Final Report: Flagging poor-quality scientific imagery in distributed memory

Cameron S. Bodine
School of Informatics, Computing, and Cyber Sytems
Northern Arizona University
Flagstaff, United States of America
csb67@nau.edu

I. Introduction

Scientists are increasingly using low-cost, recreation-grade sonar systems to image benthic environments to inform research in aquatic systems [1]–[10]. Scientific-grade sonar systems utilize multiple instruments to measure conditions (i.e., sound velocity profiles, secchi depth, etc.) during the time of the survey which are used to post-process the imagery and apply corrections. These instruments, in part, drive up the cost of the sonar system and require significant expertise from the field technician to collect and operate correctly. Recreation-grade sonar systems, in contrast, are easy to operate and require minimal expertise, but also aren't equipped with instruments to correct the imagery. Very little information about the mechanics of the sonar system is provided by the manufacturer, leaving the research scientist with few options to post-process and correct these data. In fact, the returned backscatter data from these systems are 8-bit integers in the range [0-255], but the units from these data are unknown. Contrasted with survey-grade systems which report backscatter returns in decibels, recreation-grade sonar users are left with no ability to apply physical models to post-process and correct the data. If these low-cost sonar systems are to be applied at large spatial extents, workflows must be developed to quickly identify and flag imagery which is not interpretable or useful for analysis.

As sound is emitted from the sonar transducer, the sound will spread and attenuate as it moves through the water column, meaning much of the energy will be lost, leaving little backscatter to be returned to the sonar transducer. For side scan sonar imagery, this results in very dark imagery as the range from the transducer to the waterbed increases. Fig. 2-3 show a 3 x 3 m portion of a riverbed, which from previous experience, is the approximate minimum mapping unit for visually identify substrates once properly trained. Fig. 3 shows sonar images, or sonograms, which have experienced backscatter loss and are too dark to visually distinguish features. Sonograms with minimal backscatter loss are shown in Fig. 2 in which features can begin to be distinguished. For example, the first panel shows some sand ripples. The higher intensity pixels are those with an aspect oriented towards the sonar transducer while darker pixels are in the lee of the ripple or simply lower in elevation to surrounding pixels. In the second and third panels, most of the bed is flat and homogenous, while linear objects, indicative of tree trunks and branches, are set upon the bed and thus, cast a shadow in the lee of the sonar pulse. Distinguishing sonograms with high-quality backscatter from those with low is important for developing a high-quality dataset for training subsequent substrate segmentation models.

II. Background

Preliminary work has shown that principal components analysis (PCA) combined with k-means clustering can quickly identify sonar images that are low-contrast and too dark to interpret. PCA is a dimension reduction technique to distill multidimensional datasets into their essential features, or principal components.

A set of sample sonograms with dimension 171 x 171 pixels (true-world dimensions of ~3 x 3 m) were processed with a Python script and the first two principal components were extracted using the SciKit-Image Python package. The reduced dimension was then plotted with the x-axis relating to the first principal component (PC1) and the y-axis the second (PC2) (Fig. 1). These points were then clustered using the K-Means functionality in SciKit-Image. Tight clustering of the points prompted for a cluster size of 25 to be chosen so that each cluster could be more closely investigated to determine causes of variation between clusters. Initial investigation of PC1 found that clusters with values near or below 0 were too dark for visual interpretation (e.g., clusters 1, 21, 9, 7, 22, and 3 in Fig. 3) meaning clusters larger than 0 are bright enough to interpret (e.g., clusters

20, 6, 4, 17 in Fig. 3). PC2 is useful for determining the amount of no-data present in the sonogram. These are the 'arms' with very high or very low PC2 values in Fig. 3. Whether these values are positive or negative tells us if the no-data region occurs near the top or bottom of the sonogram, respectively. Therefore, we can use this approach to identify both images which are too dark for visual interpretation and have too much no-data to allow for substrate classification.

Applying this at scale, however, is time consuming as a typical day in the field can result in ~50 km of surveyed river resulting in a continuous sonogram of size 36,000 x 1,500 with study systems exceeding 500 km, this becomes computationally intractable for a personal computer. This project develops and implements a workflow, written in C, that is run on a cluster to process many datasets in a time-efficient manner.

III. Methods

A. *Algorithm Overview*

The following section outlines the original proposed algorithm workflow to accomplish the objectives of this project. Steps 1-4.a were implemented in the algorithm. Adequate time was not planned for to implement 4.b-5, however, future development of the algorithm will incorporate these steps.

