Lossy Compression and Real-Time Geovisualization for Ultra-Low Bandwidth Telemetry from Untethered Underwater Vehicles

by

Christopher Alden Murphy

B.S., Franklin W. Olin College of Engineering, 2006

Submitted to the Dept. of Electrical Engineering and Computer Science
& Dept. of Applied Ocean Sciences and Engineering
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

and the

WOODS HOLE OCEANOGRAPHIC INSTITUTION

September 2008

Author .....
Dept. of Electrical Engineering and Computer Science
& Dept. of Applied Ocean Sciences and Engineering
August 29, 2008

Certified by .....
Dr. Hanumant Singh
Associate Scientist
Thesis Supervisor

Accepted by .....
Dr. Alexandra Techet
Chair, AOSE Joint Committee
Woods Hole Oceanographic Institution

Accepted by .....
Professor Terry P. Orlando
Chair, EECS Department Committee on Graduate Students
Massachusetts Institute of Technology

Lossy Compression and Real-Time Geovisualization for Ultra-Low Bandwidth Telemetry from Untethered Underwater Vehicles

by

Christopher Alden Murphy

Submitted to the Dept. of Electrical Engineering and Computer Science
& Dept. of Applied Ocean Sciences and Engineering
on August 29, 2008, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Oceanographic applications of robotics are as varied as the undersea environment itself. As underwater robotics moves toward the study of dynamic processes with multiple vehicles, there is an increasing need to distill large volumes of data from underwater vehicles and deliver it quickly to human operators. While tethered robots are able to communicate data to surface observers instantly, communicating discoveries is more difficult for untethered vehicles. The ocean imposes severe limitations on wireless communications; light is quickly absorbed by seawater, and tradeoffs between frequency, bitrate and environmental effects result in data rates for acoustic modems that are routinely as low as tens of bits per second. These data rates usually limit telemetry to state and health information, to the exclusion of mission-specific science data.

In this thesis, I present a system designed for communicating and presenting science telemetry from untethered underwater vehicles to surface observers. The system's goals are threefold: to aid human operators in understanding oceanographic processes, to enable human operators to play a role in adaptively responding to mission-specific data, and to accelerate mission planning from one vehicle dive to the next. The system uses standard lossy compression techniques to lower required data rates to those supported by commercially available acoustic modems ($O(10)$ – $O(100)$ bits per second).

As part of the system, a method for compressing time-series science data based upon the Discrete Wavelet Transform (DWT) is explained, a number of low-bitrate image compression techniques are compared, and a novel user interface for reviewing transmitted telemetry is presented. Each component is motivated by science data from a variety of actual Autonomous Underwater Vehicle (AUV) missions performed in the last year.

Thesis Supervisor: Dr. Hanumant Singh
Title: Associate Scientist

Acknowledgments

First, thanks to my advisor, **Hanu Singh**, for taking me to latitudes ranging from 8° South to 85° North in support of underwater exploration, all while giving me the latitude to explore a variety of topics while here at home. On each expedition, I've learned something new about oceanography, robotics, myself and the amazing world we live in. For that I owe thanks not only to Hanu, but also to my labmate **Clay Kunz**, former Joint Program students **Mike Jakuba**, **Ryan Eustice** and **Chris Roman**, Engineer **John Bailey**, and the scientists, especially **Rob Reves-Sohn** and **Roy Armstrong**, who have made a space for me in their expeditions. Thanks also to **Ko-ichi Nakamura** for sharing his Eh data.

While at the Deep Submergence Lab, I owe thanks to numerous individuals who make life better every day, through acts small (conversations, bringing dogs) and large (ensuring I have funding, housing and travel arrangements). For these acts of kindness, I thank **Ann, Cathy, Dana, Judy, Marsha**, and **Valerie**. At MIT, **Bill Freeman** and **Al Oppenheim** have both guided me and supported me in my first few years of graduate life, and I thank them for the time and space they've offered me. Thanks also to **Rob Wang** who started investigating some aspects of vector quantization with me before this thesis became all consuming. Hopefully we'll get back to it soon!

This thesis would not be complete, and indeed I wouldn't be where I am today, without the extensive support of my family. **Mom**, **Kathryn**, **Susan**, thank you for always being there for me when I needed you. I love each of you, and appreciate that I can turn to you for guidance and support on any day, at any time. Susan, thank you especially for taking my screwy schedule and lack of sleep over the past few weeks in stride, for always being ready with words of encouragement, and for reading draft after draft. I miss the conversations with my **Dad** that I'm sure would have been spurred by this thesis, and I think of him for the passion to learn he instilled in both me and my sister.

Finally, thanks to the **National Science Foundation Center for Subsurface Sensing and Imaging (CenSSIS ERC)** for supporting this research. The National Science Foundation, for which inflation-adjusted funding has decreased since 2003, serves a crucial mission in the United States. I am proud to have been funded by the CenSSIS ERC through grant EEC-99868321, and hope that our elected representatives will continue to recognize the crucial role that NSF plays through increased funding.

Contents

1 Introduction **12**

2 Background **15**

2.1 Underwater Vehicles 15

2.2 Underwater Communications 17

2.3 Scalar Time-Series Telemetry 19

2.4 Photo Telemetry 20

2.5 User Interfaces 20

3 Telemetry Classes **22**

3.1 AUV State Information 22

3.1.1 The AGAVE Expedition: Recovery in Complex Environments 24

3.2 Time-Series Scalar Data 27

3.3 Imagery 28

3.3.1 Cameras as Scalar Sensors 28

4 Telemetry Compression **33**

4.1 Time-Series Scalar Data 33

4.1.1 Methods for Lossless Compression 34

4.1.2 Introduction to Lossy Compression Techniques 35

4.1.3 Source Encoders: The DCT and DWT 35

4.1.4 Quantization, Entropy Coding, and Packet Forming 41

4.1.5 Results 44

4.2 Compression of Image Telemetry 49

4.2.1 JPEG 50

4.2.2 Wavelet Methods 51

4.2.3 Vector Quantization 51

5 User Interface **61**

5.1 Representative Interface 62

5.2 Architecture 63

5.2.1 Telemetry Decoding 64

5.2.2 KML Generation 65

6 Conclusions **67**

6.1 User Interface 68

6.2 Data Compression 69

A Wavelet Coding Source Code **71**

List of Figures

1-1 Example photos from underwater robots . . . 12
1-2 Example time series . . . 13

2-1 Seabotix and JASON II Remotely Operated Vehicles (ROVs) . . . 15
2-2 Seabed-class Puma AUV . . . 16
2-3 Acoustic transmission loss . . . 17
2-4 WHOI Micro-Modem . . . 19
2-5 REMUS and Seabed Topside software . . . 21
2-6 GeoZui3D, DVLNAV, and Seabed-Plot software . . . 21

3-1 Aliasing of heading sensor . . . 23
3-2 Arctic vehicle operations . . . 24
3-3 Under-ice 'acoustically tethered' vehicle recovery . . . 25
3-4 Scalar time-series data used in examples . . . 28
3-5 Example of using a camera as a scalar sensor . . . 29
3-6 Uncorrected color image . . . 30
3-7 Sample coral reef images . . . 32

4-1 Standard lossy signal encoder . . . 35
4-2 First four basis functions of the Discrete Cosine Transform (DCT) . . . 36
4-3 Example DWT transformation . . . 37
4-4 Wavelet families . . . 38
4-5 Comparison between DCT and DWT . . . 40
4-6 Contents of Wavelet-encoded packet . . . 42
4-7 Percentage makeup of a wavelet-encoded packet . . . 45
4-8 Wavelet compressed telemetry: lowest rate . . . 46

4-9 Wavelet compressed telemetry: medium rate 47
4-10 Wavelet compressed telemetry: high rate 48
4-11 Example of using a camera as a scalar sensor 49
4-12 JPEG compression at low data rates 50
4-13 Wavelet compression results (image 1) 52
4-14 Wavelet compression results (image 2) 53
4-15 Wavelet compression results (image 3) 54
4-16 Vector quantization results (image 1) 56
4-17 Vector quantization results (image 2) 57
4-18 Results for 5x5 tile matching (image 1) 59
4-19 Results for 5x5 tile matching (image 2) 60

5-1 Example KML in Google Earth 62
5-2 Streaming data in Google Earth 63
5-3 KML server system diagram 64
5-4 KML file structure 65

6-1 Prototype User Interface 69

List of Tables

3.1 Seabed CCL packet contents 22
3.2 Data produced by scalar sensors 27

4.1 Lossless compression ratios 34
4.2 Sample TDMA schedule for deep chemical sensing 43
4.3 Image transmission rate table 50

Glossary

Underwater Robotics

AUV Autonomous Underwater Vehicle

HOV Human Occupied Vehicle

LBL Long Baseline

REMUS Remote Environmental Monitoring UnitS

ROV Remotely Operated Vehicle

USBL Ultra-Short Baseline

Sensors

ADCP Acoustic Doppler Current Profiler

CTD Conductivity, Temperature, and Depth

Eh Reduction Potential

OBS Optical Backscatter

Software

GMT Generic Mapping Tools

HDF Hierarchical Data Format

KML Keyhole Markup Language

UDP User Datagram Protocol

XML Extensible Markup Language

Compression and Communication

BWT Burrows-Wheeler Transform

CCL Compact Control Language

DCT Discrete Cosine Transform

DWT Discrete Wavelet Transform

FH-FSK Frequency Hopping Frequency Shift Keying

JPEG Joint Photographic Experts Group

LZW Lempel-Ziv-Welch

MAC Media Access Control

PSK Phase Shift Keying

RLE Run-Length Encoding

SPIHT Set Partitioning in Hierarchical Trees

TDMA Time Division Multiple Access

Places

MIT Massachusetts Institute of Technology

WHOI Woods Hole Oceanographic Institution

SMAR Southern Mid-Atlantic Ridge

AGAVE Arctic Gakkel Vents Expedition

CHAPTER 1

Introduction

> If I am to speak ten minutes, I need a week for preparation; if fifteen minutes, three days; if half an hour, two days; if an hour, I am ready now. – Woodrow Wilson

The applications of underwater robotics within the field of Oceanography are as varied as the undersea environment itself. Robots have helped locate hydrothermal vents[72] and characterized the water column through chemical surveys[12, 13], performed careful photographic surveys to document coral reef health[3] or archaelogical sites[21], and allowed biologists to perform animal population censuses[60] in otherwise hard to characterize environments. In each of these tasks the robot's purpose is to acquire data for scientists, who

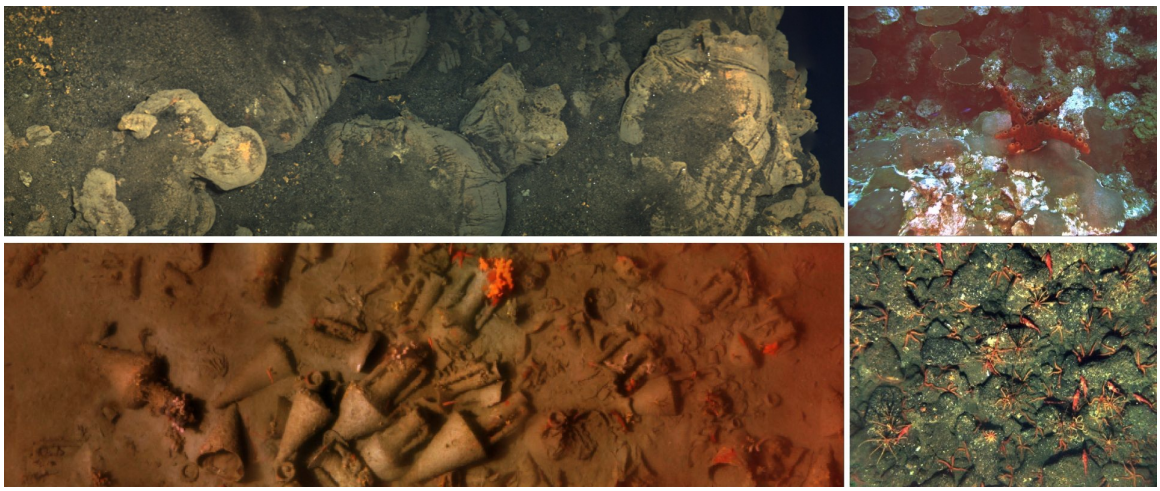


Figure 1-1: Seafloor photos captured by underwater vehicles. Clockwise from top right; coral reef characterization off of Puerto Rico[3], groundfish population census off the United States' Pacific Northwest [60], archaeological survey off of Chios, Greece[21], and the Arctic seafloor[51].

can then form new hypotheses from observations and develop plans for further exploration. This data may take the form of imagery, like the seafloor photographs in Figure 1-1, or a time series of sensor data as shown in Figure 1-2. Tethered robots are able to communicate

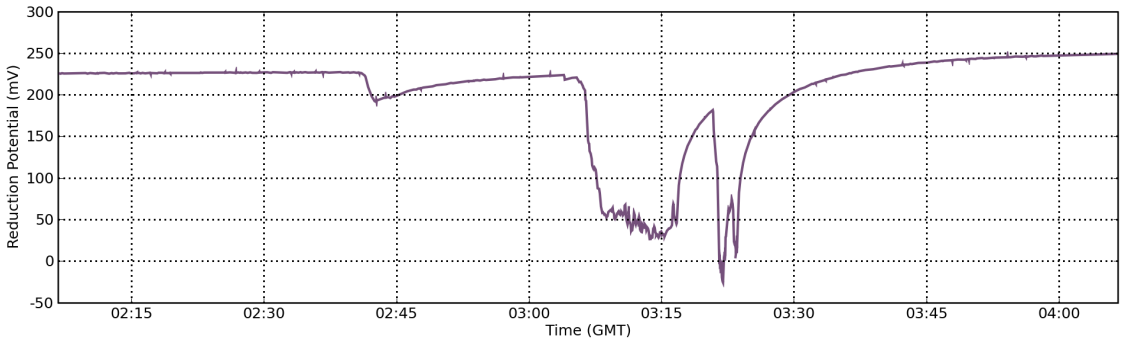


Figure 1-2: Time series data from the Reduction Potential (Eh) sensor mounted on the Puma AUV acquired during a hydrothermal plume survey on the Southern Mid-Atlantic Ridge. Eh sensor and data generously provided by Dr. Ko-ichi Nakamura[37]

this data to surface observers instantly via a cable; for robots without a physical tether to the ocean's surface, communicating discoveries is much more complicated. The ocean imposes severe limitations on wireless communications; AUV and surface ship noise combine with environmental conditions such as seafloor makeup and water depth to cause a host of problems including frequent packet loss, high latency, and low effective bandwidth. Data rates for acoustic modems, used to communicate underwater, are routinely as low as tens of bits per second. For scale, the text of this paragraph is over twelve thousand bits in size.

Seabed-class Autonomous Underwater Vehicle (AUV)'s, originally designed at Woods Hole Oceanographic Institution (WHOI) for photographic surveys[49], have now performed each of the missions described above and continue to be applied to novel fields and problems. Data transmission rates currently limit vehicle telemetry to vehicle state and health information. If science data is included at all, it is included in an ad-hoc mission-specific way. As underwater robotics moves toward the study of dynamic processes with multiple vehicles, there is an increasing need to summarize large volumes of data from the vehicles and deliver it quickly to human operators.

In this thesis, I present a system designed for communicating and presenting science data from untethered underwater vehicles to surface observers. The system's goals are threefold: to aid human operators in understanding oceanographic processes, to enable human operators to play a role in adaptively responding to mission-specific data, and to accelerate mission planning from one vehicle dive to the next. The system uses standard lossy compression techniques extensively to lower required data rates to those supported by commercially available acoustic modems.

Chapter 2 provides background on the problem, while reviewing the previous work that influenced and inspired the design of the system. Chapter 3, describes the types of data which would be desirable to telemeter, and presents sample data from recent AUV dives which will be used as examples throughout the thesis. Chapter 4 focuses in depth on compression of telemetry. Within the chapter, a method for summarizing scalar time-series science data and vehicle state information using lossy compression is presented, before science data from deep (4000 meter) AUV hydrothermal plume surveys on the Southern Mid-Atlantic Ridge (SMAR) is compressed using the presented methods. After touching on the applicability of scalar compression methods to underwater photographs, methods for low bitrate image compression are compared, and evaluated against images captured during shallow-water (80 meter) AUV photo surveys off Puerto Rico. Chapter 5 presents a novel user interface for reviewing transmitted telemetry. The interface is usable with all major operating systems, and integrates with a number of existing geodetic applications. Finally, Chapter 6 concludes that lossy compression methods can be applied to deliver useful science telemetry over extremely low bandwidth connections, and that it can be presented to surface observers in a way that will enable new interactions with, and applications for, underwater robotics.

CHAPTER 2

Background

2.1 Underwater Vehicles

Underwater vehicles broadly fall into two categories; those like Remotely Operated Vehicles (ROVs) which possess a physical cable tethering them to a support ship, and those like Autonomous Underwater Vehicles (AUVs) which do not. Tethers allow underwater vehicles to transmit data to waiting scientists on the surface, and receive power and commands in return. Contemporary ROVs ranging from shallow-water commercial models to the WHOI JASON II[5, 20] ROV shown in Figure 2-1 enable human operators to explore the depths almost as if they were there, by operating a joystick and watching computer monitors. Scientists and engineers can observe imagery, video, and science data in realtime as it is transmitted to the surface through the physical tether. As this data is received, scientists can begin to form hypotheses from new observations, develop plans for upcoming dives, and even alter the plan for what remains of the current dive.

Figure 2-1: Seabotix 150 and JASON II ROVs (JASON II photo courtesy L. L. Whitcomb)

This same tether puts a number of limitations on the capabilities of a robot. Bound to the surface ship, an ROV cannot operate any distance further from the ship than the length

of cable available. This means that the surface ship cannot leave the ROV for any reason, which may hamper or preclude operation in environments where ship motion is limited. For deep-sea ROVs the logistical requirements of managing a 10 kilometer tether are not trivial – large winches and high voltages mixed with a rocking ship and salt water are dangerous, and require a large team of trained professionals to manage safely. While not inherently a restriction of the tether, ROVs must constantly have a human operator available, as ROVs typically possess minimal innate control logic.

