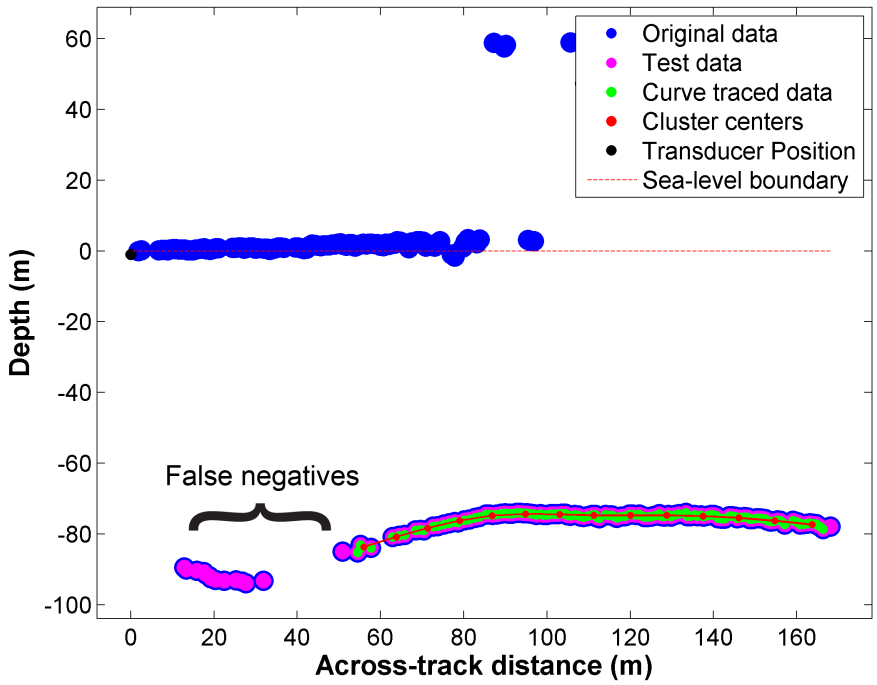

Figure 4.4: Ping with both the largest percentage and largest gross false negative error. The region of the sea-floor was not traced by the chain of cluster centers because it was filtered out of the sea-floor profiling algorithm during the amplitude filter stage.

second fastest function presented in table 4.2. At first glance, such a high performance is rather surprising because each cluster chain computes an error value by looping over all the data points. This looping process is similar to the FCM methods, and one might anticipate the computation time to be similar. However, because *isolating sea-floor bathymetry* mainly consists of efficient (hardcoded in C++) vector projection and squared-distance measures (square root functions are slow to compute), the computational cost is insignificant.

Due to a large computational bandwidth and $O(n^2)$ complexity per iteration, the functions that are derived from the FCM algorithm (*FCM with repulsion*, *Chain curvature constraint*, *Joining chains* and *Casting shadows*) are the most time consuming functions in the sea-floor profiling solution[1]. The initial FCM function, *FCM with repulsion* is the greatest bottleneck because it is initialized with a large number of cluster centers. The number of cluster centers in the converged solution of *FCM with repulsion* is reduced by removing cluster centers that were ejected due to the repulsion penalty term (see subsection 2.3.2). The reduced cluster center count, along with largely converged initial positions for the cluster centers, improves the performance of the other three FCM-based functions.

[1]Recall from chapter 2 that the *Joining chains* and *Casting shadows* functions call the *Chain curvature constraint* function any time two cluster chains are connected

Operation	Average computation time (ms)
Boundary filter	0.0076
Amplitude filter	0.0098
Angle ray-casting	0.0085
FCM with repulsion	4.74
Fuzzy curve tracing	0.0031
Breaking chain loops	0.0046
Chain curvature constraint	1.04
Joining chains	1.08
Casting shadows	0.387
Isolating Sea-floor bathymetry	0.0039
Total	**7.40**

Table 4.2: Average computation time of each major component in the sea-floor profiling algorithm

The average computation times are derived from processing the test data set for subsection 4.1.1 using an Intel i7 3.50 GHz CPU on a single thread. Given the desired goal from section 1.3.1 to process a ping within 270ms or less, the C++ implementation of the sea-floor profiling algorithm can reliably be considered a real-time algorithm. Further optimization of the sea-floor profiling algorithm was not necessary to achieve the goals specified in this thesis, however additional improvements of the FCM components of the algorithm could easily be realized through parallelization or an octree spatial partition.

4.2. Sea-floor Classification Results

4.2.1. Ground Truthing

The accuracy of the side-scan classification method is based on comparing the output of a classifier with the class assignment from a human operator. For example, if a region of a side-scan image is identified by a human operator as having rocky features, the classifier should assign this region a *rocky* label. However, the human operator must be able to confirm that this rocky side-scan region indeed contains rocky features. This confirmation is referred to as ground truthing.

For this thesis, one side-scan image of the URL's Pam Rocks data set was ground truthed with drop-camera footage to verify features identified as rocky and sandy regions within the side-scan image correspond to the actual sea-floor. Figure 4.5 presents a map of the seven drop-camera runs around the Pam Rocks area. The white arrows indicate the direction that the vessel travelled during the video capture (except *Run3* because there was no room). Figure 4.6 presents a zoomed-in map in the northern region of figure 4.5 that also includes the vessel's travel during the sonar survey used to produce figure 1.2 (labelled *Nov23_036*). The highlighted area in figure 4.6 represents the approximate survey area during the *Nov23_036* run.