1. Store sonar image files on Monsoon
2. Distribute workload
    a. Rank 0 will scan the image directory.
    b. Determine the size of all images.
    c. Perform a histogram procedure to evenly distribute work across *p* process ranks.
    d. Use MPI_Bcast() to assign the images to the process ranks.
3. Principal Components Analysis (PCA)
    a. Each rank will open their image and store pixel values in memory.
    b. *C* principal components are calculated from the pixel values and stored in memory.
4. K-Means clustering
    a. Each rank will perform K-means clustering on the principal component values.
    b. Preliminary runs on small number of images will help determine thresholds at which clusters should be flagged, indicating poor image quality. (NOT IMPLEMENTED).
5. Flagged image path and file name's will be written to a csv file so they can be removed from the dataset. Alternatively, flagged images will be moved to a sub-directory. (NOT IMPLEMENTED).

B. *Distribute Workload*

The sonar chips were uploaded to Monsoon cluster at Northern Arizona University (NAU). To ensure that the workload was balanced between process ranks, the image directory was iterated by Rank 0 to determine the size of each of the sonar images. This was not strictly necessary for this application as the sonar chips were all the same size. However, future applications of the algorithm will need to be flexible enough to work with files of any size. The total size of the entire dataset was calculated, and the workload for each rank was determined by dividing total dataset size by the number of process ranks. Specific images were assigned to each process rank through a histogram procedure like that from CS599 Module 3: Sorting. Assignment of workload was communicated to all process ranks with the OpenMPI call MPI_Bcast.

C. *Principal Components Analysis (PCA)*

Once each process rank received their respective workload, PC's for each sonar image were calculated. For each image, first the image was opened to access the pixel values. The values were stored in a mxn matrix, where m is the number of rows and n is the number of columns. The values in the matrix were then normalized and centered by subtracting the image mean from each pixel, resulting in values $[-1, 1]$. The variance-covariance matrix was then calculated, resulting in a symmetric matrix.

Eigen vectors and values were then calculated from the variance-covariance matrix. The two largest eigen values and associated vectors represent the first two principal components. The image data were then reduced to two dimensions by multiplying the pixel matrix by the eigen vectors for the first two principal components. These two numbers were then stored in a reduced dimension matrix. The algorithm has been designed to allow computing of any number of principal components dependent of the needs of the application.

D. *K-Means Clustering*

A distributed memory K-Means clustering algorithm following the implementation developed for CS599 Module 5: K-Means to cluster the reduced dimensions of each image. A key challenge in distributed memory clustering is the need to communicate the cluster membership count across process ranks and update cluster centers globally. To allow for this, Rank 0 initialized the centroids for the clusters from the first k reduced dimension matrix. This was communicated to each process rank using the OpenMPI MPI_Bcast() call.

Each process rank then iterates each local image's reduced dimension matrix and calculates the Euclidean distance from those values to each k centroid. An image was then classified as belonging to a specific cluster if it was the minimum distance. An image's cluster membership was stored in an array, and the local count of image's were stored in another array.

After each iteration of determining each point's membership to a cluster, the global count of images per cluster was calculated using the OpenMPI MPI_Reduce() call with an MPI_SUM function at Rank 0. The global cluster count was then communicated to each rank. The global cluster count was used by each rank to calculate the weighted mean of the points belonging to each cluster. The local weighted means were then sent to Rank 0 with another MPI_Reduce() call with MPI_SUM function. These summed means represent the new cluster centroids based on the points belonging to a given cluster. This procedure was repeated for a total of ten iterations, but more iterations could be added by editing the KMEANSITERS value in the algorithm.

E. Algorithm Steps NOT IMPLEMENTED

Adequate time was not planned for to implement steps 4.b-5. This would be straightforward to implement in the future, however.

IV. RESULTS

The following sub-sections list the testing performance of the algorithm on a test dataset, development roadblocks and workarounds.

A. Test Dataset

Side scan sonar imagery collected by the Florida Fish and Wildlife Conservation Commission (FWC) in April 2016 on the Escambia River were used to test the performance of the PCA and K-means algorithm developed for this proposal. A Humminbird© side scan sonar device was installed on a jon boat. The sonar transducer was fixed to a pole and mounted to the bow of the boat using a transom motor mount. The sonar control head was mounted near the steering wheel. The riverbed was scanned with the sonar device at an average speed of 7.5 km/hr. Approximately 50 km of the river were scanned. Sonar data were recorded to a sonar recording file.