Figure 2-2: The Seabed-class Puma AUV

AUVs in contrast, such as Puma shown in Figure 2-2, require no physical surface tether. This allows AUVs to reach areas that are inaccessible to ROVs, such as under Arctic ice sheets[6]. AUVs can be left for hours, or even days, before recovery with a surface ship, and perform navigation tasks without oversight. AUVs, however, are often preprogrammed; missions are typically described in a high level language as a set of waypoints and leave few, if any, mission planning decisions to the vehicles control systems. As Christopher von Alt noted in a 2003 whitepaper,

> in general [AUVs do not] use sensor data obtained during a mission to make them more successful and/or reliable. Sensor information is recorded. It is not processed and used to provide the vehicle with the ability to adapt, and change its current objective; it is simply recorded for future analysis.[64]

If AUVs were capable of communicating science telemetry to scientists on the surface, or remotely available via telepresence, they could benefit from the experience of surface

operators or allow mission goals to be adapted while still underwater like an ROV.

2.2 Underwater Communications

While typical surface or air-based robots might communicate using high-frequency electromagnetic signalling, such as radio modems or 802.11 "WiFi", electromagnetic radiation is quickly dispersed by water. The ocean environment presents numerous challenges for acoustic communication, including low available bandwidth and large propagation delays[52, 1]. These challenges are made worse by operating over long distances[53] and by environmental conditions such as seafloor makeup and water depth. AUV and surface ship noise transmit directly into the channel, further exacerbating the problem. As a result, the use of long-range underwater communication is characterized by extremely low effective bandwidth, high latency, and frequent packet loss. Urick provides one approximate formula for calculating acoustic transmission losses due to absorption and spherical spreading – a nomogram[61] based on the formula is shown in Figure 2-3.

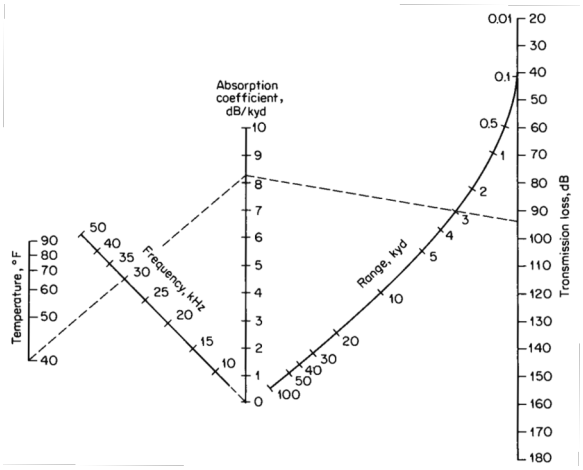


Figure 2-3: Acoustic transmission loss due to spherical spreading and absorption. The dashed line indicates how to use the plot; at $40°F$ and 30 kHz, the transmission loss over 3000 yards is 94 dB. Figure and caption taken from [61].

To accomodate the peculiarities of the medium, channel coding methods with high rates of error-correction are typically employed. While underwater acoustic communications has achieved rates up to hundreds of kilobits per second [52], maintaining reliable acoustic communications over long distances and a diverse set of environmental conditions currently requires the use of low-rate communications with high error tolerance, such as Frequency

Hopping Frequency Shift Keying (FH-FSK)[41] or highly error-corrected Phase Shift Keying (PSK)[41]. In addition, AUVs may rely on acoustic navigation schemes such as Long Baseline (LBL)[36] or Ultra-Short Baseline (USBL). Since the ocean is a shared broadcast medium, time-multiplexing of the channel for navigation, or communication with other vehicles, lowers effective bit-rates further.

Some underwater vehicles operate with an "acoustic tether" to obtain much higher bitrates than are typically available with acoustic communications. In this mode, a one-way acoustic link replaces the physical cable of an ROV. To increase the available bitrate, the surface ship must carefully track the vehicle and remain directly above it. The Hugin AUV[31] has successfully employed this strategy on a number of deep-water missions. In the case of the Hugin AUV, the one-way acoustic data link is capable of transmitting at 1400 bits per second. It is complemented by a second bidirectional communications link, designed to be more resistant to errors, which operates at 55 bits per second[31].

The new Hybrid ROV being developed at WHOI, Nereus[8], has the potential to deliver data to surface operators at very high rates by using a single strand of fiber-optic cable to complement acoustic communication. The Nereus vehicle will have an acoustic modem for backup communications, or for while operating in an untethered AUV mode. This development has great potential for communicating with underwater vehicles, but if the fiber-optic strand breaks, the vehicle must fall back on acoustic communications. For now acoustic modems provide the only wireless, long-distance underwater communications method.

Hardware

AUVs are limited by power constraints, as all power must be brought down with the AUV or generated underwater. Most AUVs choose to obtain power from large on-board battery packs. Acoustic modems designed for AUVs should therefore ideally use minimal power. Commercial modems are available from Teledyne-Benthos[7] and LinkQuest[34]. Seabed-class AUVs use the WHOI Micro-Modem[22] (shown in Figure 2-4) for communication and navigation[50]. To minimize power usage, the Micro-Modem has fixed firmware and functionality; this allows it to use only 10 Watts while transmitting and only 0.08 Watts while receiving[24]. While at sea, it is crucial that telemetry can be adapted to meet potentially changing needs of specific missions. Thus, software-implemented encoding solutions are often pursued instead of modifying lower-level modem processing.

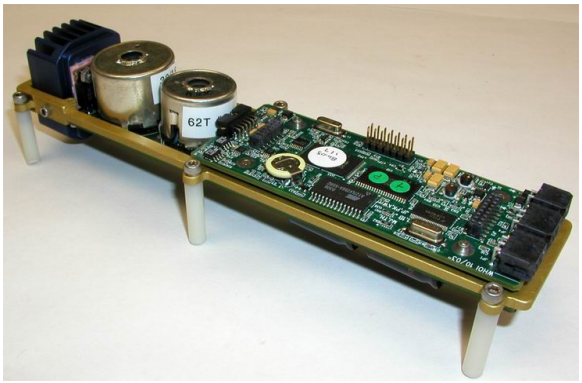


Figure 2-4: The WHOI Micro-Modem and Power Amplifier, Photo by WHOI Acoustic Communications Group

The WHOI Micro-Modem provides Media Access Control (MAC), and uses low frequency bandwidth to allow for multiple senders and receivers. It is capable of sending one 256-bit FH-FSK packet in slightly over 3 seconds, or one 1536-bit error-tolerant PSK packet in slightly over 6 seconds, delivering an effective bit-rate between 80 and 256 bits per second. Commercially available options from Teledyne-Benthos and LinkQuest advertise 80-360 bits per second for environments with harsh multi-path. Advances in coding theory bring increased bitrates, but there is always a tradeoff between enhanced reliability and higher bitrates – with oceanographic vehicles, reliability usually wins. Summarizing data for transmission at such low bit-rates, especially when time-multiplexed with acoustic navigation methods, presents a significant hurdle. As a result, current telemetry is often quite limited.

2.3 Scalar Time-Series Telemetry

During current deployments of Seabed-class[49] AUVs, a surface operator monitors simple vehicle telemetry to track the AUV and watch for any indication of a problem. Seabed-class and Remote Environmental Monitoring UnitS (REMUS) AUVs currently make use of the Compact Control Language (CCL)[54] for telemetry, which was developed at WHOI to meet the needs of an AUV.

CCL provides a data-formatting standard for AUV to AUV, and AUV to surface-ship, communications. The standard describes a number of 256-bit packets which can be used for file transfer or for transmitting vehicle state, salinity data, bathymetry estimates, and other

oceanographic data[55]. Each of these packets is designed to be self-contained; for example, the MDAT_BATHY (bathymetry) packet contains measurements of depth, altitude, latitude and longitude from three distinct locations. No other packets are required to make sense of the data contained within the packet. To provide rudimentary compression, data are requantized to varying levels to fit into an integer number of bytes. While CCL is adequate for transmitting individual datapoints from an AUV, or for transmitting commands to an AUV, it is not particularly efficient. CCL was not designed to take advantage of advanced compression, and it makes no use of the inherent correlation between successive instrument samples. CCL is also only designed to work with packets of exactly 256 bits in length.

2.4 Photo Telemetry

Low bitrates have inhibited attempts to transmit photo telemetry from untethered vehicles but there has been some research on transmission of photos over higher bandwidth acoustic tethers. In 1992, researchers from NEC presented a system for transmitting low-resolution compressed images from the Shinkai 6500 submersible[59]. Researchers at WHOI have developed high speed prototype acoustic tethers capable of transmitting video[40]. In addition, Hoag, Ingle et al. have extensively studied the application of wavelet compression techniques to underwater images[25] and video sequences[26].

Craig Sayers, and others at the University of Pennsylvania, developed techniques for selecting specific frames and 'regions of interest' from a video sequence that best describe an ROV manipulator and environment state, and transmitted these regions to surface operators over a 10000 bit per second acoustic tether as JPEG images[47].

2.5 User Interfaces

There has been substantial previous work on user interfaces for ROV and Human Occupied Vehicle (HOV) telemetry. Georeferenced tools for AUV's serve two primary purposes: monitoring the AUV while it is underwater, and analyzing science and engineering data after AUV recovery. In the first category, there are a number of utilities. Operators of REMUS AUVs can make use of the REMUS Vehicle Interface Program, which tracks vehicle state and allows simple commands to be sent to the AUV while a mission is underway. Seabed-class vehicles make use of a topside monitoring program written in MATLAB while the

AUV is underwater. Screenshots from REMUS and Seabed applications are shown in Figure 2-5. DVLNAV, developed by Kinsey and Whitcomb at Johns Hopkins University[30], also provides an extensive graphical interface on top of a navigation and tracking program for underwater vehicles.

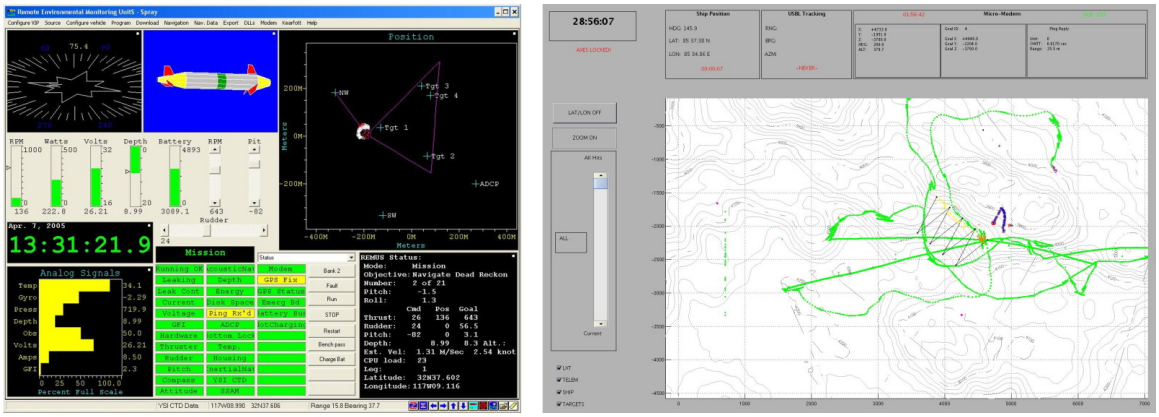


Figure 2-5: Screenshots of (from left) the REMUS Vehicle Interface Program[56] and Seabed Topside software.

After vehicle recovery, GeoZui3D, developed at the University of New Hampshire's Center for Coastal and Ocean Mapping, provides a powerful 3D interface for reviewing captured data [66]. In 2005, a system for real-time 3D monitoring of ROV's and HOV's based on integrating DVLNAV with GeoZui3D was presented[35]. Data from Seabed vehicles can be analyzed using the Seabed-Plot suite of MATLAB tools. GeoZui3D, DVLNAV, and Seabed-Plot are shown in Figure 2-6. Oceanographers also commonly rely on a mish-mash

Figure 2-6: Screenshots of (from left) GeoZui3D[66], DVLNAV[30], and the Seabed-Plot software in use.

of existing plotting tools, such as the open-source Generic Mapping Tools (GMT)[69], the no-cost Google Earth software, commercial GIS packages, and the MATLAB programming environment.

CHAPTER 3

Telemetry Classes

Contemporary AUVs acquire gigabytes of data in each hour of deployment, ranging from simple measurements of scalar science data to multibeam sonar bathymetry and high-resolution digital photographs. Yet, for many underwater vehicles, the estimated location and vehicle health are the only pieces of information available before recovery.

3.1 AUV State Information

Data	**Bits**	**Precision**
Packet type code	8	N/A
Position (X,Y)	48	1 m
Depth	16	0.1 m – 0.5 m
Altitude	16	0.01 m
Heading	8	1.4°
Goal Position (X,Y)	48	1 m
Goal Depth	16	0.1 m – 0.5 m
Goal ID or Error Code	16	N/A
Unused	80	N/A

Table 3.1: Contents of standard Seabed CCL packet. The packet contains vehicle position and health, along with information about the current goal. Recently, unused bits have been adopted for a variety of mission-specific data.

Seabed-class AUVs use a CCL packet containing the vehicle's three dimensional location, current mission goal and health to communicate their state, as shown in Table 3.1. The packet contains the vehicle's current location, and the location of the goal that is currently being pursued. Additionally, the vehicle can indicate if something has gone wrong. Rather than transmitting latitude and longitude, Seabed-class AUV's currently use X and Y coordinates in a local 'AlvinXY' coordinate frame[1] to represent their location. The AUV

[1]The AlvinXY coordinate frame is an equirectangular map projection with a dive-specific local origin –

is configured, using a Time Division Multiple Access (TDMA) cycle, to telemeter packets to the surface ship at regular intervals. Typically each field of the CCL packet contains the most recent sample or estimate for that source.

This strategy is sufficient to provide a surface ship an indication of the AUV's location and goal, yet state telemetery may be strongly aliased due to the low sample rate. Aliasing of vehicle heading is particularly likely, as an AUV without the ability to control heading (such as caused by an inoperational thruster) may spin at a much higher rate than the Nyquist frequency associated with the telemetry. Figure 3-1 shows heading data acquired

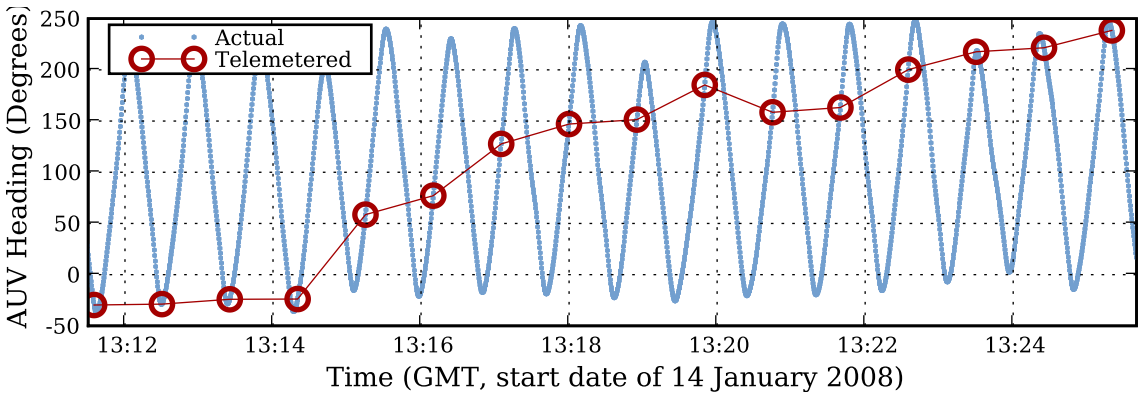


Figure 3-1: Heading sensor aliasing evident during the Arctic Gakkel Vents Expedition. Red circles show what telemetry would be sent to the surface with a 55 second long TDMA cycle, and blue dots show the actual heading. The X axis is in minutes.

during the Arctic Gakkel Vents Expedition (AGAVE)[51] as the Jaguar[32] AUV spiraled to the seafloor – if heading were telemetered every 55 seconds, it would tell a very different story than what is actually occurring.

Furthermore, the low sampling resolution of the telemetry in time may translate directly to an unacceptably low sampling resolution in space. An AUV with a horizontal speed of 0.5 meters per second, telemetering data every sixty seconds, will only provide one sample for every thirty meters of forward travel. While the natural sampling frequency of an instrument may not be achievable, signals may need to be subsampled to meet some other desired criterium. For missions with a desired horizontal spatial sampling resolution, a sampling frequency could be calculated from a nominal or maximum vehicle speed.

Accurate and timely vehicle location and state information can be critical to safe recovery of the vehicle. In complex or dangerous environments, such as AUV missions under

X coordinates are to the East in meters, and Y coordinates are due North in meters.

Arctic ice caps, as shown in Figure 3-2, vehicle location and state information are necessary for safe recovery of the vehicle. If a surface ship has no independent method of obtaining the vehicle's position, such as a shallow-water USBL system, the AUV's latitude, longitude and depth or altitude must be telemetered from the vehicle.

Figure 3-2: The Puma AUV sits in a small clearing in the pack-ice after completing an under-ice mission in the Arctic.

3.1.1 The AGAVE Expedition: Recovery in Complex Environments

Seabed-class AUVs are equipped with the capability to receive new goal positions and depths via the acoustic link for human guidance during complex recoveries. The positions are sent as absolute locations in the AUV's reference frame, allowing the AUV to operate with no knowledge of ship position and orientation. These capabilities enabled the AUVs to be successfully recovered after over ten dives performed under the Arctic ice cap at 85°N latitude during the AGAVE Expedition. When operating in open-water, an AUV can simply return to the surface in the case of a problem or the end of a mission. While operating under Arctic ice caps, such a scenario is no longer feasible. At the completion of a mission, an AUV needs to communicate with the surface ship and be gradually guided back to a safe location before surfacing. This location may be a natural clearing in the sea-ice, or a small pond created by the ship.