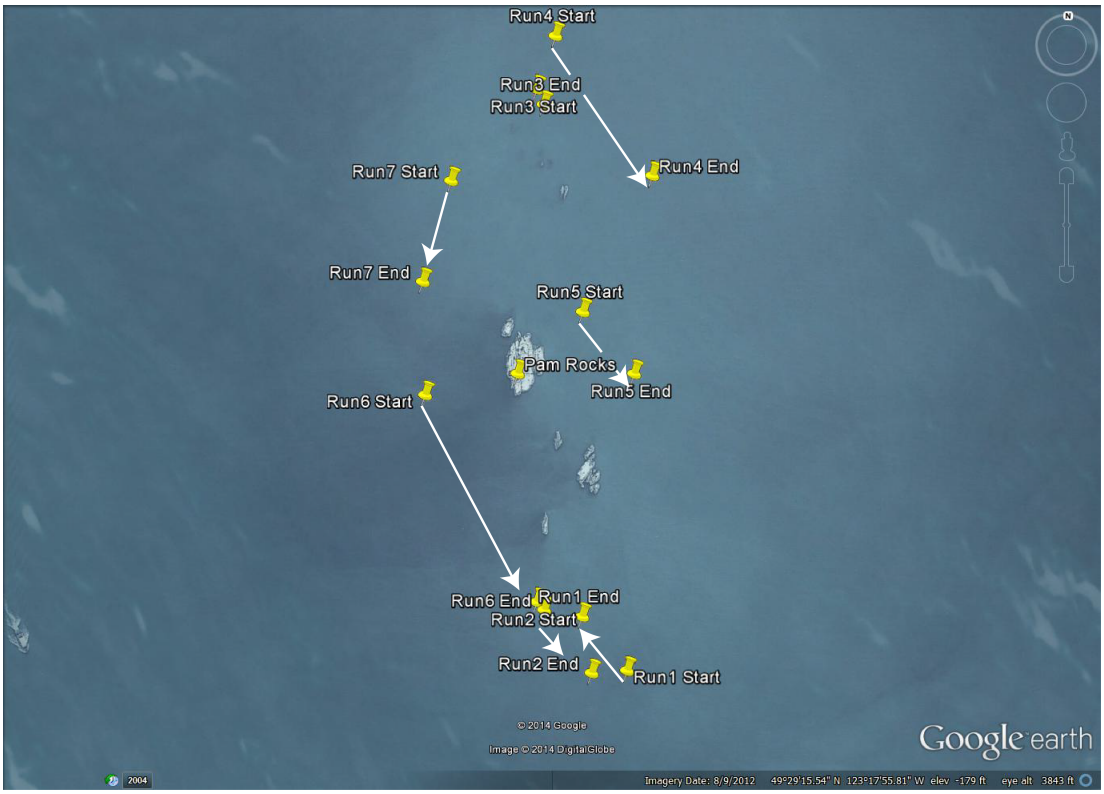

Figure 4.5: Map of drop-camera runs around Pam Rocks

The drop-camera runs *Run3* and *Run4* appear to intersect with the surveyed area from *Nov23_036*, so their footage was investigated to provide ground truth information for the side-scan image of *Nov23_036*. Figure 4.7 is a screen-capture image from *Run4* that indicates this particular region contains a sandy bottom with small rocks. Also, because the image is rather dark, it implies that the sea-floor is deep enough that light from the surface has been attenuated. Alternatively, figure 4.8 is a screen-capture image from *Run3* and it indicates that this region contains jagged rocky features. Additionally, the image is much brighter than figure 4.7, which indicates it is much closer to the surface.

The observations from the drop-camera videos are consistent with the classified 3-D side-scan example from figure 4.9 (reprinted from figure 1.5). The the right-hand side of the *Nov23_036* survey is tens of meters deep and is classified as mostly sandy region. This section of figure 4.9 agrees with the observations in figure 4.7. Additionally, the left-most portions of the survey are classified as shallow rocky regions. This section agrees with the survey area and drop-camera intersection as well as the vessel direction in figure 4.7.

The training data set presented below was hand selected by a human operator based on identifying a subset of side-scan image features from drop-camera observations. That means that ground truthing was not used explicitly to build the entire training data set,

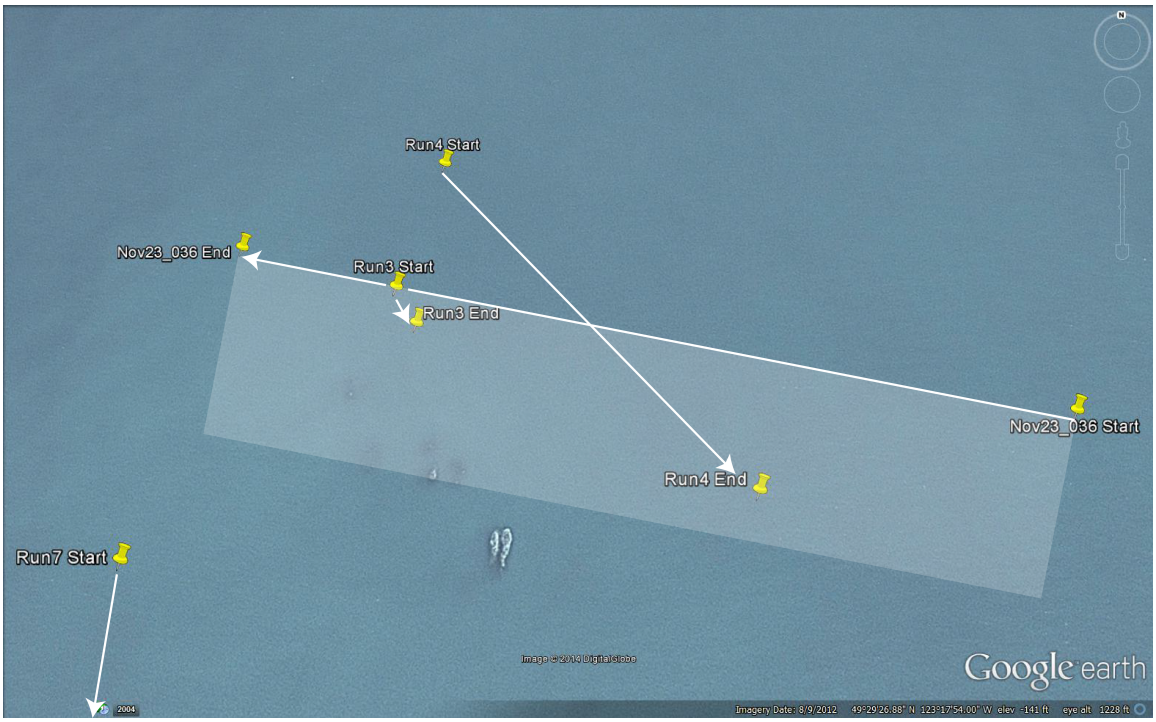

Figure 4.6: Map of drop-camera runs around Pam Rocks with the sonar survey run for *Nov23_036*. The highlighted area indicates the approximate survey area during the sonar run.