Sonar recordings were warped and geographically located using SonarTRX [11]. Exported sonar images were then mosaiced in sonar maps in a geographic information system (GIS). The georeferenced mosaics were then sampled using a Python script to export 3m x 3m (165 pix x 165 pix) (Fig. 2-3), referred to here as sonar chips, resulting in a total of 47,096 sonar chips with a total size of ~1.16 GB.

B. Performance Testing

The test dataset was uploaded to NAU's Monsoon cluster on the scratch drive. The algorithm script and dependency library header files were compiled with the -O3 flag. The tests were all run on two Skylake Xeon compute nodes, each with 28 cores, 2.6 GHz, and 196 GB of memory. For each test, the total response time, maximum time to distribute the workload, maximum time to calculate image principal components, and maximum time to perform K-Means clustering was recorded (Tbl. 1).

Four different tests were run on the test dataset, where each test was run with a different number of process ranks: $p = 1, 10, 20, 30\ and\ 40$. Two compute nodes were used for each of the tests, with the number of ranks per compute nodes evenly distributed. Each of the four tests were run three times, and times reported in Tbl. 1 averaged across each test run.

# of Ranks (*p*)	**Total Time (s)**	**Time to Distribute (s)**	**PCA Time (s)**	**K-Means Time (s)**	**Parallel Speedup**
1	379.62	42.536	337.000	3.973	-
10	60.68	26.278	34.345	2.888	6.256
20	43.34	26.034	17.187	1.555	8.758
30	38.30	26.512	11.562	1.242	9.911
40	37.07	27.756	9.066	1.282	10.242

Tbl. 1 Algorithm performance results on the test dataset.

Tbl. 1 shows the results from the tests. Parallel speedup was calculated based on the total processing time. Total processing time decreased as the number of ranks increased. Total processing time does not decrease as quickly as the number of processing ranks increases from 30 to 40, however. The time to distribute the data remains constant for $p > 1$ process ranks. It is unclear why the time to distribute was so large for only one process rank. The constant time to distribute indicates that this procedure is a bottleneck and additional effort in developing alternative approaches for distributing the workload could improve performance. The time to calculate principal components and perform data reduction decreased as the number of ranks increased.

Performing K-Means clustering represented the smallest proportion of the total processing time. This would likely be different if a larger number of PC's were used in the algorithm. The results show that more work is needed to optimize the algorithm to achieve better performance.

C. *Comparison to Python*

This algorithm was originally developed in Python using the SciKit-Learn package to calculate PCA and perform clustering. A test was conducted on 20 [165x165] sonar images. Calculating PCA and performing clustering too approximately 30 seconds, meaning each image takes approximately 1.5 s. For images of this size, assuming there are ~47,000, and assuming clustering will scale with the number of images, this would take my computer approximately 19.6 hours to compute.

Developing this algorithm in C/C++ shows dramatic performance improvement, even when processing on a single rank. Even though the processing results shown in Tbl. 1 are not ideal, when compared to processing sequentially with a Python script, the outcome is drastically better.

D. *Development Roadblocks*

This project experienced significant roadblocks during the research and development of the algorithm. The most difficult challenge to overcome was incorporating libraries from other developers in the workflow. I have minimal experience coding in C and C++, let alone integrating outside libraries into the workflow.

My initial research into working with imagery datasets led to OpenCV (https://opencv.org/), a popular C++ library for working with, analyzing, and manipulating image datasets. The library provides functions for reading pixel values into arrays as well as a function to calculate principal components from those values.

Installing and utilizing OpenCV on Monsoon quickly resulted in challenges. OpenCV is not included in the modules installed on Monsoon. Consultations with Monsoon HPC staff showed how to install the library in an Anaconda (https://www.anaconda.com) virtual library. I was able to successfully install OpenCV into a virtual library, but was not successful in integrating the library into my algorithm. When I tried to compile my algorithm script, the compiler was looking for file versions which were not part of the library. Since I did not have administrative privileges on Monsoon, I was unable to apply any workarounds I found on the internet.

E. *Roadblock Workaround Strategy*

Without the use of OpenCV, I sought out help from my classmate Andrew to read image pixels into a matrix. Andrew found a header-only library called stb (https://github.com/nothings/stb) that provides functionality to load images into memory.