The AUV recovery process for a representative mission is shown in Figure 3-3. Throughout the recovery, AUV operators sent a sequence of acoustic commands requesting depth and bearing changes for the AUV to follow. These changes are represented in the figure as

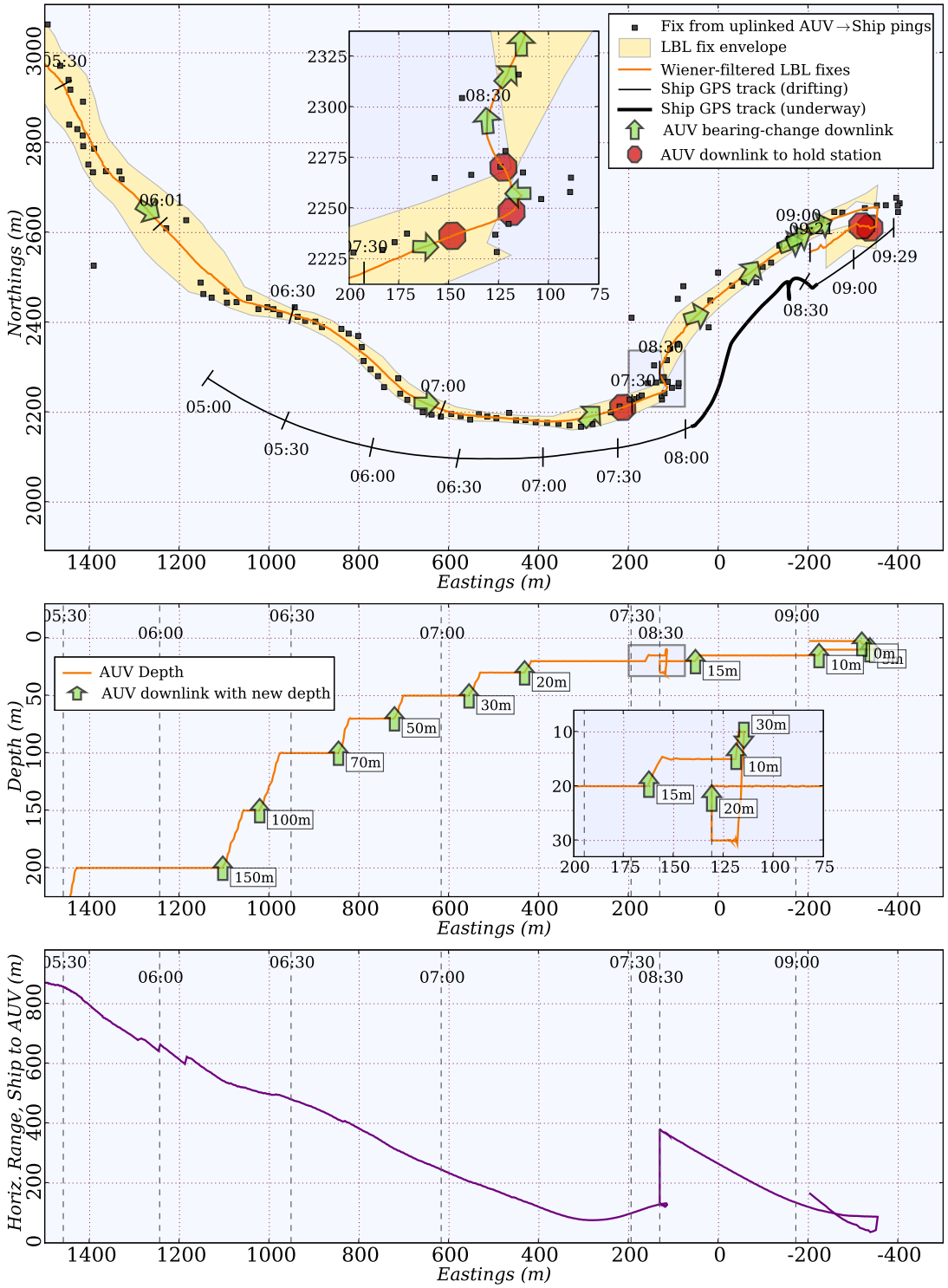


Figure 3-3: These plots provide three different views of the AUV recovery. The top plot shows the ship and AUV tracks, along with error bounds on the AUV position. The middle plot shows the AUV depth throughout the recovery. The bottom plot shows the horizontal range from the ship to the AUV over the course of the dive.

green arrows in the direction of the request, or red octagons where the AUV was ordered to hold a horizontal position. The bearing changes were sent as absolute points for the AUV to drive to; this prevented the AUV from driving forever if communications were impeded. For clarity, commands have been omitted from the figure when they were sent only to extend the destination point, allowing the AUV to continue along the same bearing.

After completing the mission, the AUV rose to a depth of 200 meters. The surface ship was approximately 750 meters from the AUV horizontally, next to an opening in the ice pack where the AUV could surface. Ice drift made recovery even more challenging, as the ship continuously drifted away from the AUV at up to twenty-five centimeters per second. As the AUV drove towards the ship, it telemetered back sub-sampled travel times to two ship-mounted navigation beacons which were used by engineers on board the ship to calculate vehicle locations. These location fixes are displayed in Figure 3-3 as dots; note that the vehicle itself was not capable of computing positions as it did not know the position of the ship. When the AUV reached the ship around 07:45, the recovery opening had become clogged with ice. Clearing the hole of ice left the vehicle and ship briefly out of contact, after which the vehicle was driven directly towards the ship and brought to the surface. A more in-depth description of these Arctic AUV operations can be found in [32].

Telemetering science data has the potential to provide similarly powerful benefits during AUV missions. While progress continues to be made towards higher levels of autonomy in AUVs[11, 72], training an AUV to alter missions in a way that achieves high-level science objectives is a task-dependant and complex problem. Ideally, AUVs could benefit from the domain and application-specific knowledge possessed by scientists and engineers on the surface. Ocean depths reach to thousands of meters, meaning that a round trip to unload data at the surface, even in open water, could mean a delay of ten (or more) hours. While battery packs are often the primary limitation on mission length, enabling an AUV to could communicate science telemetry to surface observers could enable new interactions between vehicles and scientists. Scientists on boad the ship, or remotely available via telepresence, could adapt mission goals based on live observations.

3.2 Time-Series Scalar Data

While the location and state data described above in section 3.1 is one example of a time-series, numerous oceanographic sensors, such as a Reduction Potential (Eh) or Conductivity, Temperature, and Depth (CTD) sensor, provide a time series of one or multiple scalar values as their data product. A temperature sensor operating at 10 Hz with only a single scalar measurement value will produce 19.2 kilobits of 32-bit floating-point data per minute. A listing of the data produced by several scalar sensors used with Seabed-class AUVs is presented in Table 3.2. In addition to the sensor readings themselves, each sample has a

Sensor	Rate	Format
Depth Sensor	0.8 Hz	1x32-bit floating point
OBS Sensor	1 Hz	3x16-bit integers
Eh Sensor	2 Hz	1x32-bit floating point
CTD Sensor	4 Hz	5x32-bit floating point
Magnetometer	10 Hz	4x32-bit floating point

Table 3.2: Data statistics for scalar sensors typically used with Seabed-class AUVs.

timestamp marking the time the reading was taken. Transmitting time information along with the data is necessary for surface observers to co-register location data and science data, and perform geo-referencing.

Throughout this thesis, two sections of sensor data from actual AUV dives will be used to illustrate the relative performance of compression algorithms. One two-hour section of Reduction Potential (Eh) data sampled at 2 Hz was acquired during the 2008 Arctic Gakkel Vents Expedition (AGAVE) Expedition[51]. The second two-hour section of potential temperature data was calculated at 1 Hz from Conductivity, Temperature, and Depth (CTD) data collected on a 2008 cruise to the Southern Mid-Atlantic Ridge (SMAR). The raw data segments are shown in Figure 3-4. The Eh data is quite noise-free, and contains several strong discontinuities along with a few smooth transitions. The potential temperature data, in contrast, is quite noisy. A couple of discontinuities set off short sections of data, but the signal is otherwise relatively constant.

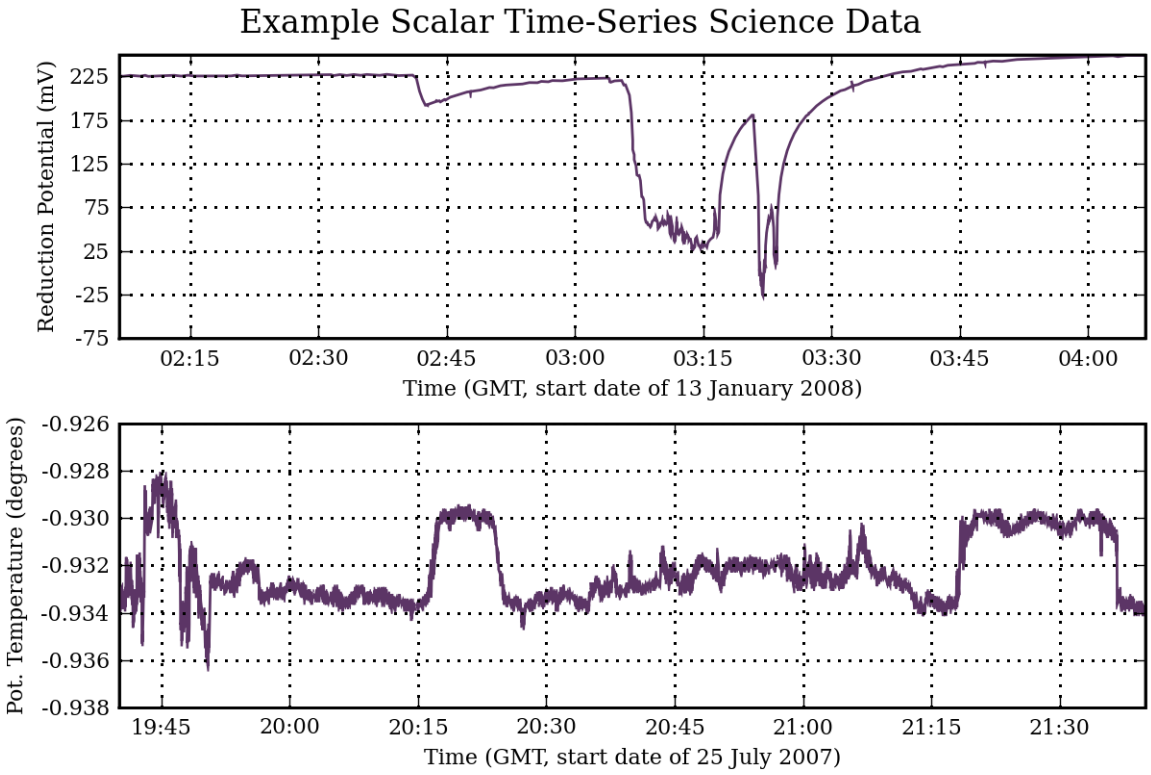


Figure 3-4: Scalar time-series data that will be used in examples throughout this thesis. At top is a two-hour section of Reduction Potential (Eh)[37] data sampled at 2 Hz. At bottom is a two-hour section of potential temperature data calculated at 1 Hz from Conductivity, Temperature, and Depth (CTD) data.

3.3 Imagery

A vehicle capturing a relatively modest 1.5 megapixel, 12-bit color image every 3 seconds will collect tens of gigabytes of raw imagery over the course of only a few hours. Transmitting only a single uncompressed image of that size would take over 10 hours at an optimistic rate of 500 bits per second; continous high-fidelity imagery at these ultra-low bandwidths is simply implausible. However, imagery can play a role in informing surface observers, even in situations with extremely low bandwidth.

3.3.1 Cameras as Scalar Sensors

Rather than considering a camera's output to be a photograph, we can use machine vision techniques to convert a photograph into a scalar value, or set of scalar values. These values may represent something as simple as the brightness or dynamic range of the image, or much

more complicated algorithms to interpret what the image actually shows. The camera then becomes yet another scalar sensor, producing a time series, with a sample rate equal to the frequency at which it takes photos. Any compression technique applicable to scalar sensors can then be used to compress the resulting data.

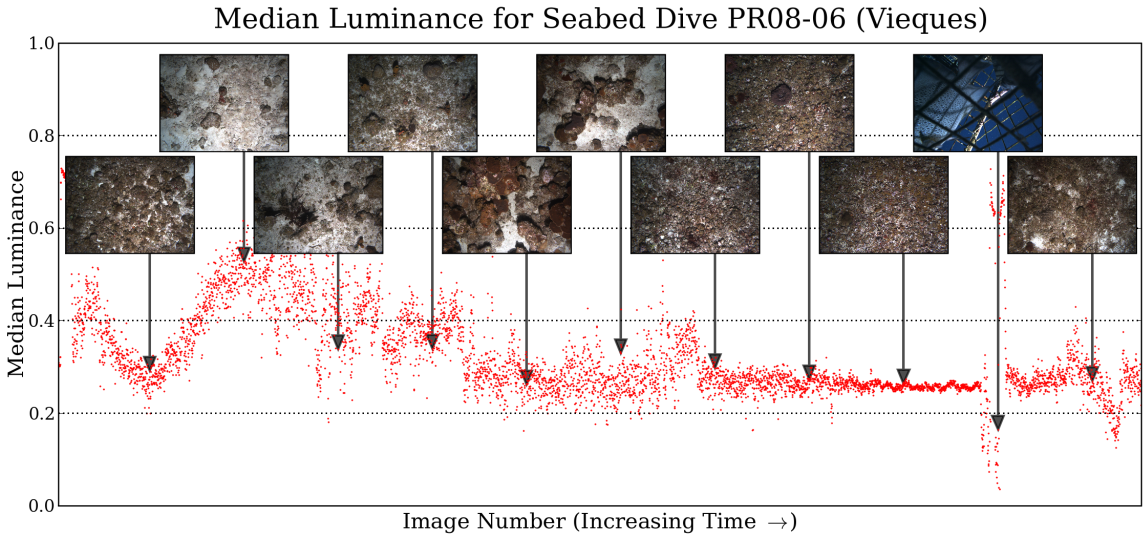


Figure 3-5: An example of using a camera as a scalar sensor. Dots indicate the median luminance of a single captured image.

Figure 3-5 shows an example of using a camera as a scalar sensor, based on a series of seafloor photos captured near Vieques, Puerto Rico. The median luminance for each image has been calculated, generating a time series. Machine vision techniques have already been applied to a number of oceanographic problems. For surveys of coral reefs, biologists would like to know the health and percentage cover for each coral species in each image[3]. Techniques based on color segmentation[17], image morphology[29] and support vector machines[28] have already been applied to this task in post-processing. When performing fish and habitat surveys, fishery employees would like to have a count of different fish species, and would like to know the boundaries for different types of seafloor terrain. Texture analysis methods could likely be applied to determing seafloor composition, and estimates of seafloor rugosity could be obtained using existing techniques for determining roughness from photographs[57].

We note, then, that to have a camera be a useful source for telemetry does not necessarily entail transmitting images. However, it is beyond the scope of this thesis to examine each of these application-specific algorithms in depth. In essence, machine vision techniques provide

one avenue of trading off processing time on the underwater vehicle's computer against the amount of bandwidth necessary for telemetry. Transmission of summary statistics is much easier than transmission of entire images. In addition, these summary statistics can be far easier to review for surface observers. Reviewing several thousand photos will take hours, but review of a time series plot consisting of several thousand points can be performed in seconds.

For some applications, photos would provide surface observers with data that simply can't be provided as scalar telemetry. This may be the case when the appearance of a target is not known (such as locating an archaelogical site), because existing machine vision algorithms are not capable of providing a useful scalar representation of the data, or due to computational limits on the AUV. Photos also provide powerful confirmation when the results of scalar telemetry are difficult to understand.

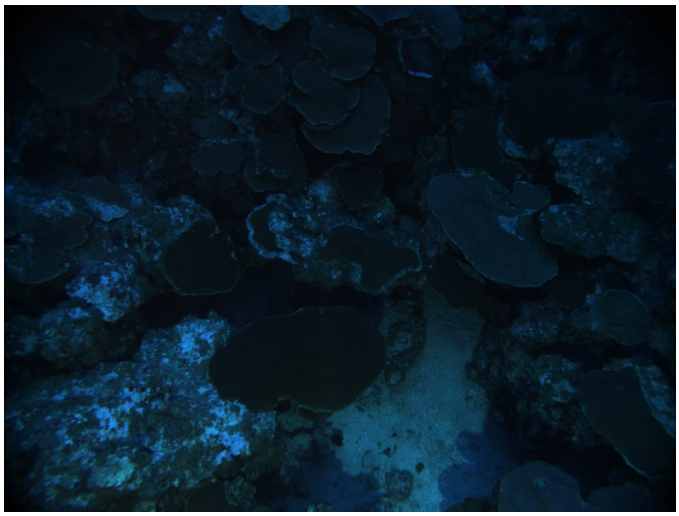


Figure 3-6: Underwater images are characterized by a strong blue cast, and poor contrast.

Photos captured underwater are characterized by highly repetitive textures, like sand and coral surfaces, poor contrast, and uneven lighting. A raw underwater image is shown in Figure 3-6. Seawater quickly absorbs visible light, and little light filters to the seafloor. Restrictive power budgets can limit a vehicle's ability to light the seafloor, which lead to poor contrast and strongly defined cast shadows. Cameras with high dynamic range are used to obtain enough color information to reproduce the environment, yet images still have a strong blue cast as seen above in Figure 3-6. Throughout this thesis, three images captured during shallow-water coral reef surveys off Puerto Rico in 2008, shown in Figure 3-

7, will be used to illustrate compression techniques applicable at very low bitrates. All three photos were captured on the same AUV dive, and all exhibit the common characteristics described above. While all three images are typical in many ways, they contain details which differentiate them. The first image is fairly uniform and low-contrast, yet contains some organic details. The second image has a predominant orange sponge in the corner, and the third image has a couple of gorgonians containing lots of fine detail. The sample images used in this thesis were corrected with a histogram equalization. While histogram equalization will not correct for cast shadows or unequal lighting, it will often balance color well enough for visualization and is simple enough to be performed on power-efficient embedded processors.