Figure 4.7: Drop-camera screen-capture of a sandy region from *Run4*

Figure 4.8: Drop-camera screen-capture of a rocky region from *Run3*

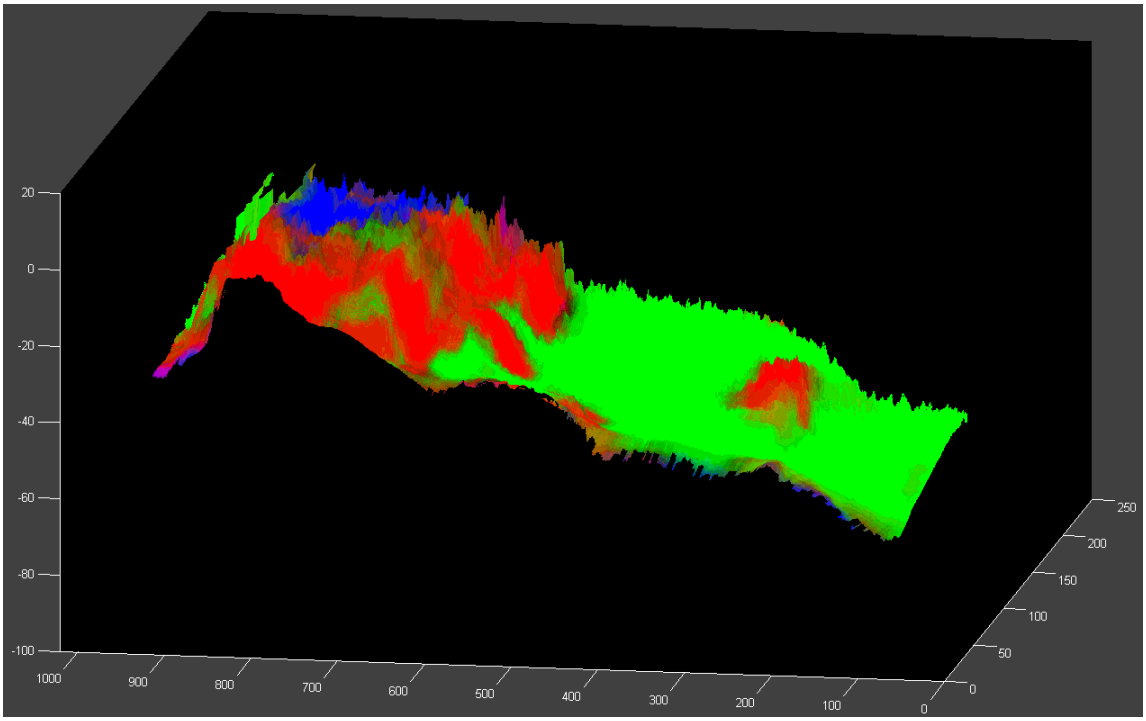

Figure 4.9: Classified 3-D side-scan example with figure 1.4 projected onto bathymetry data from figure 1.3

but it was used to build an initial understanding of how to identify features in the side-scan images from the URL's Pam Rocks data set.

4.2.2. Training Data

The ANN classifier from section 3.2 is trained by image texture feature vectors from hand-picked and classified training image chips. An operator selects clippings of side-scan images to represent corresponding sea-floor features and labels them accordingly. These clippings, or chips, can be of arbitrary size so, so in order to maintain consistency between image chips, a sample region of a constant size is processed around each accessible pixel in the image chip. An accessible pixel refers to a pixel at the center of a sample region, where no part of the sample region lies outside of the image chip. For example, an image chip with dimensions of 100x100 pixels using a 100x100 sample region will only contain one accessible pixel.

Figure 4.10 presents three training image chips (scaled individually from their original size) corresponding to *rocky*, *sandy* and *shadow* features. While the *rocky* image chip mostly contains rocky outcropping features, smooth and sandy regions are visible around the feature. Preventing bleed-through of other feature classes can be a challenging problem when trying to find large enough features fill a sample region.

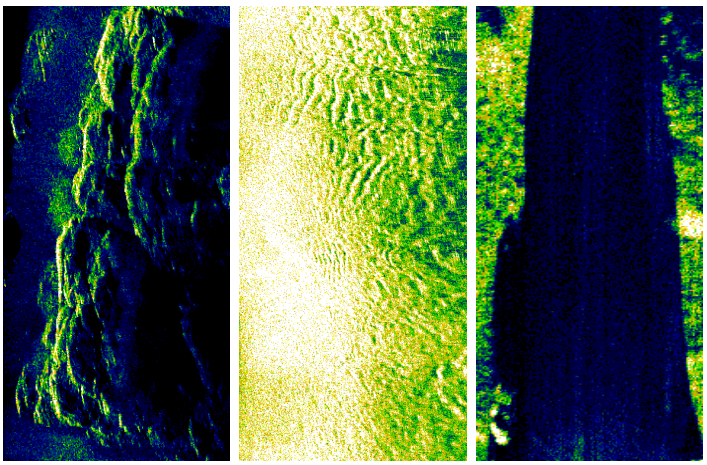

Figure 4.10: Training image chips corresponding to the *rocky* (*left*), *sandy* (*middle*), and *shadow* (*right*) feature classes.