Once images contents could be loaded, I found another library called Eigen (https://eigen.tuxfamily.org). The developers of Eigen specifically designed it to be easily integrated into any processing workflow without the need to install anything. I was able to use Eigen to calculate eigen values and eigen vectors from the image pixel values and perform additional linear algebra to decompose the pixel values into fewer features, allowing for K-Means clustering.

This was the first time I have included other C/C++ libraries into my own scripts. It was excellent experience and I think if I had allowed more time for the project that I would have developed my algorithm better.

V. Conclusion

This project successfully implemented a PCA and K-Means algorithm in C/C++ that can be used to quickly perform image data reduction and cluster the data. This approach can be used to identify sonar images with similar spectral properties, most important of which, is to identify and flag images with poor or undesirable spectral properties. Converting an existing Python workflow to a distributed memory algorithm showed dramatic performance increase by reducing overall processing time from hours to minutes or less. This performance enhancement provides an opportunity to better understand and filter sonar data which is undesirable for further analysis.

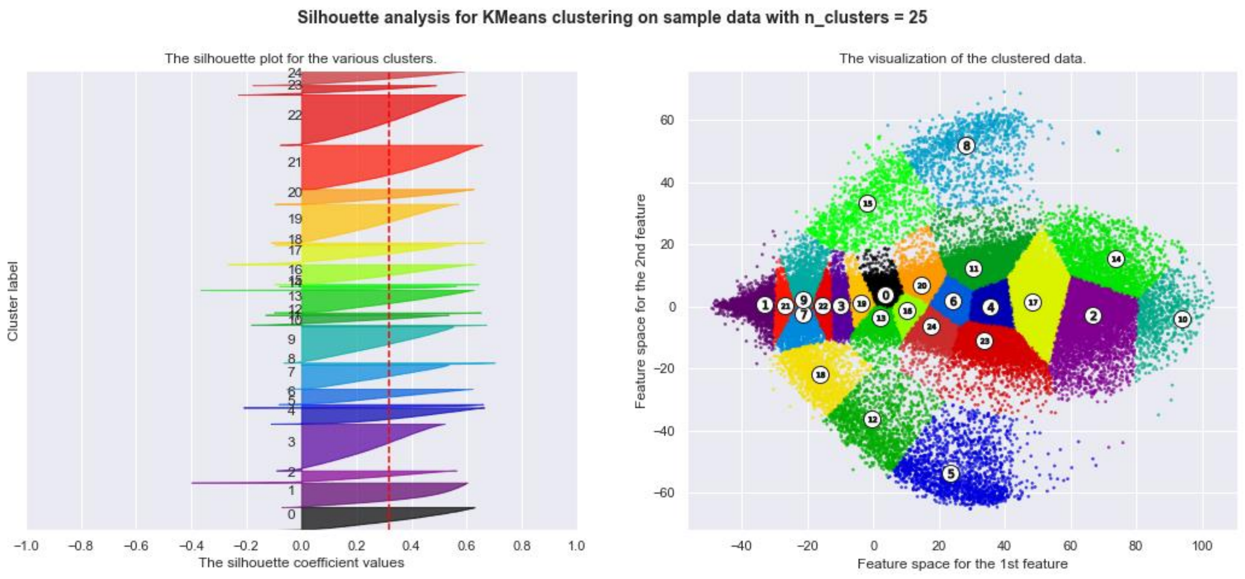

Fig. 1. K-Means clustering of principal components 1 and 2. Points on far right of plot have the highest intensity and best image quality. Quality degrades as you move to the left. Upper and lower arms belong to images with significant no-data present.

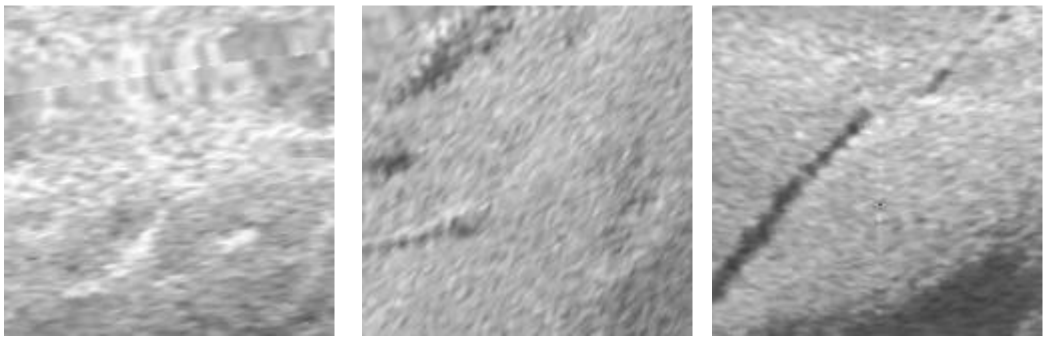

Fig. 2. Sonar imagery of approximately 3 x 3 m portion of a riverbed with a sand bottom. High quality imagery.