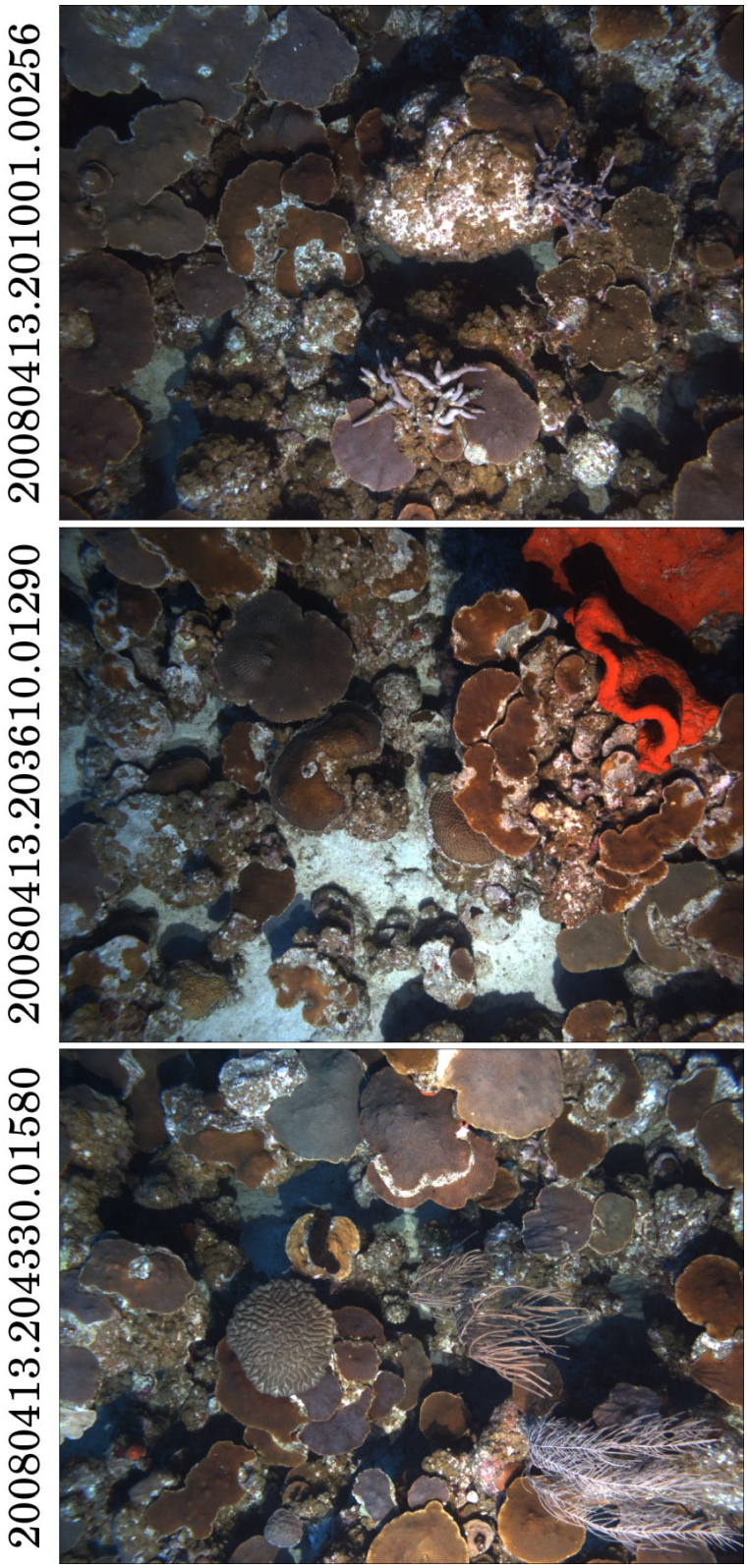


Figure 3-7: Three sample coral reef images captured during shallow-water coral reef surveys off Puerto Rico in 2008. These images possess a number of characteristics common to underwater images.

CHAPTER 4

Telemetry Compression

> . . . we cease to notice the annoying hubbub of surrounding conversations, but a sudden silence reminds us of the presence of neighbors. Our attention is clearly attracted by transients and movements as opposed to stationary stimuli, which we soon ignore. — Stéphane Mallat, *A Wavelet Tour of Signal Processing*

Providing all of the data described in Chapter 3 at full resolution would require orders of magnitude more bandwidth than is currently available. As a result, most of this data remains unavailable until the AUV returns to the surface after a dive. Compression techniques can take two forms; lossless compression, which allows faithful reconstruction of the original data, and lossy compression, which does not. In this chapter we will investigate both methods for compressing time-series science data along with methods for compressing color images.

4.1 Time-Series Scalar Data

Beyond subsampling the data, one of the simplest ways to compress scalar data is to calculate and telemeter summary statistics. This strategy was used during a 2008 research cruise to the Southern Mid-Atlantic Ridge (SMAR) to telemeter science data to the surface while the AUV continued its tracklines. The "Unused" bits in the MDAT_STATEXY CCL packet (shown previously in Table 3.1) were allocated to transmitting summarized Optical Backscatter (OBS) and Reduction Potential (Eh) science data. OBS data was median filtered to remove noise, and anti-aliased before being telemetered. Eh data acquired since the last telemetry packet was low-pass filtered with a Butterworth filter, differentiated, and the variance calculated[1]. Unfortunately, complex summary statistics such as these can be difficult to interpret in familiar units and terms. To truly allow scientists a clear understanding

[1] Unpublished work by Michael Jakuba and the author.

of what the AUV is sensing, time-series plots and geo-referenced plots at high sampling rates are necessary. Ideally, scientists should be presented with data at high enough resolution that the full-resolution data can be used for confirmation if the AUV is recovered, but is not necessary for decision-making. To provide such time-series data, we look to more elaborate forms of compression.

4.1.1 Methods for Lossless Compression

There exist an entire alphabet of general-purpose lossless compression algorithms, such as LZ77[73], Deflate[15], Lempel-Ziv-Welch (LZW)[68], and the Burrows-Wheeler Transform (BWT)[9]. These algorithms form the basis for a number of generic compression utilities; *gzip*, *winzip*, and *pkzip* utilities make use of the Deflate algorithm, while *bzip2* uses the BWT algorithm. As early as 1996, Eastwood et al. looked at the efficacy of these techniques, and proposed a couple of heuristics for use when transmitting data over an acoustic modem[19]. While these algorithms perform well with plain text and other widely used document formats, they do not perform particularly well on floating-point scientific data. Welch addresses this directly in his notes on the performance of LZW[68], saying:

> Arrays of floating point numbers look pretty much like white noise and so they compress rather poorly. The fractional part is a nearly random bit pattern. ... Some floating point arrays expand by 10 percent going through the compression algorithm, when the exponents vary widely.

Compression Method	**Reduction Potential**		**Potential Temperature**	
	Size (Bytes)	**Ratio**	**Size (Bytes)**	**Ratio**
Raw data	115200	—	57208	—
gzip	29770	387.0%	38173	149.9%
bzip2	24300	474.1%	40318	141.9%
FPZip[33]	77460	148.7%	35475	161.3%
FPCompress[10]	88345	130.4%	41820	136.8%

Table 4.1: Comparison of compression ratios for selected lossless compression methods. FPCompress and FPZip were performed using source code from the authors' websites.

As a result, special-purpose approaches designed specifically for compressing floating-point scientific data have been developed[10, 33, 43]. However, even the most recent lossless

algorithms yield on the order of 2:1 compression for high entropy time-series science data[10]. Compressing the example time-series data using each of the methods mentioned above actually resulted in significantly better compression ratios from the general-purpose algorithms in one case than from the floating-point specific algorithms, as shown in Table 4.1.

The goal of low-bandwidth AUV telemetry is to supply observers with a rough sketch of the AUV's state and environment in as many modes as possible, not necessarily to provide observers with full-resolution science results. Since current lossless compression techniques cannot provide a huge storage benefit beyond sending raw data, lossy techniques seem better suited to achieving this goal.

4.1.2 Introduction to Lossy Compression Techniques

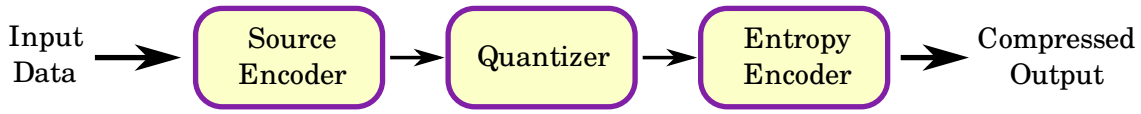


Figure 4-1: System diagram for a simple, standard, lossy signal encoder. Data is first transformed with a source encoder, then quantized, then entropy coding is applied to the result. Graphic adapted from [44].

Most lossy data compression schemes use a similar pattern to obtain reasonable compression levels, as shown in Figure 4-1. First, data is encoded into a new domain by a source encoder, using methods such as the Discrete Cosine Transform (DCT) or Discrete Wavelet Transform (DWT), where it is believed to have a more easily compressed representation. Next, the resultant data is scalar or vector quantized before being re-encoded with some form of entropy encoding [44].

4.1.3 Source Encoders: The DCT and DWT

Source encoders take advantage of the inherent correlation within a signal to concentrate the signal's energy into a smaller representation than the original signal. A sinusoid, for example, is shorter to represent as a single magnitude and phase in frequency space than as an entire series of discrete-time samples. The values output by an efficient source encoder will no longer be correlated across different input sequences, as the coefficients could then be compressed further[46]. Source encoders are not necessarily lossy; neither the DCT nor

the DWT described in this chapter are inherently lossy. However, since source encoders concentrate most of the energy of the original signal into a smaller number of coefficients, an approximate reproduction of the input signal can be reconstructed from this smaller representation. The DCT and DWT, both linear transforms, are widely used as source encoders for both time-series and image data. While both are described below, readers are encouraged to consult "Wavelets, Approximation and Compression" by Vetterli[63], in which a variety of linear and non-linear compression methods are explained in a clear and accessible manner.

Discrete Cosine Transform (DCT)

The DCT transforms a signal into a representation consisting of a sum of scaled cosines of fixed frequency and phase. This is similar in many ways to the well-known Fourier transform, yet the cosine basis functions of the DCT differ by having no variable phase offset. A discretely sampled signal L samples in length is represented by a sum of L scaled cosines. The equation for the k-th basis cosine of the DCT for a signal of length L is provided in Equation 4.1[58], and cosines representing the first four components in a DCT-transformed signal of sixteen samples are shown in Figure 4-2.

$$C_k^{II}(n) = b(k)\sqrt{\frac{2}{L}}\cos\left(\frac{\pi k(n+\frac{1}{2})}{L}\right), \qquad b(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } k = 0, \\ 1 & \text{otherwise.} \end{cases} \tag{4.1}$$

Note that the first 'cosine' has an infinite period, and represents the DC offset of the

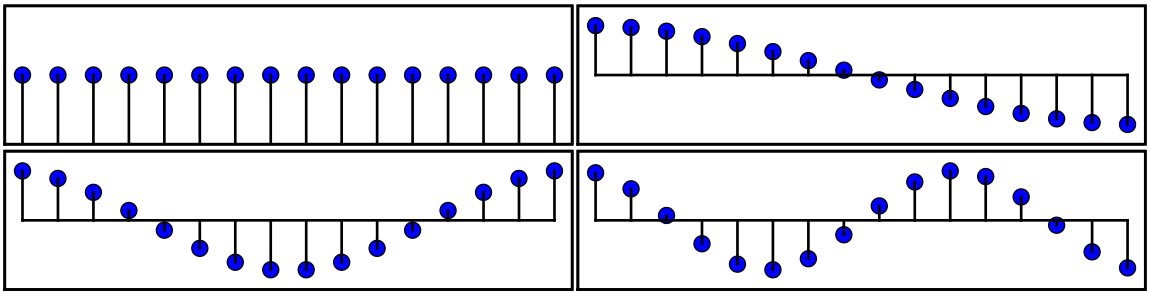


Figure 4-2: The first four basis cosines of the DCT for a discrete signal with sixteen samples.

signal. The details of performing the DCT or its dual, the inverse DCT, are best left to other resources; Strang[58] and Oppenheim[39] both provide solid introductions to the topic.

A simple approximation to the original signal can be obtained by discarding the DCT coefficients representing higher-frequency cosines. After those coefficients are discarded, the signal can be reconstructed from the remaining coefficients, resulting in an approximation with high frequency details removed More advanced DCT-based methods, such as those employed by the JPEG format, include all coefficients but apply higher levels of quantization to coefficients for higher frequency cosines before entropy coding is performed[65].

Discrete Wavelet Transform (DWT)

The basis function of the DWT is the 'Wavelet', a short waveform which dies out quickly in both directions. A signal can be approximated from a subset of its wavelet coefficients, much like with a DCT. However, when a signal is reconstructed from its DCT representation, the cosines used as basis functions contribute to signal reconstruction at each point in time. Wavelets, in contrast, can have *compact support* – then they are identically zero outside of some time window. This means that removing a single coefficient in the DWT transformed version of a signal will not effect the entire signal, only a section of it. For our purposes this is helpful, as coefficients can be selected to emphasize areas with changes rather than improve the overall accuracy of the entire signal.

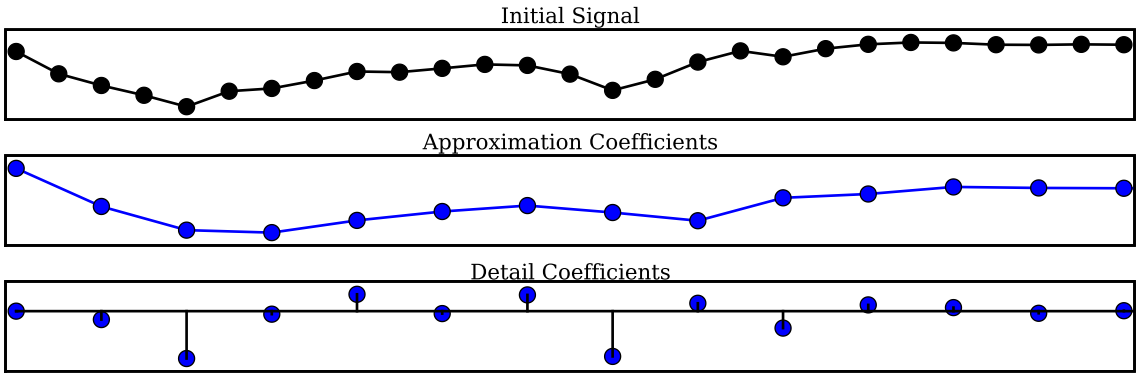


Figure 4-3: Example DWT transformation of a simple signal.

The DWT is calculated by applying a low-pass filter to the input signal, generating one set of coefficients, and then applying a high-pass filter to the input signal to generate a second set of coefficients. Both sets of coefficients are then downsampled by two, resulting in the same number of values as the original input. Calculating the DWT of a signal thus results in two distinct sets of coefficients; a decimated version of the signal known as

the 'approximation coefficients', and a set of 'detail coefficients' which contain the higher-frequency information lost during decimation. A simple signal, and both sets of coefficients from performing the DWT on it, are shown in Figure 4-3. The DWT is typically applied iteratively to the approximation coefficients to generate several levels of detail coefficients; each level of detail coefficients then represents the detail lost by decimation at that iteration of the transform. Each detail coefficient in the resulting set is therefore localized in time as well as being associated with a 'scale', or level of detail. For a well-written and more formal introduction to wavelets, DeVore and Lucier provide an excellent reference[16].

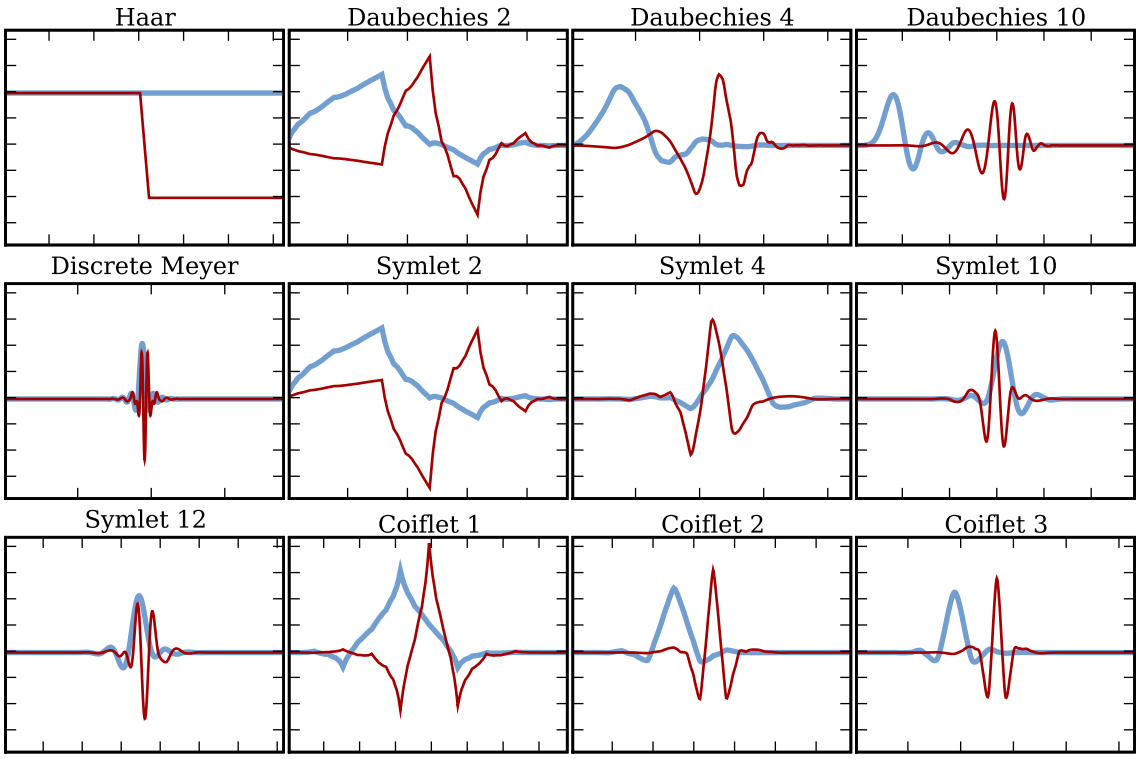


Figure 4-4: Plots of low-frequency (thick blue line) and high-frequency (thin red line) filter functions in the time domain for a variety of wavelets. The low-frequency filter is responsible for generating approximation coefficients, and the high-frequency filter for generating detail coefficients. Figure generated with PyWavelets[67].