Each sample region created by the image chips is processed to produce an image texture feature vector, and each feature vector is used as the input training data to the ANN. The sample regions are also assigned a classification attribute, which is used as the desired ANN activation output, or output training data. With both input and output training data sets created for the ANN, the backpropagation technique is used to train the classifier. As mentioned in section 3.2, the backpropagation training method is repeated multiple times and the system with the lowest cross-validation error is chosen for use during future sonar

surveys.

4.2.3. Sample Region Size

The size of the sample region formed around an image pixel is an important parameter. If the sample region size is too small, it won't accurately capture enough image texture information to classify the pixel area. However, because computing the feature space is $O(n^2)$ complex, where n is the length of a square sample region, increasing the length will substantially increase the computation cost. Figure 4.11 presents the cross-validation error of each image texture class for a range of sample region sizes using the Hybrid R feature space and an artificial neural network with two hidden layers (20 input neurons, two hidden layers of 20 neurons and an output layer of 3 neurons – or a 20-20-20-3 ANN). For each sample region size, the backpropagation technique was executed 100 times and the 50 most accurate results were used to produce the average cross-validation error.

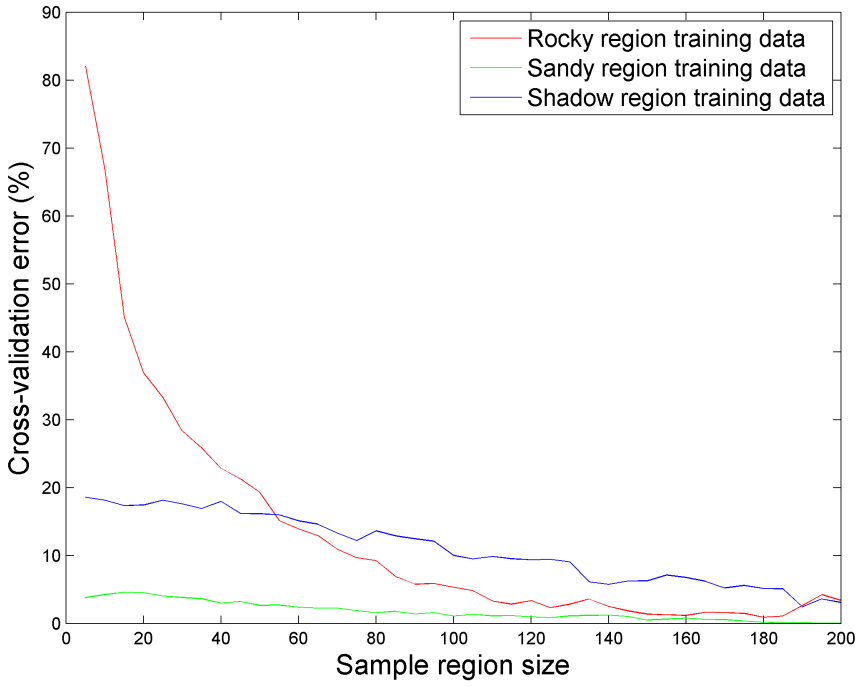

Figure 4.11: Cross-validation error as a function of sample region size. The error is

As expected, the error for each class reduces as the sample region size increases. Interestingly, the shape of the sample region size vs. cross-validation error for the *rocky* class in figure 4.11 is dramatically different than the other two classes. The large initial error is due to the fact that *rocky* regions contain long linear components, which requires a large enough sample region to capture these features. In other words, the GLRL features need long enough run-lengths in several directions to provide the classifier with enough infor-

mation to discriminate the *rocky* image chips from the other two features classes. The cross-validation error for the *sandy* and *shadow* feature classes are also reduced as the sample region size is increased, but not nearly as drastically because their distinct feature information is highly localized.

To balance performance and accuracy, a sample region size of 100x100 pixels was chosen as a desirable parameter setting to process the side-scan images in the URL's Pam Rocks data set.

4.2.4. Neural Network Architecture

Choosing the ideal number of hidden layers (or architecture) for an artificial neural network is a difficult problem. Determining the optimal ANN architecture is beyond the scope of this thesis, however several different ANN architectures are tested using cross-validation to determine the most accurate ANN design within the set (again, without making claims of optimality).

The degrees of freedom of an ANN are the weights along the connections between neurons of different layers. Therefore, by changing the number of hidden layers and number of neurons, an designer implicitly controls the degrees of freedom of the system. In the same sense as least-squares projection, if there are too few degrees of freedom, the underlying function can't be accurately approximated and if there are too many degrees of freedom, the system tends to over-fit the data. Both cases result in undesirably large cross-validation error. In addition to over-fitting, increasing degrees of freedom will also increase computation time.

A set of ANN architectures were tested using the training data from subsection 4.2.2, with a sample window size of 100x100. Table 4.3 presents the results for eight different ANN architectures. In each case, the input layer consists of 20 neurons and the output layer consists of 3 neurons. Each number between the input and output represents the number of neurons in their corresponding hidden layer. For compactness, the last three rows of the table use 50xN to represent N hidden layers with 50 neurons.

The first row of table 4.3 presents the cross-validation error and feedforward computation time (of the entire set of sample regions from all the image chips) for a basic, input-output, ANN without any hidden layers. In practice, it is rare to see an ANN without any hidden layers and this example indicates the reason. The cross-validation error for all three class is the greatest for the basic ANN design, although the computation time is the fastest.