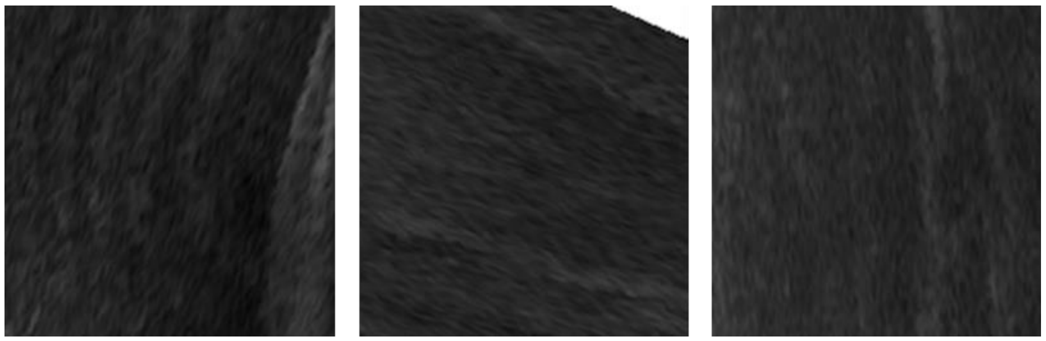

Fig. 3. Sonar imagery that is too dark to interpret and therefore bad quality.

References

[1] R. Smit and A. J. Kaeser, “Defining freshwater mussel mesohabitat associations in an alluvial, Coastal Plain river,” *Freshw. Sci.*, vol. 35, no. 4, pp. 1276–1290, Dec. 2016.

[2] A. J. Kaeser and T. L. Litts, “An Assessment of Deadhead Logs and Large Woody Debris Using Side Scan Sonar and Field Surveys in Streams of Southwest Georgia,” *Fisheries*, vol. 33, no. 12, pp. 589–597, Dec. 2008.

[3] A. J. Kaeser and T. L. Litts, “An Illustrated Guide to Low-cost, Side Scan Sonar Habitat Mapping,” 2013.

[4] A. J. Kaeser and T. L. Litts, “A Novel Technique for Mapping Habitat in Navigable Streams Using Low-cost Side Scan Sonar,” *Fisheries*, vol. 35, no. 4, pp. 163–174, Apr. 2010.

[5] A. J. Kaeser, R. Smit, and M. Gangloff, “Mapping and modeling the distribution, abundance, and habitat associations of the endangered fat threeridge in the Apalachicola river system,” *Journal of Fish and Wildlife Management*, vol. 10, no. 2. U.S. Fish and Wildlife Service, pp. 653–675, 2019.

[6] M. R. Goclowski, A. J. Kaeser, and S. M. Sammons, “Movement and Habitat Differentiation among Adult Shoal Bass, Largemouth Bass, and Spotted Bass in the Upper Flint River, Georgia,” *North Am. J. Fish. Manag.*, vol. 33, no. 1, pp. 56–70, Feb. 2013.

[7] A. J. Kaeser, T. L. Litts, and T. W. Tracy, “Using low-cost side-scan sonar for benthic mapping throughout the lower Flint River, Georgia, USA,” *River Res. Appl.*, vol. 29, no. 5, pp. 634–644, Jun. 2013.

[8] D. Buscombe, “Shallow water benthic imaging and substrate characterization using recreational-grade sidescan-sonar,” *Environ. Model. Softw.*, vol. 89, pp. 1–18, 2017.

[9] D. Hamill, D. Buscombe, and J. M. Wheaton, “Alluvial substrate mapping by automated texture segmentation of recreational-grade side scan sonar imagery,” *PLoS One*, vol. 13, no. 3, Mar. 2018.

[10] D. Buscombe, P. E. Grams, and S. M. C. Smith, “Automated Riverbed Sediment Classification Using Low-Cost Sidescan Sonar,” *J. Hydraul. Eng.*, vol. 142, no. 2, p. 6015019, Feb. 2016.

[11] Leraand Engineering Inc., “SonarTRX [Computer Software].” 2022.

.