There are a number of paired filters which can be applied during the two filtering stages. These filters are grouped into 'wavelet families', within which there are functions of different support length. Wavelets with longer support will result in smoother approximations, but also limit the number of times that the DWT can be iteratively performed. A wavelet decomposition using a filter with longer support will result in more approximation coefficients

than a decomposition with a more compact filter. Sample wavelets of a few families and support lengths are shown in Figure 4-4. The filters in the thicker light blue line are used to generate low-frequency approximation coefficients, whereas the filter in the thinner red line generates high-frequency detail coefficients.

An approximation to the original signal can be obtained by inverse transforming all of the approximation coefficients and a subset of the detail coefficients, selected on the basis of having the largest magnitude. Since wavelets are localized in both time and scale, wavelet coefficients will be larger near discontinuities and abrupt changes. This method of wavelet compression is described by Donoho et al. as being especially appropriate for functions that are "piecewise-smooth away from discontinuities"[18]. While not all sensors emit signals of this form, this is an apt description for some oceanographic sensors, such as an AUV's Eh or OBS sensor as it comes upon a feature of interest.

DCT and DWT Comparison

The time-series science data initially presented in Figure 3-4 was transformed with the DCT and the DWT (using the *Daubechies 2* Wavelet), and increasing numbers of coefficients were used to reconstruct the data segments. Figure 4-5 shows the results of reconstruction based on the subset of coefficients; the number of coefficients for the DWT transformed signal includes the count of approximation coefficients and detail coefficients. Being able to reconstruct the signal with fewer coefficients means that there are fewer numbers to be transmitted to the surface, and improves the resulting compression.

With only twelve coefficients the wavelet-compressed version shows two distinct dips in the value of the Eh sensor and a better signal-to-noise ratio than the DCT-compressed version, yet has artifacts which could easily be misinterpreted by surface observers. With twenty-five coefficients, the magnitudes of the dips are approximately correct, and most artifacts are gone. By fifty coefficients, the DWT version of the Eh signal has been replicated fairly accurately, yet the DCT version still doesn't even capture the two close dips in Eh. Similar results are visible in the potential temperature signal, where high-frequency details are captured by th DWT-compressed version, but simply not replicated in the DCT-compressed version. The Mean Square Error and Signal-to-Noise Ratio error metrics are shown in the plots for comparison. Provided enough coefficients are transmitted to minimize artifacts, possibly by using one of the metrics above as a guide, the DWT-compressed version

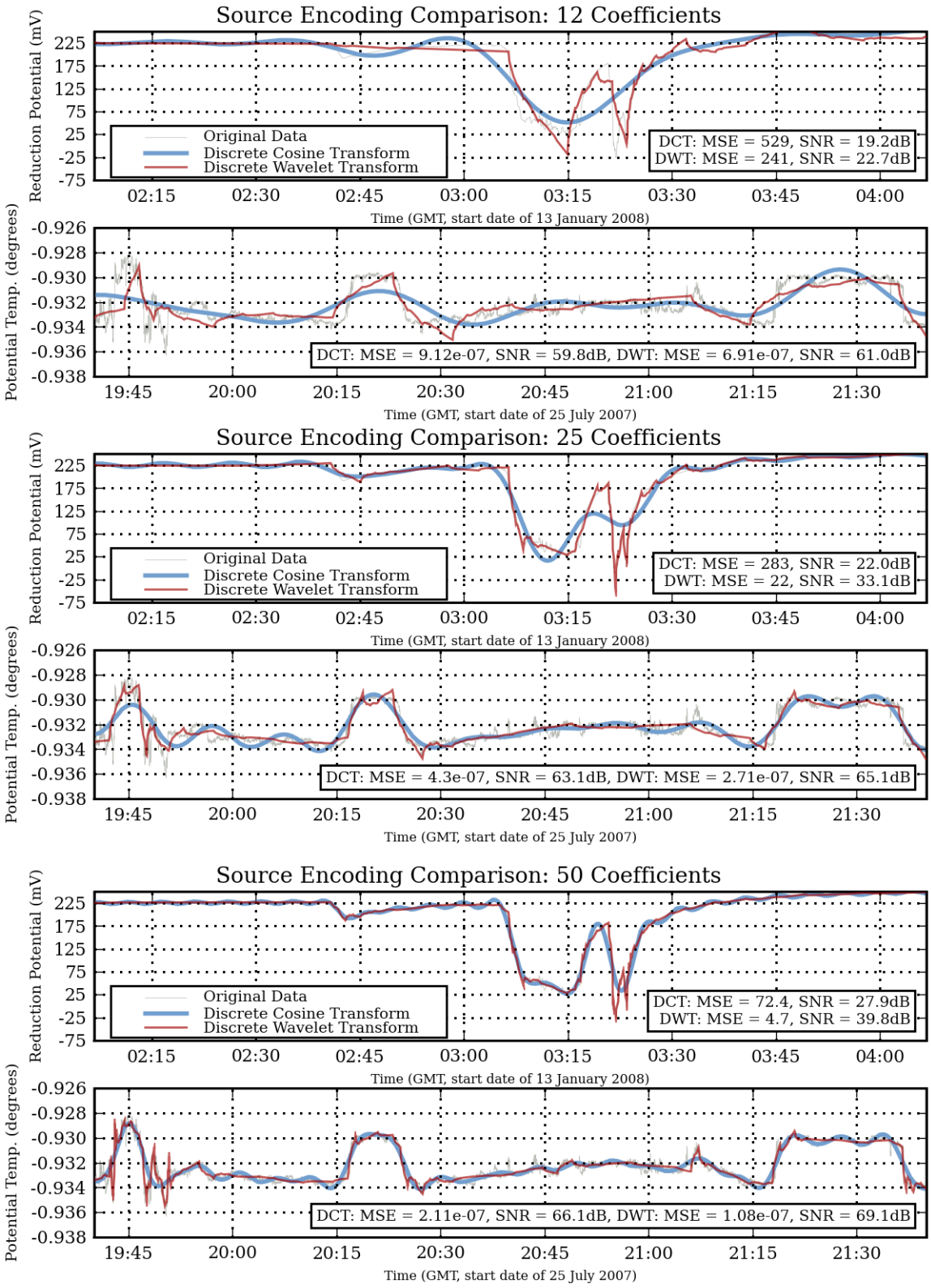


Figure 4-5: Science data reconstructed from increasing numbers of DCT and DWT coefficients, with Mean Square Error (MSE) and Signal-to-Noise Ratio (SNR) error metrics.

shows that a high fidelity representation of the signal can be transmitted in much smaller space than the original signal. For a measure of the DWT performance in terms of bits, continue to section 4.1.5.

4.1.4 Quantization, Entropy Coding, and Packet Forming

To develop a method for compressing time-series oceanographic data, we build on the promising results of the DWT reconstruction obtained above. First, the DWT is iteratively applied to the time-series data, resulting in a multilevel wavelet decomposition. The appropriate wavelet should be selected while considering target packet sizes; for packets of only tens of bytes, an wavelet with extremely compact support (such as the *Haar* wavelet, or *Daubechies 2* wavelet) should be used. When bitrate constraints allow larger packets, a wavelet with larger support can be used to obtain smoother results. To show the results obtainable with small packets, the examples in this thesis use the *Daubechies 2* wavelet – this is partially responsible for the jagged artifacts visible in the wavelet-compressed telemetry.

The result of the iterative wavelet decomposition is therefore a small number of approximation coefficients, and a large number of detail coefficients. For smooth time-series data with few discontinuities, the detail coefficients will be low in magnitude, with interspersed large magnitude coefficients near discontinuities. This sparsity allows us to efficiently compress the data during quantization and entropy coding. To reconstruct a lossy version of the signal, all of the approximation coefficients must be available, but low magnitude detail coefficients can be discarded.

Encoding Wavelet Coefficients

What level and method of quantization is appropriate for a given signal depends upon the dynamic range, maximum and minimum values, and acceptable level of error for a time-series. In the examples below, approximation coefficients are transmitted as standard 32-bit floating-point values. This ensures that the underlying low-fidelity approximation to the time-series is accurate - it is likely that these could be quantized further. Detail coefficients are quantized to use 16 bits each, as either half-precision floating point numbers or 16-bit fixed point values, before being entropy coded.

Entropy encoders perform a simple form of lossless compression, designed to encode low-entropy data in less space than would otherwise be required. Two common techniques

include Huffman Coding[27] and Run-Length Encoding (RLE). Huffman coding requires that the encoder and decoder share a decision tree showing the mapping of bits to symbols. This can either be shared ahead of time, such as a mapping based on the frequency of letters in general english text, or calculated optimally for a given dataset and transmitted along with the data. Given that the data statistics will likely be unknown beforehand and the transmitted data for AUV telemetry will be quite short, the overhead of sending a Huffman tree may be as large as (or larger than) the data itself.

Run-Length Encoding (RLE) takes advantage of repeated sequences of symbols within data to compress the data. This is done by providing some method to reference previous data within the stream, or by providing a short way of encoding repeated sections. Due to the severe bandwidth limitations, the number of non-zero coefficients will be quite small. Since the data will be so sparse, a modified RLE-like strategy is used. Each detail coefficient to be transmitted is stored as a quantized magnitude, along with an index that identifies the scale and time associated with the coefficient.

Packet Forming

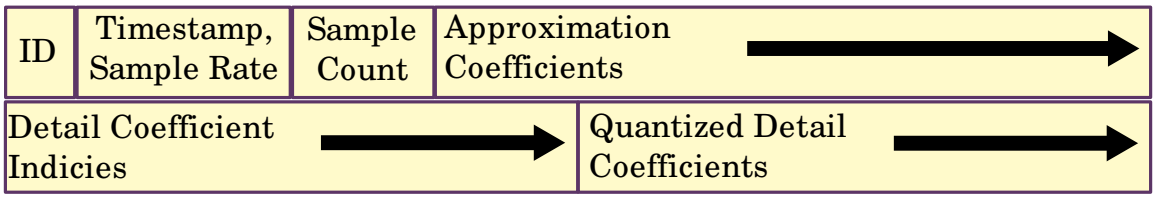


Figure 4-6: Contents of a packet for transmitting time-series data to surface observers.

Acoustic modems often use fixed packet sizes. To accomodate this reality of the medium, a fixed packet size is set as the target size of compressed time-series data. This size is typically based upon the size of packet used by the acoustic modem, a desired update frequency, and the communications TDMA schedule. A simple TDMA schedule, such as the one in Table 4.2, can provide generous cycle lengths, an LBL ping each minute, two standard 'State' packets for fallback, and still supporting a surprising number of sensors. The fixed packet size governs the level of quantization, and determines the quality of the uncompressed signal after it is received by surface observers.

The transmitted packet contains four types of data; metadata, approximation coefficients, detail indices, and detail coefficients. Metadata is sparse, and limited primarily to

Table 4.2: Sample TDMA schedule for deep chemical sensing

#	Cycle Name	Cycle Length	#	Cycle Name	Cycle Length
1	X Position	14 sec.	10	Potential Temp.	14 sec.
2	X Position	14 sec.	11	Potential Temp.	14 sec.
3	Y Position	14 sec.	12	Redox Potential	14 sec.
4	LBL Nav.	11 sec.	13	LBL Nav.	11 sec.
5	Y Position	14 sec.	14	Redox Potential	14 sec.
6	Depth	14 sec.	15	Opt. Backscatter	14 sec.
7	Depth	14 sec.	16	Opt. Backscatter	14 sec.
8	LBL Nav.	11 sec.	17	LBL Nav.	11 sec.
9	CCL State	14 sec.	18	CCL State	14 sec.
Total schedule length:			240 seconds = 4 minutes		
Rate for each timeseries:			$\frac{64\,\text{bytes}}{4\,\text{min.}} = 2.13\frac{\text{bits}}{\text{sec.}}$		

the metadata necessary for reconstructing the time-series. A single byte at the beginning of the packet identifies the sensor, and whether it is the beginning of a telemetry stream or the continuation of one. The first packet in a telemetry stream then includes information for reconstructing the timestamps associated with each sample, and the total number of original data samples. The number of original samples is used to calculate the number of detail coefficients in each level of the DWT decomposition during signal decompression.

Next, all of the approximation coefficients are transmitted though, as mentioned, selecting a wavelet with compact support minimizes their number. Finally, a subset of detail coefficients can be transmitted, along with a way to identify the detail coefficients associated scale and time. The scale and time information is stored as an M-bit index into the full array of detail coefficients. All of the indices are transmitted as fixed length M-bit unsigned integers, concatenated across byte boundaries to consume less space. This byte sequence is included in the packet, followed by the 16-bit quantized detail coefficients. Each detail coefficient thus consumes $16 + M$ bits to encode; the total number of encodable coefficients can be easily calculated from the fixed target size of the packet. When the packet is received, any detail coefficient not received is set to be zero. As a side-benefit of the quantization, the reconstructed signal will be de-noised; discarding low-magnitude wavelet coefficients is

an effective form of noise reduction[63].

Encoding the Time Axis

Each packet contains, as metadata, a 24-bit time code. The time code contains the time of day of the final sample in the time series, measured in 20ths of a second, which consumes 21 bits. A 3-bit index then selects the instrument's sampling rate from a preset list of sample rates between 20 Hz and 0.1 Hz. Combining the ending time and sample rate allows easy reconstruction of the data's time axis. The time of day could be condensed significantly in cases where there are guarantees on the age of transmitted data (such as that it was no older than the time between TDMA cycles). The sampling rate could also be omitted altogether if sensor sampling rates were known beforehand. Including it, however, offers the option for the AUV to encode a decimated version of a longer time-series in environments with heavy packet loss.

4.1.5 Results

The example time-series data was divided into sections of telemetry eight minutes long, each of which was wavelet compressed and uncompressed in turn. The data was packed into packets thirty-two bytes (256 bits) long, to simulate transmission over the FH-FSK mode employed by the WHOI Micro-Modem. The first packet contained the metadata described above; each successive packet contained only a one byte packet ID in addition to coefficients. Source code for an interactive version of the wavelet encoder and decoder, and example usage, are provided in Appendix A, though it does not include a full implementation of the TDMA cycle and packet division used here. Figures 4-8, 4-9 and 4-10 show the results of compressing the two sample datasets using three different compression levels.

For the smallest number of packets, where two were sent every eight minutes, sixty-four bytes were telemetered. For the largest, 256 bytes were telemetered - one thirty-two byte packet per minute. When using only two packets, almost half of the space is devoted to metadata and approximation coefficients; the larger encodings devote more space to detail coefficients and indices as shown in Figure 4-7. At the lowest compression level, the wavelet compressed Eh data does not fully capture the 'double dip', though the potential temperature is captured relatively accurately. At the medium compression level, most of the detail in both signals has been captured. By the highest level of detail, high frequency

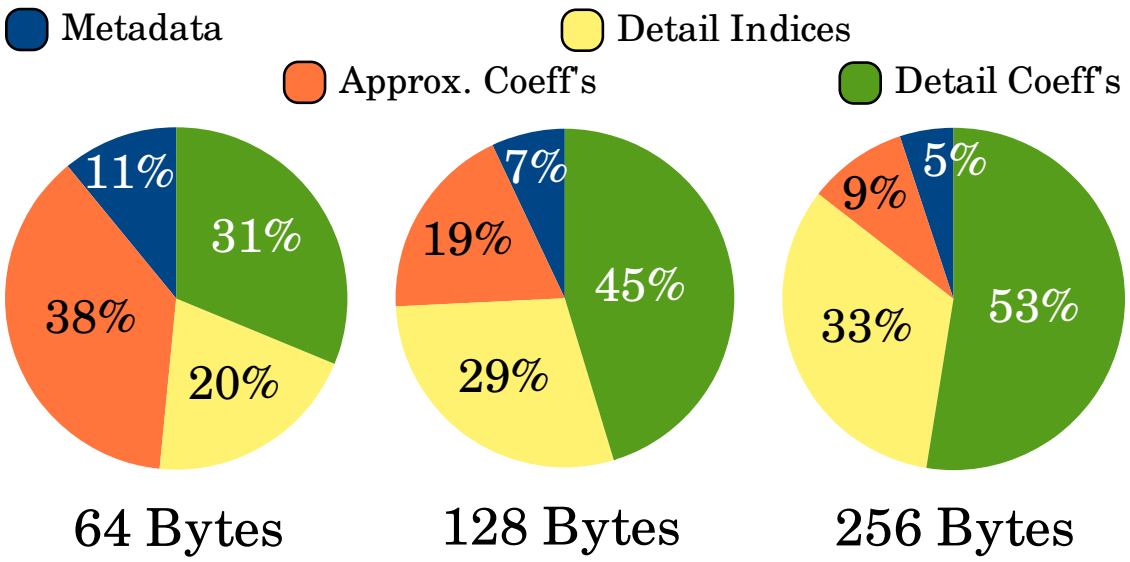


Figure 4-7: Percentage makeup of a wavelet-encoded packet for different encoding lengths.

details, or possibly noise, has begun to be replicated. Even operating at only tens of bits per second, an acoustic modem could transfer this *two-hour* section of telemetry in less than a minute.

Capturing high-frequency details such as the 'double dip' can be very important for some missions. When localizing hydrothermal vents, a rapidly decreasing signal indicates contact with a hydrothermal plume; two hits can mean two plumes. Identifying that there are two plumes might keep scientists from selecting the middle of the dip for a follow-up survey, and missing both plumes.