The cross-validation error of the *rocky* class for ANNs 20-20-3 through 20-50x3-20-3 seem to be reduced as more neurons and hidden layers are added (from 5.48% to 4.98%), however the *sandy* and *shadow* classes do not experience this trend. In fact, the cross-validation error for both *sandy* and *shadow* classes seem to vary at random between ANNs 20-20-3 through 20-50x3-20-3. Even though the cross-validation error in table 4.3 is averaged over many

ANN architecture	Cross-validation error			Computation time (ms)
	Rocky	*Sandy*	*Shadow*	
20-3	7.83%	1.88%	10.5%	17.5
20-20-3	5.48%	0.98%	9.94%	22.1
20-50-3	5.58%	0.90%	9.19%	35.3
20-20-20-3	5.36%	1.11%	10.3%	27.9
20-50-20-3	5.14%	1.03%	9.69%	52.0
20-50x3-3	5.03%	0.89%	9.17%	153.0
20-50x3-20-3	4.98%	1.02%	10.3%	182.0
20-50x11-3	5.26%	1.34%	9.70%	1220.0

Table 4.3: Cross-validation error for a range of ANN architectures. The first row only consists of an input and an output layer and it produces the largest error values for both *rocky* and *sandy* regions.

training operations, it is possible that the randomness from backpropagation is the cause of these error fluctuations. The last row of table 4.3 presents an extreme example of 11 hidden layers, where the cross-validation error values begin to increase again. This is consistent with the over-fitting problem mentioned above.

A trend that is quite important and prevalent in the various ANN designs is the increase in computation time. As the number of neurons grows, so does the time needed to compute all the activation levels. This growth is a direct result of having to sum over more activation levels (and more connection weights) and having to apply more sigmoid functions (see equations 3.2 and 3.3)[2]. For clarity, although ANN 20-20-20-3 appears below ANN 20-50-3, there are more connection weights and neurons in 20-50-3 than 20-20-20-3 and so the computation time is greater for 20-50-3.

As expected, the computation time increases with the increased ANN complexity so the desirable ANN architecture should balance accuracy and computation efficiency. A 20-20-20-3 ANN was used for this thesis because it demonstrates similar accuracy when compared to more time consuming ANN architectures in table 4.3.

4.2.5. Feature Spaces

The accuracy of the both the Hybrid R and Hybrid S feature spaces from [31] were tested using the training data from subsection 4.2.2 with a 100x100 sample region and a 20-20-20-3 ANN classifier. Table 4.4 compares the cross-validation error for both methods and confirms [31]'s claim that the Hybrid R feature space is the most accurate of the two hybrid methods. The Hybrid R method is much better at describing the *rocky* and *shadow* image texture features than the Hybrid S method. The difference between the hybrid spaces is that Hybrid R uses GLRL feature values and Hybrid S uses SLGDM feature values; the remaining feature values for both feature spaces are the same. The difference between the

[2]The exp() function in C++ is very expensive.

Cross-validation error					
Hybrid R			Hybrid S		
Rocky	Sandy	Shadows	Rocky	Sandy	Shadows
5.36%	1.11%	10.3%	10.4%	1.93%	14.6%

Table 4.4: Cross-validation error of both hybrid feature spaces for *rocky*, *sandy* and *shadow* regions.

Cross-validation error	
SVM	ANN
25.5%	3.40%

Table 4.5: Total feature space cross-validation error for the SVM and ANN classifiers.

methods and the improved accuracy with Hybrid R suggests that the run-length feature values give a more accurate description than spatial variations for *rocky* and *shadow* image chips. Greater accuracy from GLRL than SGLDM might seem counter intuitive for *shadow* examples because *shadow* regions shouldn't have large linear features like *rocky* regions. However, the shadow boundaries (like in figure 4.10) do have linear components and these boundaries are more accurately described by GLRL feature values than SGLDM.

The three base feature spaces from section 3.1 are not investigated here because [31] has shown them to be less accurate than either hybrid method. Because the Hybrid R method is the more accurate feature space with the URL's Pam Rocks data set, it was the chosen method for implementation on the GPU.

4.2.6. Classifier Comparison

For completion, the accuracy of another popular supervised learning classifier, support vector machines (SVM), was tested with the Hybrid R feature space. There does not seem to be any consensus in the literature regarding which classifier is more accurate. In general, it appears that the ideal classifier is case-specific and so one cannot say a priori that ANN is a more accurate classifier than SVM for side-scan image segmentation.

The LIBSVM Matlab library was used as an black box SVM classifier and it was trained and tested in the same manner as the ANN classifier [59]. Table 4.5 presents the average total cross-validation error of all three image texture regions for both classifiers. The ANN classifier is an order of magnitude more accurate than the SVM classifier for this particular data set. Although the cross-validation error for both classifiers might vary depending on the side-scan image data set used, the ANN classifier completely outperforms any reasonable fluctuation in the accuracy of the SVM classifier. For this particular application, the ANN classifier appears to be the more accurate of these two popular supervised learning methods.

4.2.7. Performance Analysis

Aside from classification accuracy, efficient computational performance of the image texture classification method is essential for a real-time side-scan sonar image classification solution. The previous subsections have indicated that the Hybrid R feature space, with an ANN classifier, is a reasonably accurate solution for sea-floor classification. However, currently-available libraries that were used to initially verify this solution have severe performance limitations. To demonstrate that it is possible to achieve real-time classification using the suggested methods in this thesis, the Hybrid R method was implemented in OpenCL and executed on an AMD Radeon HD 7970 GPU.

Table 4.6 presents a breakdown of the computation time for each process in the real-time classification method. Separate OpenCL kernels are used to compute the GLRL and GLDR feature values from subsection 3.1.4, as well as an OpenCL kernel to compute the four-direction mean and variance values specific to the Hybrid R feature space. The only non-GPU component is the ANN classifier stage, which is computed with the *Neural Network Toolbox* in Matlab [58].