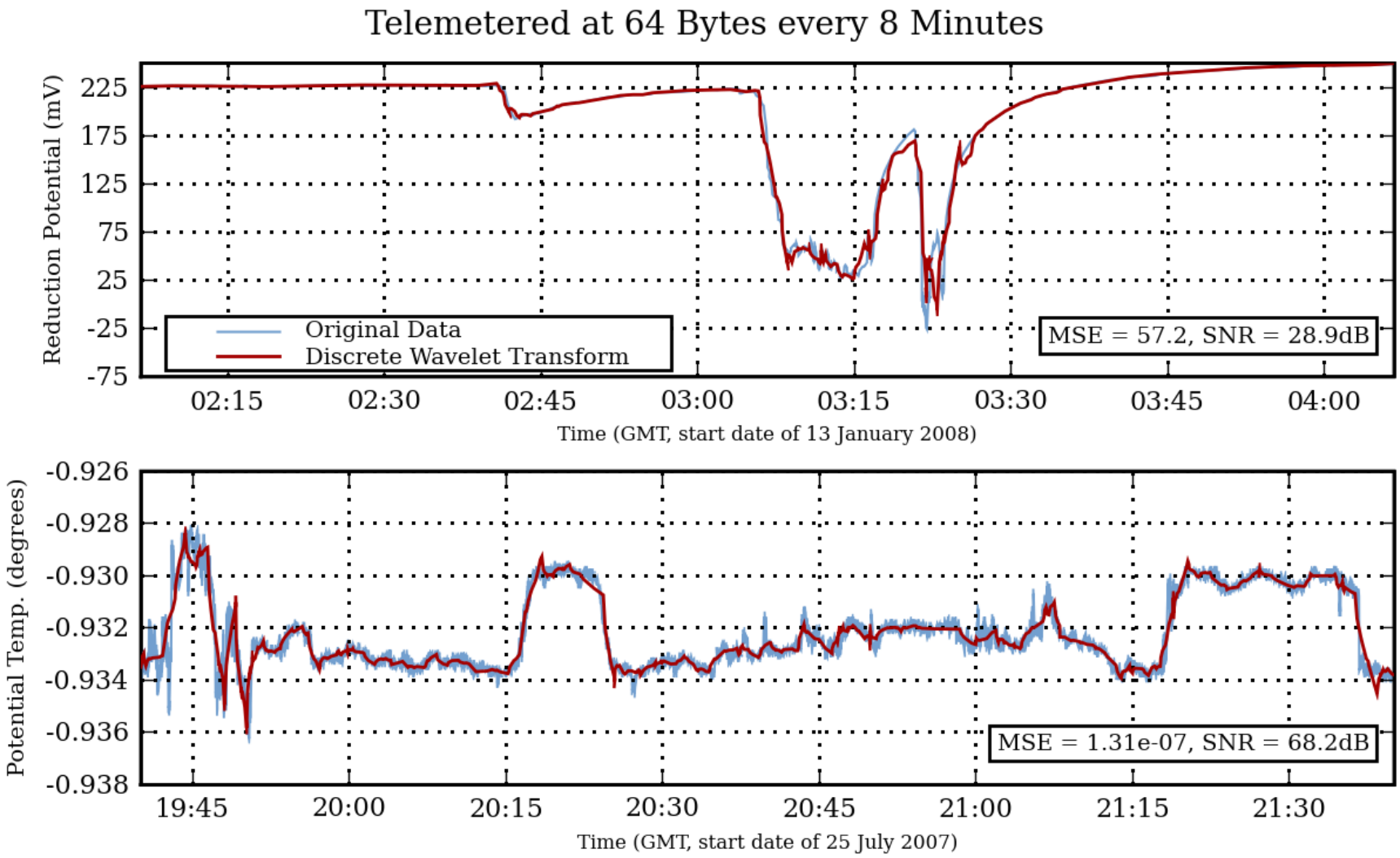


Figure 4-8: Example Eh and potential temperature science data telemetered using 64 bytes every 8 minutes.

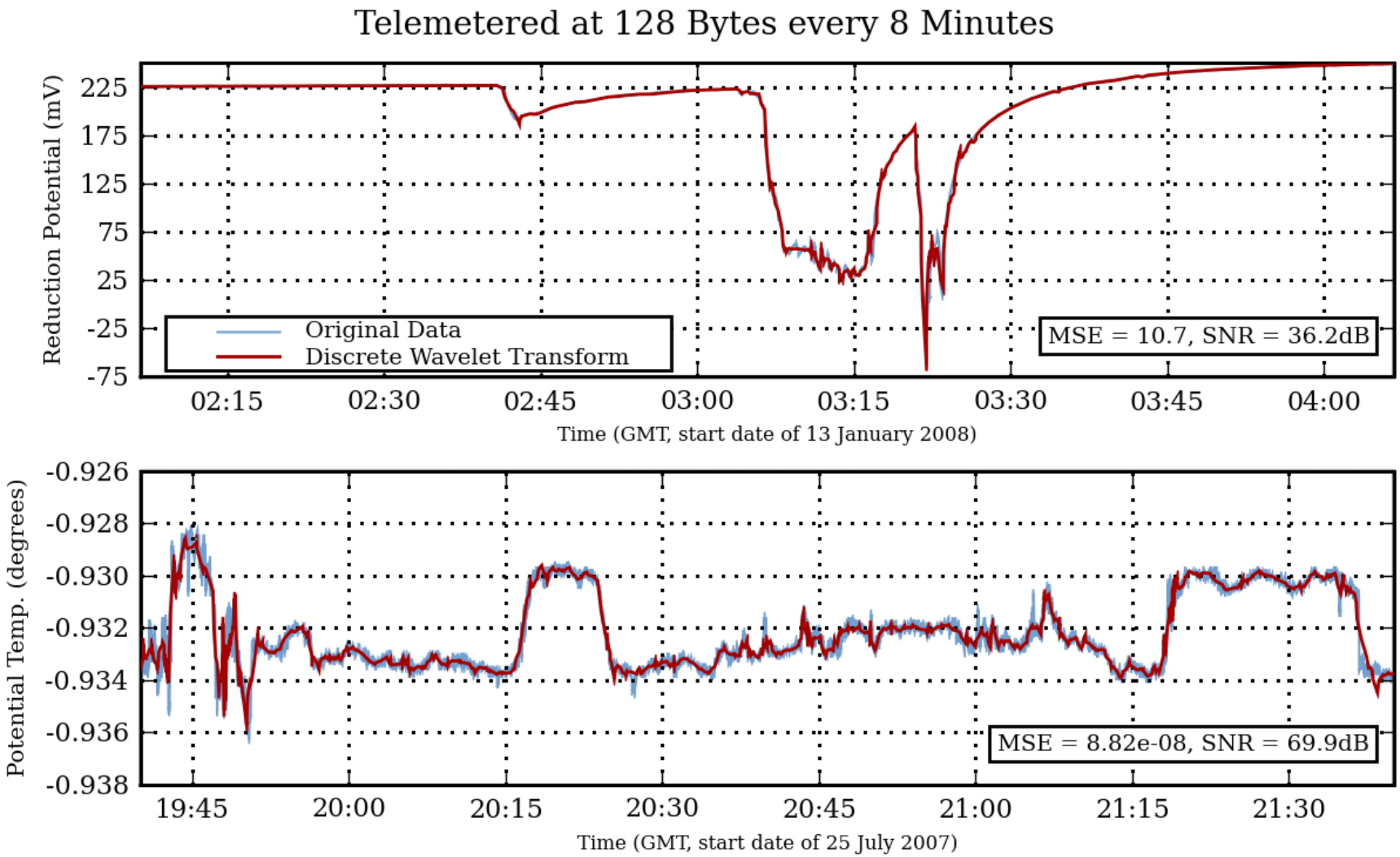


Figure 4-9: Example Eh and potential temperature science data telemetered using 128 bytes every 8 minutes.

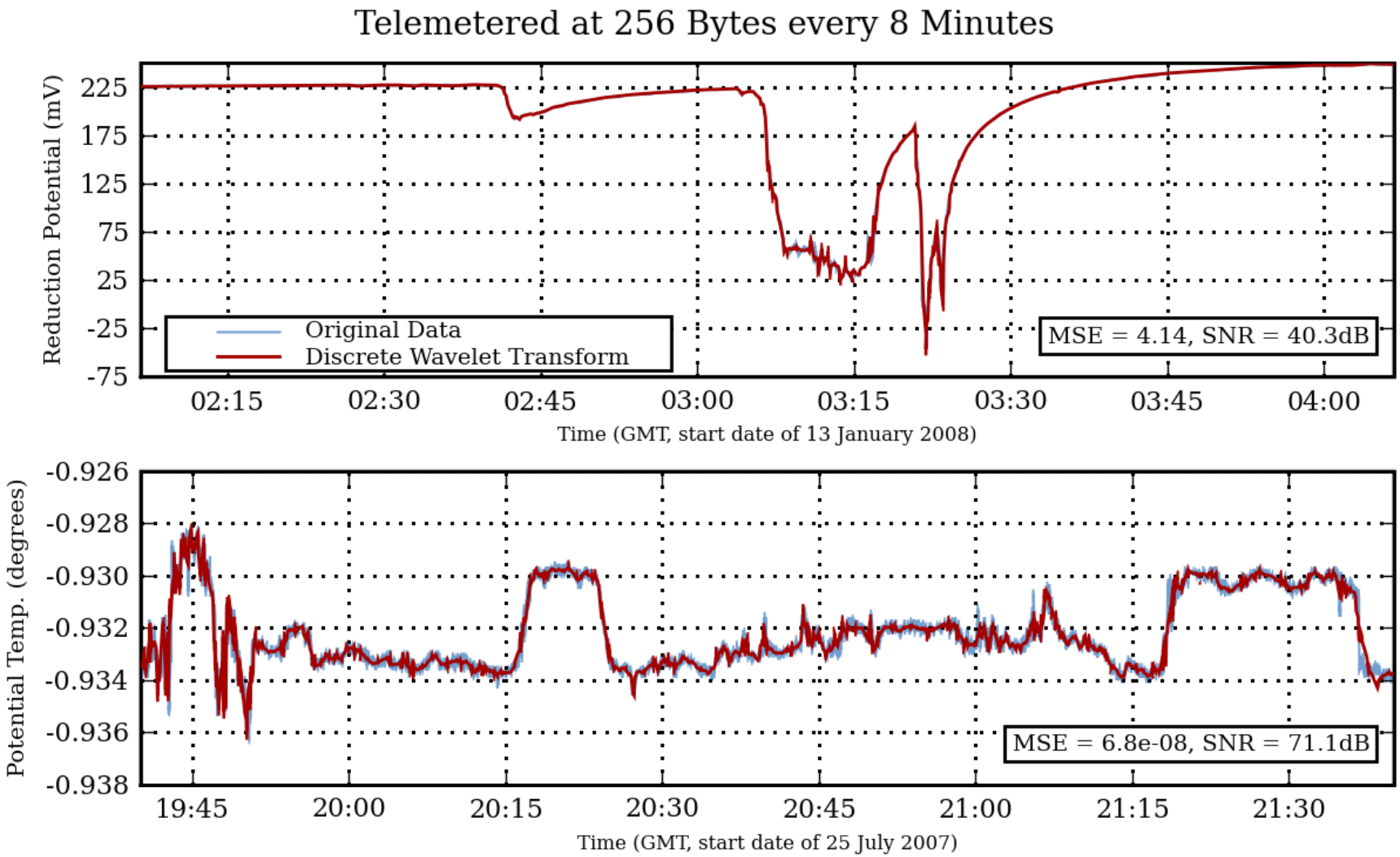


Figure 4-10: Example Eh and potential temperature science data telemetered using 256 bytes every 8 minutes.

4.2 Compression of Image Telemetry

As mentioned in Chapter 3, machine vision techniques can turn a camera into a source of time-series scalar telemetry. This is a powerful technique, capable of compressing thousands of images to a small amount of data. The time series presented before, generated by calculating the median luminance for each of over 6700 images from a dive, was compressed with the Daubechies wavelet, using the wavelet compression technique presented above. The result is shown in Figure 4-11. While small differences can readily be seen from the actual data, this representation consumes 2048 bits instead of 215232 bits – a data compression ratio of over 10500%. Over five hours of imagery could be transmitted in less than a minute with this representation.

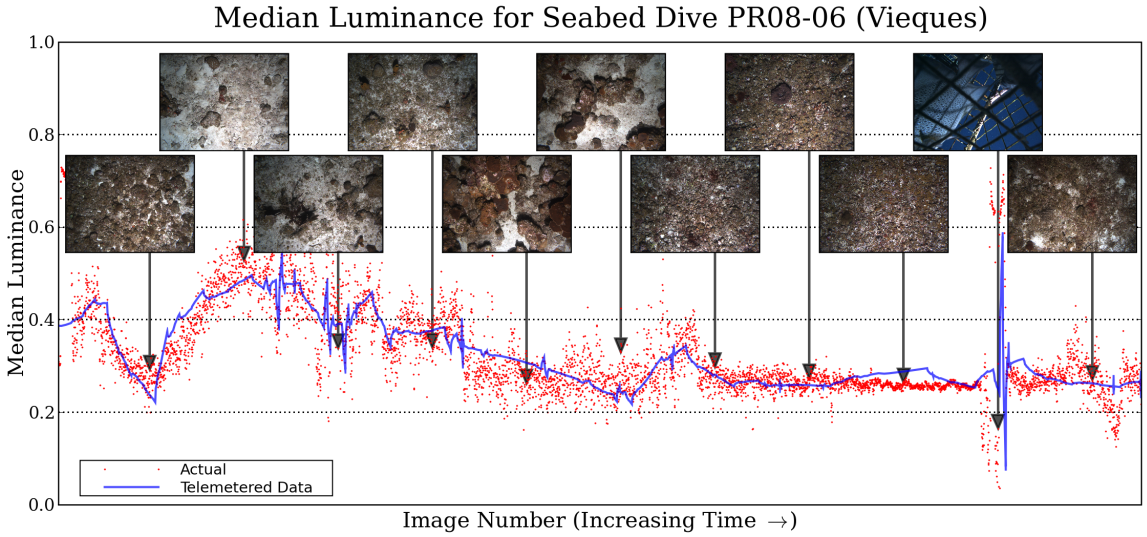


Figure 4-11: An example of using a camera as a scalar sensor. Dots indicate the median luminance of a single captured image. The entire series was transmitted, as the blue line shows, after being compressed to 2048 bits with the *Daubechies 2* wavelet.

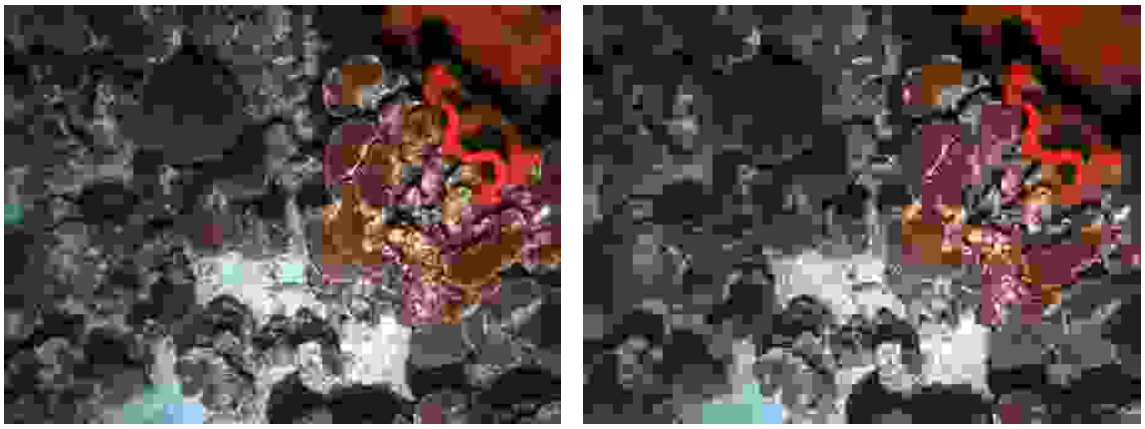
Full photos can be subsampled and transmitted to the surface. The limited available bitrate means, however, that extensive image compression will be necessary. Table 4.3 lists the time required to transmit a single 448 pixel by 336 pixel color image to the surface, in minutes, for a range of image compression levels and acoustic communication bitrates. To transmit a new image every few minutes, and to allow other telemetry to be sent to the surface, we see from the table that transmitting an image compressed to an extremely aggressive level at 0.11 bits per pixel will completely saturate a 240 bit per second acoustic

Modem rate (bits/sec.)	*Bits per Color Image Pixel*								
	0.03	**0.05**	**0.07**	**0.09**	**0.11**	**0.13**	**0.15**	**0.20**	**0.25**
40	1.9	3.1	4.4	5.6	6.9	8.2	9.4	12.5	15.7
80	0.9	1.6	2.2	2.8	3.4	4.1	4.7	6.3	7.8
120	0.6	1.0	1.5	1.9	2.3	2.7	3.1	4.2	5.2
160	0.5	0.8	1.1	1.4	1.7	2.0	2.4	3.1	3.9
200	0.4	0.6	0.9	1.1	1.4	1.6	1.9	2.5	3.1
240	0.3	0.5	0.7	0.9	1.1	1.4	1.6	2.1	2.6
280	0.3	0.4	0.6	0.8	1.0	1.2	1.3	1.8	2.2
320	0.2	0.4	0.5	0.7	0.9	1.0	1.2	1.6	2.0
360	0.2	0.3	0.5	0.6	0.8	0.9	1.0	1.4	1.7

Table 4.3: Minutes required to transmit a single color image for various levels of image compression and acoustic modem bitrates.

link for a minute. To maintain a reasonable rate of image transmission at these moderate dimensions, a compression rate of around 0.05 to 0.2 bits per pixel is necessary.

4.2.1 JPEG



(a) JPEG compressed image (0.17 bits per pixel) (b) JPEG compressed image (0.11 bits per pixel)

Figure 4-12: Comparison of JPEG compression at two different data rates. At these low bitrates, the image has degraded to where colors and textures are unrecognizable, and shapes become difficult to discern.

The widely used JPEG standard[65] is essentially a two dimensional analog of the canonical lossy compressor discussed for scalar compression, using the Discrete Cosine Transform (DCT) for source coding. Images are broken into blocks of 8x8 pixels, each of which then is transformed using a two dimensional form of the DCT. The resulting

coefficients are then scalar quantized, and entropy coded. Standard JPEG compression, however, does not perform well at very low bitrates. The JPEG standard highly quantizes color information, meaning that at low bitrates color information is almost entirely gone, or badly distorted as seen in Figure 4-12. High-resolution details are also entirely lost, leaving large blotches of different brightnesses, with few discernable features. The severe losses in coding quality observed in JPEG images when compressed to more than 0.15–0.20 bits per pixel (40:1) are well documented[25], and make JPEG unfit for our purposes.

4.2.2 Wavelet Methods

Numerous image compression algorithms based on the Discrete Wavelet Transform (DWT) have now been developed, and found to have much higher performance at low data rates[25] than JPEG compression. These include the next generation algorithm from the Joint Photographic Experts Group, JPEG2000[70], and the Set Partitioning in Hierarchical Trees (SPIHT)[45] algorithm. While both algorithms use wavelet compression for source coding, they differ in their quantization methods and entropy coding of the wavelet coefficients. Figures 4-13, 4-14 and 4-15 on the next several pages show the results of compressing the example images using both JPEG2000 and SPIHT at different levels of compression.

While detail is still lost at the lowest resolutions, the resulting images are smoother to the eye, and degrade more gracefully. There are none of the block artifacts which dominated the low bitrate JPEG images, and color information gradually diminishes rather than being charicatured into cyan and brown splotches as in Figure 4-12(b). Details such as the gorgonian are preserved relatively well, considering the low bitrate, which is not surprising given the strengths of wavelet compression in representing discontinuities. What is not represented well is texture information; even the coral in the images compressed at 0.15 bits per pixel has been reduced to a muddy brown and grey mess - none of the original texture information is discernable. Vector quantization, described below, allows us to trade image detail for texture information, while decreasing required bitrates.

4.2.3 Vector Quantization

Vector quantization differs from the previous compression methods in that there is no initial image transformation involved. Instead, vector quantization relies on a pre-generated tile library, consisting of fixed-sized tiles from previously seen training images. During encoding,

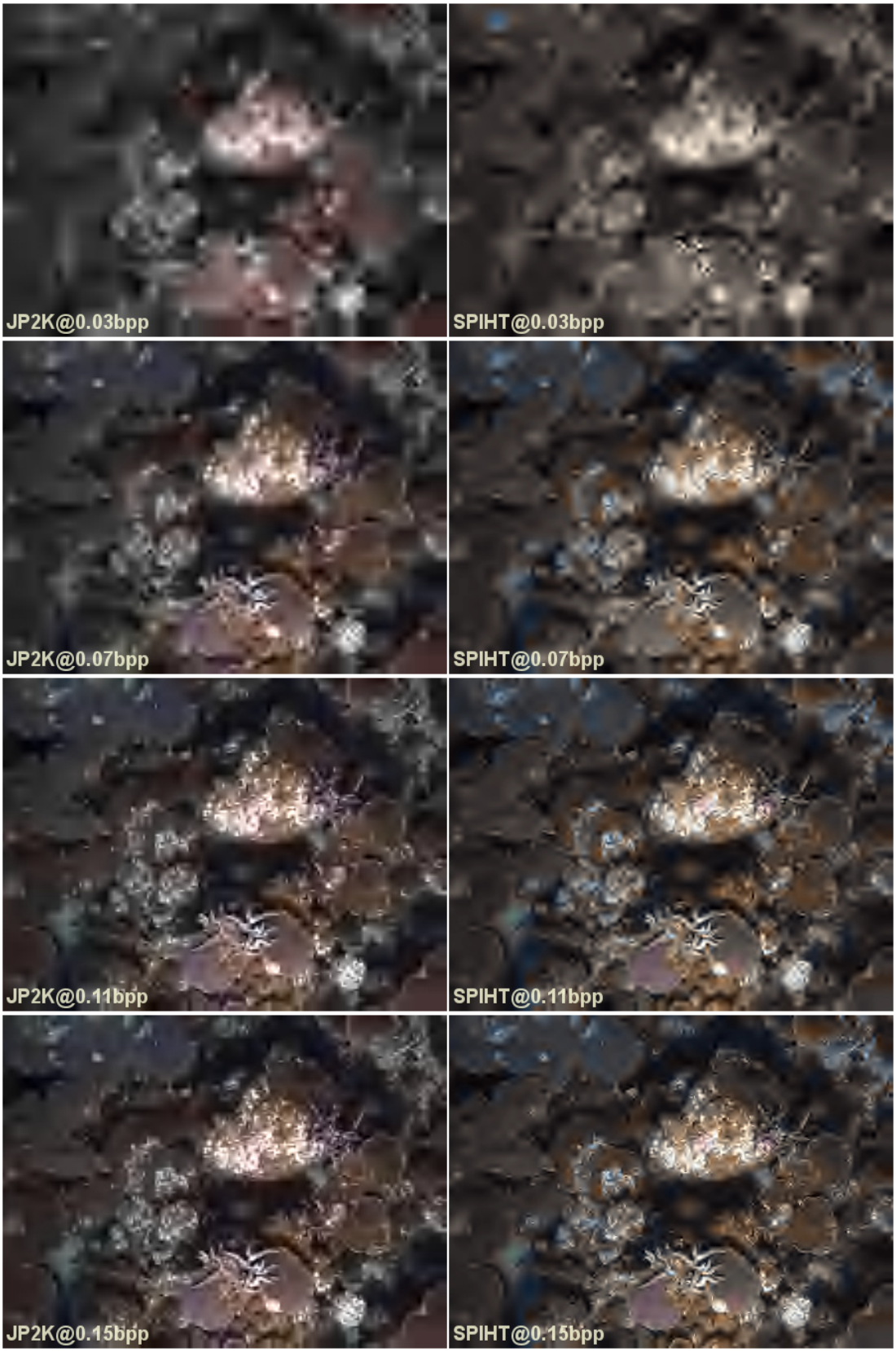


Figure 4-13: Results of wavelet compressing first sample image.

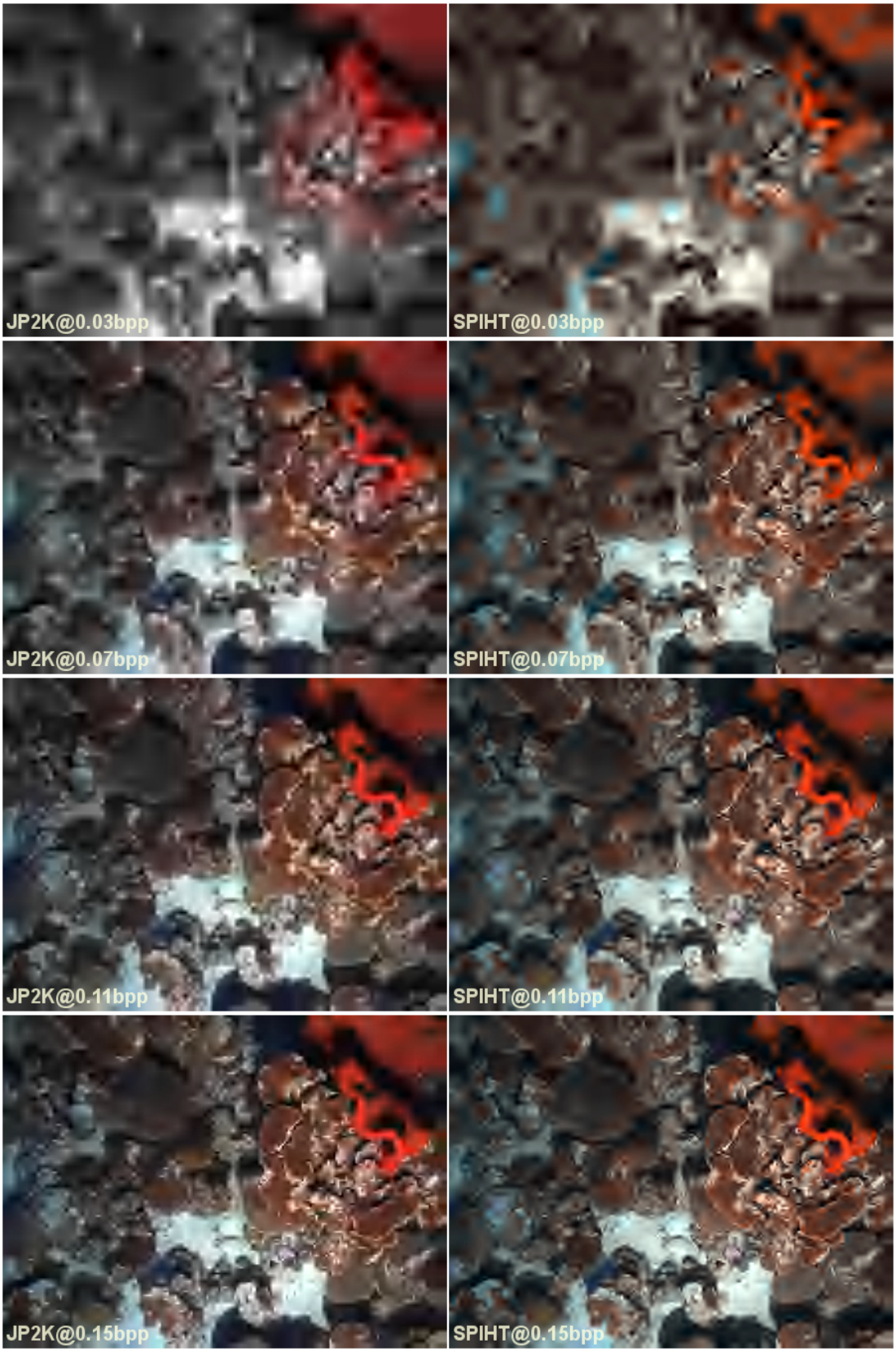


Figure 4-14: Results of wavelet compressing second sample image.

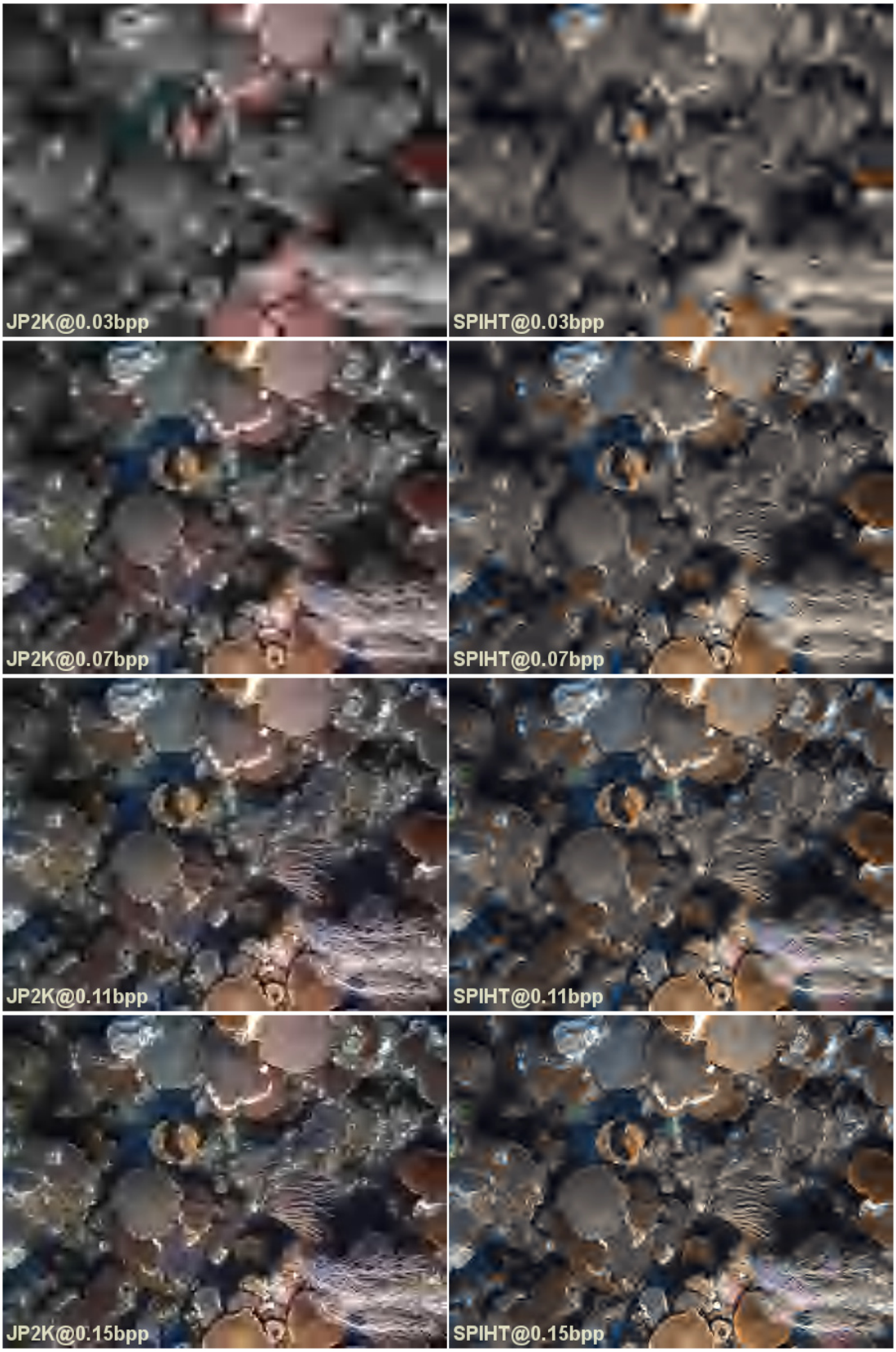


Figure 4-15: Results of wavelet compressing third sample image.

an image is segmented into fixed-sized blocks the same size as the tiles in the library. A tile from the library is found which best matches each block, and the index for that tile is stored as the value for the block. This allows hundreds of pixels to be represented by a single index, which ranged between 17 and 22 bits for the examples shown below. It is, in essence, the process of generating a photomosaic from a set of previously seen image pieces.

This library of previously seen image pieces can be generated automatically by dividing a set of training images into tiles, or can be carefully constructed to contain a specific set of textures based on the expected content of new images. Each of the three sample images used throughout this section are from the same AUV dive performed off of Puerto Rico in 2008. To generate a tile library which would not contain the images to be encoded, images from the previous dive on a nearby site were divided into tiles. The final compression rate when using vector quantization depends on a number of factors, including the size of the compressed image, size of image tiles, and number of tiles in the library. Equation 4.2 shows the relationship between each of these factors and the final compression rate of the image.

$$\frac{\text{bits}}{\text{pixel}} = \overbrace{\frac{1}{w_t \times h_t}}^{\text{Tiles per pixel}} \times \overbrace{\left\lceil \log_2 \Big(\underbrace{N \times \left\lfloor \frac{w_i}{w_t} \right\rfloor \times \left\lfloor \frac{h_i}{h_t} \right\rfloor}_{\text{Tiles in library}} \Big) \right\rceil}^{\text{Bits per tile}} \tag{4.2}$$

Image encoding is performed by dividing the image into fixed size tiles of the same dimensions as those in the tile library. The best match for each of those tiles is then located in the library, and the tile index is stored to represent that section of the image. While each tile was encoded in the below examples by an index with a fixed number of bits, entropy coding could improve compression ratios even further. Selecting the best match for each tile can be done in a number of ways, though typically Euclidean distance is used. The images below were encoded using the Euclidean distance in the YUV color space. Two of the sample images are shown in Figures 4-16 and 4-17 after being compressed with a range of tile sizes.

While little or no detail is visible in the most-compressed images, they still impart a fairly clear understanding of the terrain that the vehicle is over. For some tasks, such as determining rugocity, that may be sufficient. The least-compressed images capture a large amount of detail; a comparison of the images compressed at 0.082 and 0.050 bits per pixel with wavelet-compressed images at similar sizes shows comparable levels of detail, with

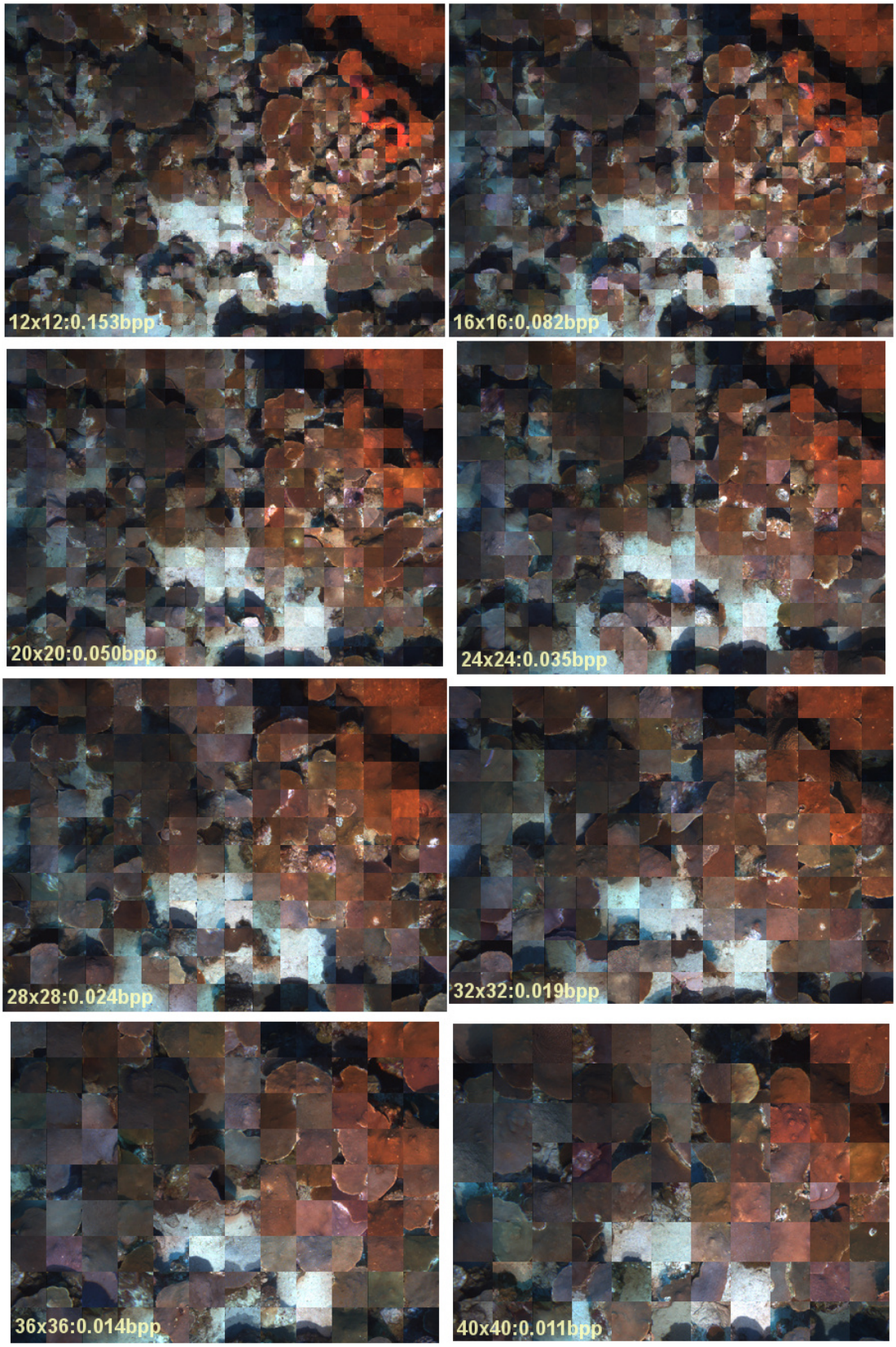


Figure 4-16: Results of Vector Quantization.