Operation	Average computation time per ping (ms)
GLRL kernel	48.9
GLDR kernel	4.55
Hybrid R kernel	6.85
ANN classification (CPU)	39.9
Total	**100.2**

Table 4.6: Average computation time of each major component in the sea-floor profiling algorithm

The GLRL kernel and the ANN classification function are the two bottlenecks in the side-scan sonar image classification method. The GLRL kernel is expensive to compute because each kernel has to loop over its entire sample region to build its histogram. This is entirely different than the GLDR kernel because the absolute difference value at a pixel location (i, j) is the exact same for every sample region that overlaps with this pixel. Therefore, each GLDR kernel computes a single column of its sample region and shares this column with all of its neighbouring kernels. Unfortunately, the GLRL algorithm is not able to be optimized as easily, and the kernel execution is considerably more time consuming. The ANN computation time could be seriously reduced by developing an OpenCL version of the feedforward classification process for execution on a GPU. Because the activation level for each neuron in the same layer can be computed independently, computing the output activation levels for each layer is easily parallelizable.

The average per-ping computation time for the entire sea-floor classification method is approximately 100ms. Section 1.3.1 asserts that a real-time method must complete in less than 270ms, which means the proposed solution method can be considered a real-time

side-scan sonar classification method.

5. Conclusion

This thesis addresses two research problems with 3-D side-scan sonar: 1. The collected bathymetry data contains more signal returns than just the sea-floor, so the sea-floor profile must be isolated to build topographical maps, and 2. Existing image texture classification libraries used to segment side-scan sonar images are extremely time consuming and limit the ability to provide classification information whilst surveying in the field.

The sea-floor profiling method is based on fuzzy c-means clustering, where the cluster centers are connected in chains to approximate the sea-floor's point-cloud curves in the bathymetry data. The cluster chains are processed to remove any loops that might have formed by the initial connection algorithm and then each chain of clusters is processed with an adapted fuzzy c-means algorithm that reduces curvature in the chains. To account for breaks in the sea-floor bathymetry data across nadir and shadow regions, the chains are connected together with a test function based on orthogonal and projected distance from each other's chain. Another test function is applied to each remaining chain to determine the optimal chain of cluster centers. The bathymetry data close to the optimal cluster chain are marked as sea-floor bathymetry data. Experimental results demonstrate that the sea-floor profiling method is very accurate when compared to a human operator for a wide range of possible outlier data and breaks in the sea-floor profile. Additionally, the results indicate that this method is extremely fast to compute.

The side-scan sonar classification method processes a sample region of a side-scan sonar image to build a feature vector. The values in the feature vector are computed from two image texture methods: grey-level run-length and grey-level difference. These feature vectors are computed vertically, horizontally and diagonally (45° and 135°) to minimize changes due to rotation. The final feature vector for each sample region in the image is passed into an trained artificial neural network and fed forward through the neuron layers to output a classification value. This value is assigned to the pixel at the center of the sample region, and colour-coded to provide a visualization of the different sea-floor regions. The side-scan sonar classification method is able to accurately identify operator-selected image chips. A cross-validation error of 10.3% for *shadow* regions being the largest error of the three classification regions. Various *rocky* and *sandy* features in the side-scan images were verified with drop-camera footage to confirm that the operator's initial classification was correct.

The OpenCL implementation improved the computational efficiency of building the image texture feature space by orders of magnitude from the Matlab libraries and demonstrated that it is possible to provide real-time classification information.

5.1. Contributions of the Thesis

This thesis presents three significant contributes towards autonomous real-time 3D side-scan sonar classification:

1. A novel angle ray-casting method to filter outlier bathymetry data and initialize the fuzzy curve tracing algorithm. This method is extremely efficient and provides an almost-converged initialization of the cluster center positions for the fuzzy c-means portion of the FCT algorithm. More importantly, the angle ray-casting method is a repeatable and deterministic initializer, instead of random cluster center initialization from a traditional fuzzy c-means algorithm.

2. A modified curve tracing algorithm used to extract the sea-floor profile through the bathymetry data containing outlier signal returns. This method adds a repulsion penalty term to the traditional fuzzy c-means algorithm to enforce separation between clusters and, as a consequence of numerical instability, adaptively remove cluster centers when the angle ray-casting initialization creates too many. The modified method also handles multiple cluster chains and connects chains across nadir or shadow regions.

3. A real-time, texture-based sea-floor classification library by the use of GPGPU implementation. Although the classification method itself was unaltered, the GPGPU implementation required an entire redesign to be efficient on SIMD hardware. In addition, the accuracy of the classification method was verified with the URL's Pam Rocks data set by the use of drop-camera ground truthing and cross-validation error of operator selected image chips.

5.2. Future Work

The work presented in this thesis was tested on a single geographical data set collected by the URL's 3-D side-scan sonar system. Further testing of the work with various sea-floor compositions and topography are needed to convincingly argue that this is a widespread solution for generating 3-D side-scan classification maps in real-time. Additionally, the sea-floor profiling and classification methods were implemented and tested individually and never directly integrated into a sonar system. Even though integration is a engineering problem and not a research one, it is still an important task in order to unequivocally state that these methods will operate real-time during field surveys.

Although thesis achieves the goal of sea-floor profiling, it does so in a 2-D sense for each ping. By processing each ping individually, a sonar system could provide a waterfall display of the sea-floor whilst the data is being collected. However, because the sea-floor data points between consecutive pings are highly correlated, a 3-D based algorithm might provide a more accurate solution.

Because the goal of this thesis was to demonstrate that a real-time side-scan sonar classification method was possible, an existing method that investigated side-scan sonar classification was implemented on a GPGPU because the method's accuracy had already been demonstrated. Future work should include an investigation into side-scan sonar classification accuracy of the CLBP method and [42]'s novel complex network method. Although both methods have been shown to accurately describe image features from Vistex image database, the direct application to side-scan sonar images is currently unknown. Additionally, the investigation should include the viability of a GPGPU implementation of these algorithms to maintain real-time efficiency.

References

[1] "Multibeam sonar theory of operation," tech. rep., L-3 Communications SeaBeam Instruments, 2000.

[2] P. Kraeutner and J. Bird, "Principal components array processing for swath acoustic mapping," in *OCEANS '97. MTS/IEEE Conference Proceedings*, vol. 2, pp. 1246–1254 vol.2, 1997.

[3] J. Bird, S. Asadov, and P. Kraeutner, "Improving arrays for multi-angle swath bathymetry," in *OCEANS 2003. Proceedings*, vol. 4, pp. 2085–2092 Vol.4, 2003.

[4] P. Denbigh, "Swath bathymetry: principles of operation and an analysis of errors," *Oceanic Engineering, IEEE Journal of*, vol. 14, no. 4, pp. 289–298, 1989.

[5] X. Lurton, "Swath bathymetry using phase difference: theoretical analysis of acoustical measurement precision," *Oceanic Engineering, IEEE Journal of*, vol. 25, no. 3, pp. 351–363, 2000.

[6] J. Bird and G. Mullins, "Bathymetric sidescan sonar bottom estimation accuracy: Tilt angles and waveforms," *Oceanic Engineering, IEEE Journal of*, vol. 33, no. 3, pp. 302–320, 2008.

[7] J. D. Penrose, P. J. W. Siwabessy, A. Gavrilov, I. Parnum, L. J. Hamilton, A. Bickers, B. Brooke, D. A. Ryan, and P. Kennedy, "Acoustic techniques for seabed classification," tech. rep., Cooperative Research Centre for Coastal Zone Estuary and Waterway Management, 2005.

[8] H. Edelsbrunner and E. P. Mücke, "Three-dimensional alpha shapes," *ACM Trans. Graph.*, vol. 13, pp. 43–72, Jan. 1994.

[9] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans, "Reconstruction and representation of 3d objects with radial basis functions," in *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, (New York, NY, USA), pp. 67–76, ACM, 2001.

[10] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, (Aire-la-Ville, Switzerland, Switzerland), pp. 61–70, Eurographics Association, 2006.

[11] M. Berger, J. A. Levine, L. G. Nonato, G. Taubin, and C. T. Silva, "A benchmark for surface reconstruction," *ACM Trans. Graph.*, vol. 32, pp. 20:1–20:17, Apr. 2013.

[12] P. Cervenka and C. de Moustier, "Postprocessing and corrections of bathymetry derived from sidescan sonar systems: application with seamarc ii," *Oceanic Engineering, IEEE Journal of*, vol. 19, no. 4, pp. 619–629, 1994.

[13] A. E. Johnson and M. Hebert, "Seafloor map generation for autonomous underwater vehicle navigation," *Autonomous Robots*, vol. 3, no. 2, pp. 145–168, 1996.

[14] E. Coiras, Y. Petillot, and D. Lane, "An expectation-maximization framework for the estimation of bathymetry from side-scan sonar images," in *Oceans 2005 - Europe*, vol. 1, pp. 261–264 Vol. 1, 2005.

[15] E. Coiras, Y. Petillot, and D. Lane, "Multiresolution 3-d reconstruction from side-scan sonar images," *Image Processing, IEEE Transactions on*, vol. 16, no. 2, pp. 382–390, 2007.

[16] Y. Lu and M. Oshima, "3-d reconstruction of the underwater acoustic images using the gps positioning and the image matching technology," in *OCEANS '02 MTS/IEEE*, vol. 4, pp. 2273–2278 vol.4, 2002.

[17] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[18] R. Raguram, J.-M. Frahm, and M. Pollefeys, "A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus," in *Computer Vision âĂŞ ECCV 2008* (D. Forsyth, P. Torr, and A. Zisserman, eds.), vol. 5303 of *Lecture Notes in Computer Science*, pp. 500–513, Springer Berlin Heidelberg, 2008.

[19] J. Bezdek and I. Anderson, "An application of the c-varieties clustering algorithms to polygonal curve fitting," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-15, pp. 637–641, Sept 1985.

[20] R. N. Dave, "Generalized fuzzy c-shells clustering and detection of circular and elliptical boundaries," *Pattern Recognition*, vol. 25, no. 7, pp. 713 – 721, 1992.

[21] Y. Man and I. Gath, "Detection and separation of ring-shaped clusters using fuzzy clustering," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 16, pp. 855–861, Aug 1994.

[22] H. Yan, "Fuzzy curve-tracing algorithm," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 31, pp. 768 –780, oct 2001.

[23] H. Yan, "Convergence condition and efficient implementation of the fuzzy curve-tracing (fct) algorithm," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 34, pp. 210–221, Feb 2004.

[24] Y. Liu, H. Yang, and W. Wang, "Reconstructing b-spline curves from point clouds–a tangential flow approach using least squares minimization," in *Shape Modeling and Applications, 2005 International Conference*, pp. 4–12, June 2005.

[25] Z. Reut, N. G. Pace, and M. J. P. Heaton, "Computer classification of sea beds by sonar," *Nature*, vol. 314, pp. 426–426, 1985.

[26] N. Pace and H. Gao, "Swathe seabed classification," *Oceanic Engineering, IEEE Journal of*, vol. 13, no. 2, pp. 83–90, 1988.

[27] R. Haralick, K. Shanmugam, and I. Dinstein, "Textural features for image classification," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-3, no. 6, pp. 610–621, 1973.

[28] J. S. Weszka, C. R. Dyer, and A. Rosenfeld, "A comparative study of texture measures for terrain classification," *Systems, Man and Cybernetics, IEEE Transactions on*, vol. SMC-6, no. 4, pp. 269–285, 1976.

[29] T. B. Reed and D. Hussong, "Digital image processing techniques for enhancement and classification of seamarc ii side scan sonar imagery," *Journal of Geophysical Research: Solid Earth*, vol. 94, no. B6, pp. 7469–7490, 1989.

[30] W. Stewart, M. Marra, and M. Jiang, "A hierarchical approach to seafloor classification using neural networks," in *OCEANS '92. Mastering the Oceans Through Technology. Proceedings.*, vol. 1, pp. 109–113, 1992.

[31] W. Stewart, M. Jiang, and M. Marra, "A neural network approach to classification of sidescan sonar imagery from a midocean ridge area," *Oceanic Engineering, IEEE Journal of*, vol. 19, no. 2, pp. 214–224, 1994.

[32] J. H. Lee, N. I. Lee, and S. D. Kim, "A fast and adaptive method to estimate texture statistics by the spatial gray level dependence matrix (sgldm) for texture image segmentation," *Pattern Recognition Letters*, vol. 13, no. 4, pp. 291 – 303, 1992.

[33] M. M. Galloway, "Texture analysis using gray level run lengths," *Computer Graphics and Image Processing*, vol. 4, no. 2, pp. 172 – 179, 1975.

[34] B. V. Dasarathy and E. B. Holder, "Image characterizations based on joint gray level-run length distributions," *Pattern Recogn. Lett.*, vol. 12, pp. 497–502, Aug. 1991.

[35] D. Alexandrou and D. Pantzartzis, "A methodology for acoustic seafloor classification," *Oceanic Engineering, IEEE Journal of*, vol. 18, no. 2, pp. 81–86, 1993.

[36] A. Laine and J. Fan, "Texture classification by wavelet packet signatures," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 15, no. 11, pp. 1186–1191, 1993.

[37] T. Chang and C.-C. Kuo, "Texture analysis and classification with tree-structured wavelet transform," *Image Processing, IEEE Transactions on*, vol. 2, no. 4, pp. 429–441, 1993.

[38] X. Tang and W. Stewart, "Texture classification using wavelet packet and fourier transforms," in *OCEANS '95. MTS/IEEE. Challenges of Our Changing Global Environment. Conference Proceedings.*, vol. 1, pp. 387–396 vol.1, 1995.

[39] X. Tang and W. Stewart, "Optical and sonar image classification: Wavelet packet transform vs fourier transform," *Computer Vision and Image Understanding*, vol. 79, no. 1, pp. 25 – 46, 2000.

[40] G. R. Cochrane and K. D. Lafferty, "Use of acoustic classification of sidescan sonar data for mapping benthic habitat in the northern channel islands, california," *Continental Shelf Research*, vol. 22, no. 5, pp. 683 – 690, 2002.

[41] X. Tang, "Texture information in run-length matrices," *Image Processing, IEEE Transactions on*, vol. 7, no. 11, pp. 1602–1609, 1998.

[42] A. R. Backes, D. Casanova, and O. M. Bruno, "Texture analysis and classification: A complex network-based approach," *Information Sciences*, vol. 219, no. 0, pp. 168 – 180, 2013.

[43] I. Marsh and C. Brown, "Neural network classification of multibeam backscatter and bathymetry data from stanton bank (area iv)," *Applied Acoustics*, vol. 70, no. 10, pp. 1269 – 1276, 2009.

[44] K. Raghupathy and T. Parks, "Improved curve tracing in images," in *Acoustics, Speech, and Signal Processing, 2004. Proceedings. (ICASSP '04). IEEE International Conference on*, vol. 3, pp. iii – 581–4 vol.3, may 2004.

[45] J. C. Dunn, "A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, no. 3, pp. 32–57, 1973.

[46] J. C. Bezdek, *Pattern Recognition with Fuzzy Objective Function Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1981.

[47] N. Pal and J. Bezdek, "On cluster validity for the fuzzy c-means model," *Fuzzy Systems, IEEE Transactions on*, vol. 3, pp. 370 –379, aug 1995.

[48] H. Timm, C. Borgelt, C. Döring, and R. Kruse, "Fuzzy cluster analysis with cluster repulsion," in *Proc. European Symposium on Intelligent Technologies (EUNITE, Tenerife, Spain), on CDROM. Verlag Mainz, Aachen, Germany*, Citeseer, 2001.

[49] H. Timm, C. Borgelt, C. DÃűring, and R. Kruse, "An extension to possibilistic fuzzy cluster analysis," *Fuzzy Sets and Systems*, vol. 147, no. 1, pp. 3 – 16, 2004. Hybrid Methods for Adaptive Systems.

[50] X. Wei, "Gray level run length matrix toolbox v1.0," Nov 2007. MATLAB Central File Exchange.

[51] A. Narayanan, "Texture feature extraction - gldm," Aug 2009. MATLAB Central File Exchange.

[52] MATLAB, *version 7.10.0 (R2010a)*, ch. Gray-level Co-ocurrence Matrix. Natick, Massachusetts: The MathWorks Inc., 2010.

[53] A. Chu, C. Sehgal, and J. Greenleaf, "Use of gray value distribution of run lengths for texture analysis," *Pattern Recognition Letters*, vol. 11, no. 6, pp. 415 – 419, 1990.

[54] S. Grossberg, ed., *Neural Networks and Natural Intelligence*. Cambridge, MA, USA: Massachusetts Institute of Technology, 1988.

[55] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1," ch. Learning Internal Representations by Error Propagation, pp. 318–362, Cambridge, MA, USA: MIT Press, 1986.

[56] J. E. Stone, D. Gohara, and G. Shi, "Opencl: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 3, 2010.

[57] A. Munshi, B. Gaster, and T. Mattson, *OpenCL Programming Guide*. Graphics programming, ADDISON WESLEY Publishing Company Incorporated, 2011.

[58] MATLAB, *version 7.10.0 (R2010a)*, ch. Neural Network Toolbox. Natick, Massachusetts: The MathWorks Inc., 2010.

[59] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.