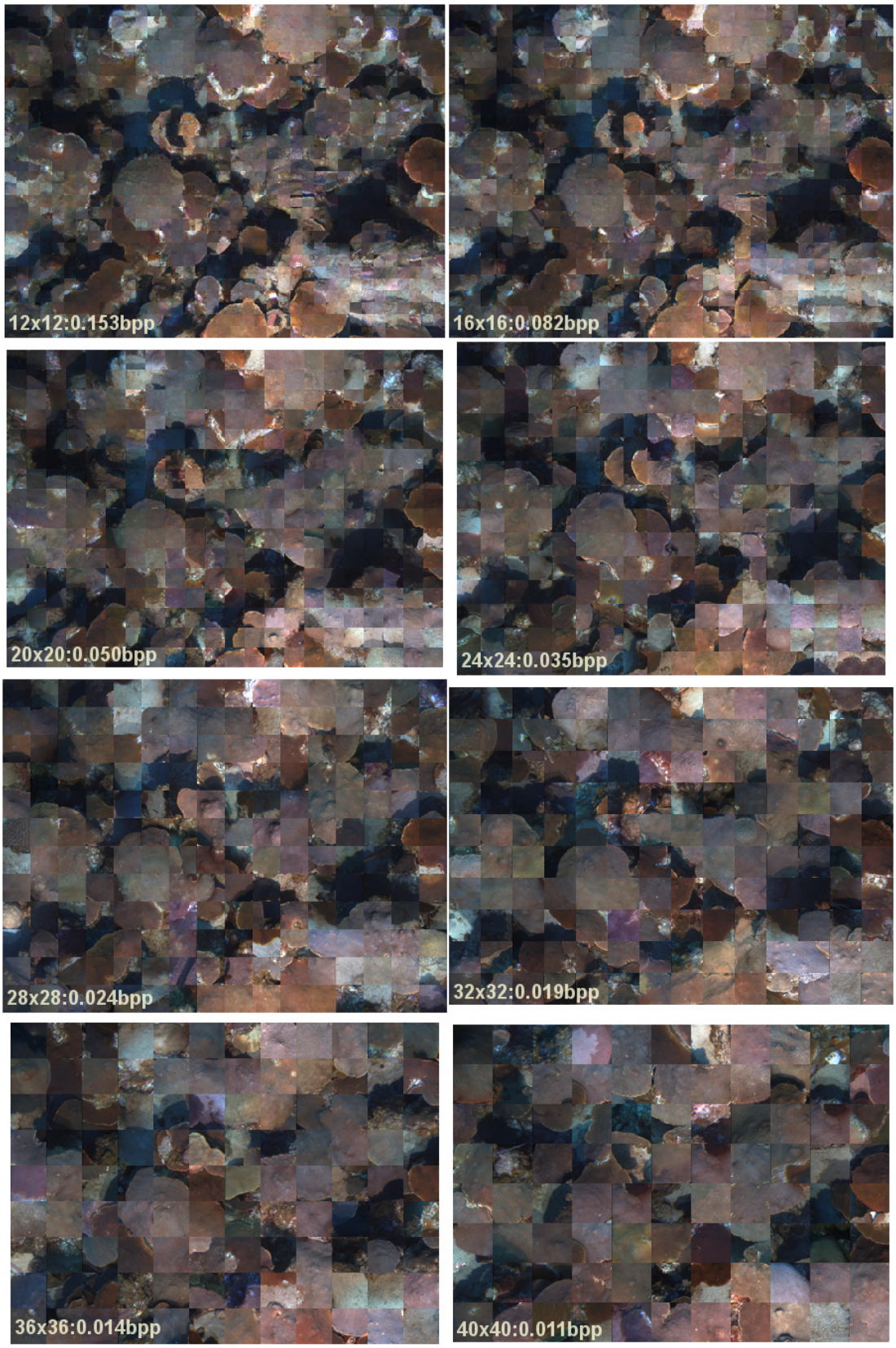


Figure 4-17: Results of vector quantization (image 2).

much better texture and color fidelity.

While decoding a vector quantized image is straightforward, encoding an image using vector quantization is computationally expensive and can take quite a long time, as a match must be found by comparing every candidate tile against the tile to be encoded. While there has been extensive research on this problem, a simple heuristic used by photomosaicking software[42] can speed up encoding significantly. Rather than comparing full-resolution tiles, downsampled versions of each tile can be compared. Results are comparable to those obtained by comparing full resolution tiles, as seen in the examples below. Figures 4-18 and 4-19 show the same sample images presented before, except tiles were selected using 5x5 tiles for comparison. This yielded up to a 100x improvement in encoding time, with no obvious decrease in viewing quality. The images encoded in this way appear in some cases to contain more detailed textures than those encoded using complete tiles, though possibly not as good matches.

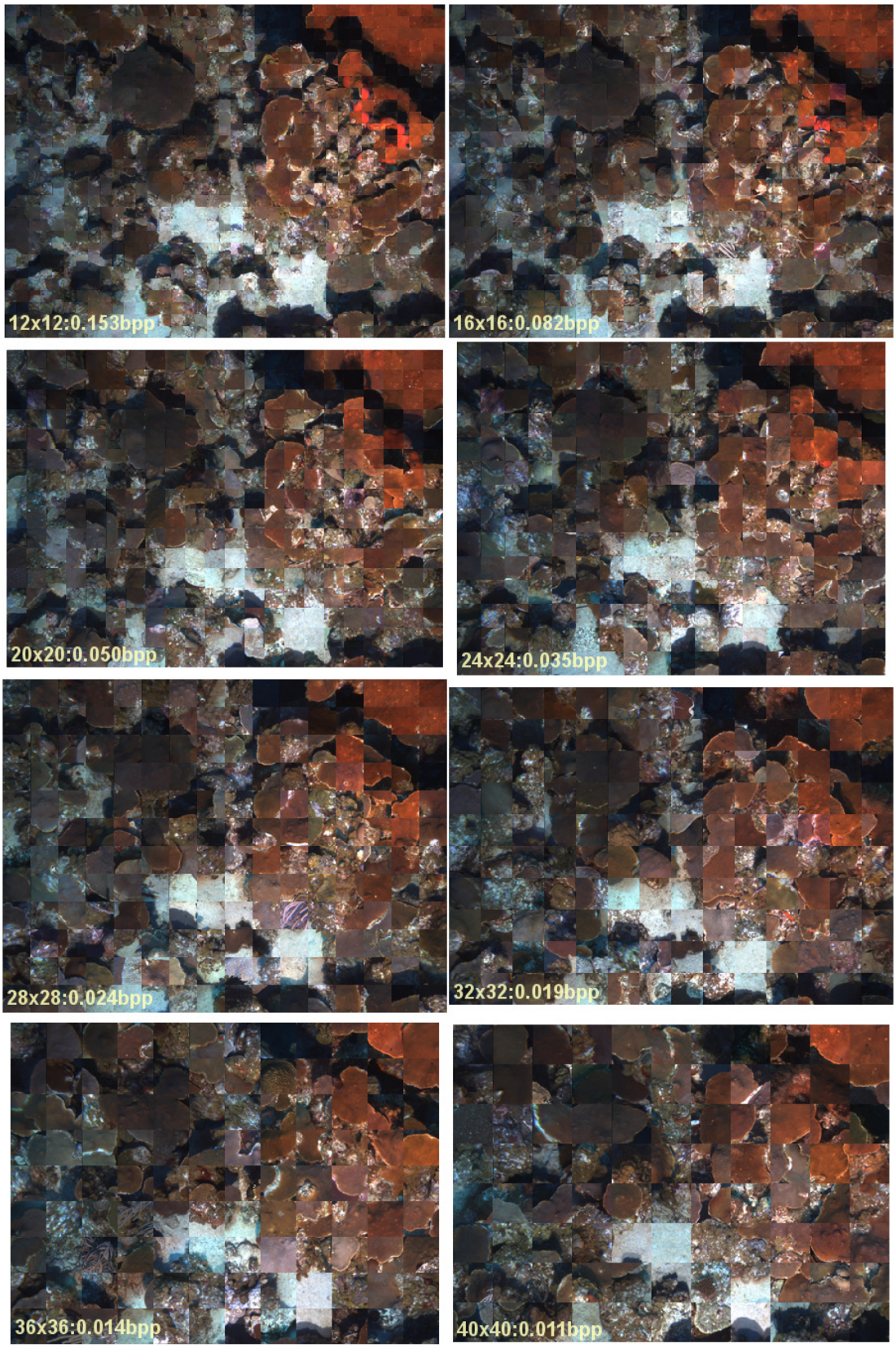


Figure 4-18: Results of vector quantization using 5x5 tile matching (image 1).

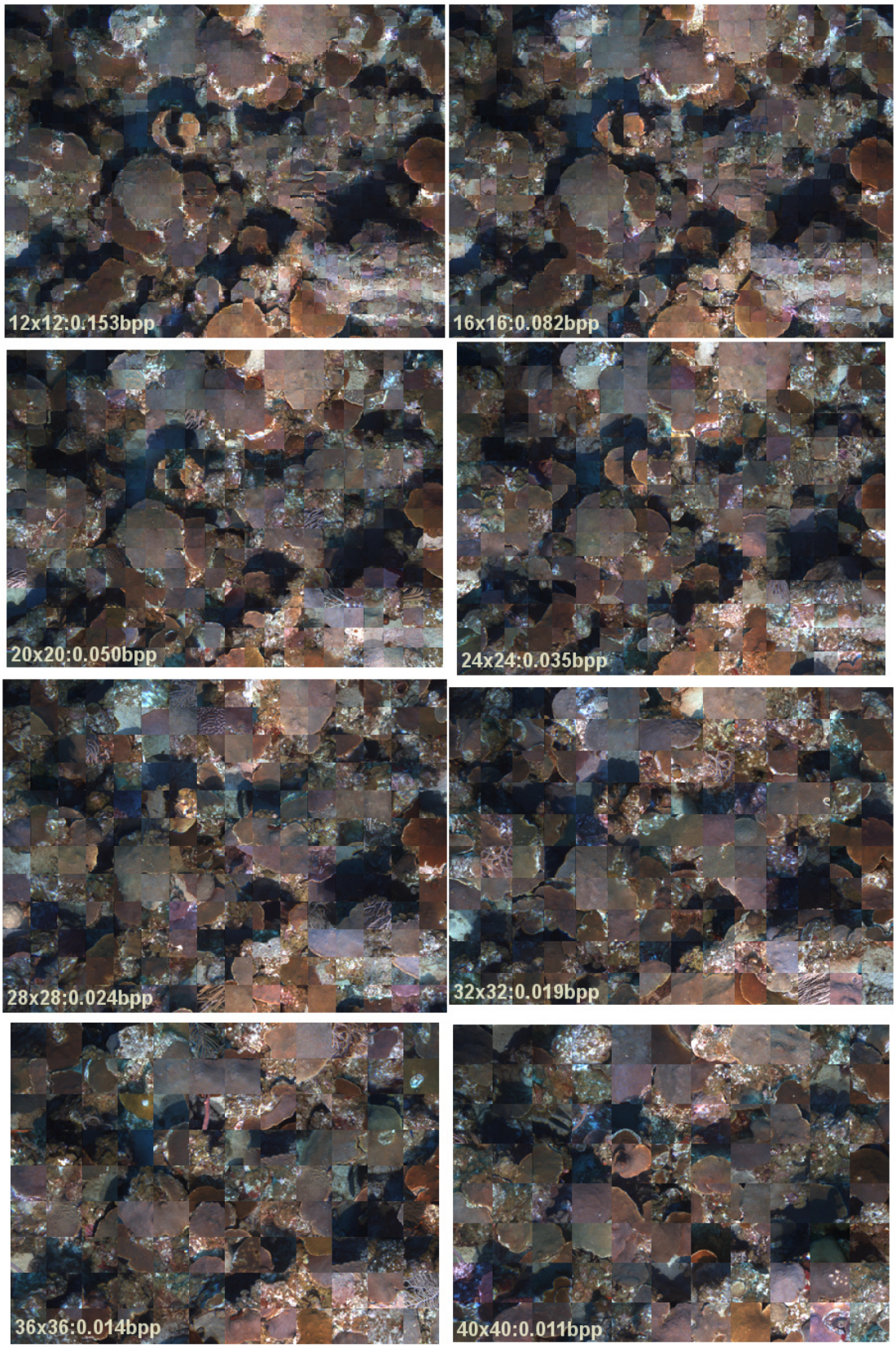


Figure 4-19: Results of vector quantization using 5x5 tile matching (image 2).

CHAPTER 5

User Interface

AUVs are typically deployed to help achieve broader scientific goals; for example, characterizing a coral reef, locating sites of hydrothermal venting or generating a photographic survey of archaelogically valuable sites. These missions may require using multiple oceanographic assets, such as a CTD or multibeam sonar system. Ideally, transmitted telemetry from AUVs should be presented in the context of this other acquired data, both while streaming and after it has been archived. The system's primary goal, therefore, is to provide a simple interface to the real-time data and contextual data, understandable by scientists without a computer science background, and capable of running on a variety of operating systems. Rather than attempting to extend any single tool to meet the specific needs of AUV telemetry, a novel system for displaying streaming telemetry and previously captured oceanographic data was designed around the Keyhole Markup Language (KML).

KML is an Extensible Markup Language (XML) schema for representing and exchanging geographic data. Since being created as the native file format for what is now *Google Earth*, KML has evolved into an open standard under the auspices of the Open Geospatial Consortium[38]. Many open-source and commercial geodetic software packages now support KML as a data interchange format, including Microsoft Virtual Earth, NASA's World Wind, KDE's Marble, and ArcGIS Explorer[71]. While several of those tools provide a friendly interface for authoring KML documents, KML can also be easily created or edited programatically. The widespread adoption of KML in GIS tools for professionals and the public means that scientists are likely to be familiar with KML-based tools already; they won't need to relearn a new software package. Additionally, KML clients are available for all major operating systems. Finally, KML data files can easily be extended to contain mission, copyright, or any other form of metadata as appropriate. KML is easily created or edited programmatically, or via a number of graphical interfaces.

5.1 Representative Interface

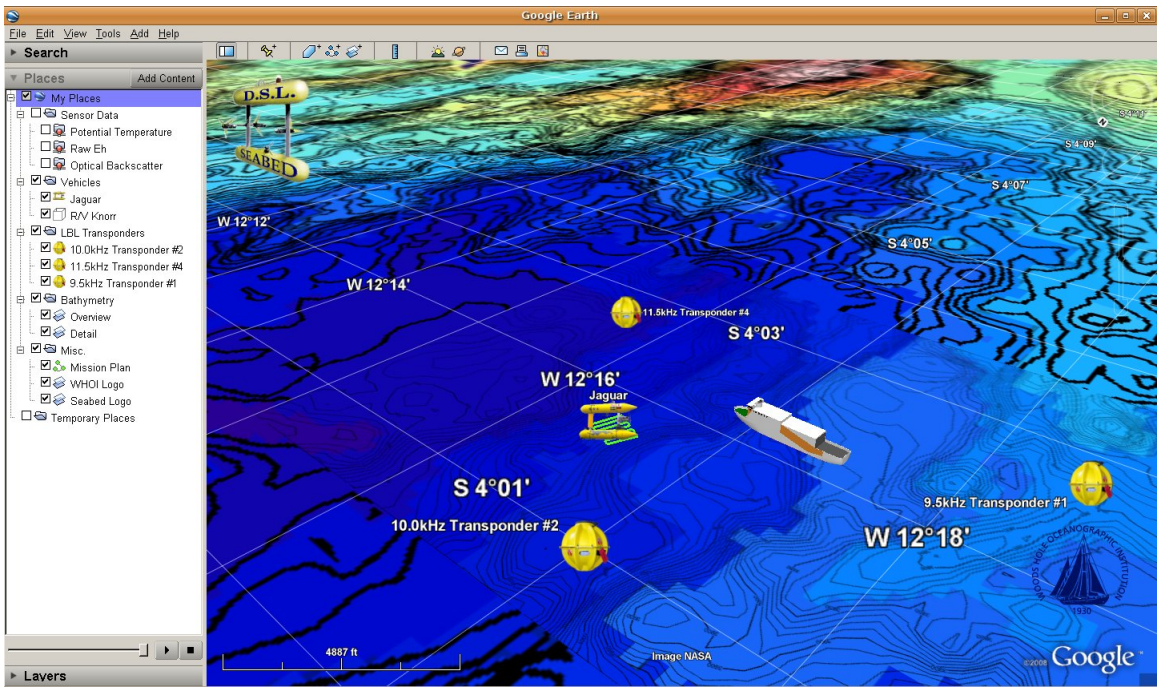


Figure 5-1: Example AUV mission displayed in Google Earth, including LBL beacon locations and two scales of multibeam bathymetry. This interface was used during a cruise on the Southern Mid-Atlantic Ridge to plot ship and AUV locations in live time as telemetry was received from ship GPS and AUV acoustic communications.

The flexibility of the KML format allows a variety of forms of data to be included besides science telemetry. Figure 5-1 shows a base KML file being displayed in Google Earth. Each item can be selected or deselected from the interface, allowing users to display only what they need to see at any given time. Figure 5-1 shows two different scales of pre-rendered multibeam bathymetry, planned mission tracklines, AUV location, a latitude and longitude grid, the location of three LBL transponders, and the current ship location and heading represented in 3D.

As science telemetry is transmitted from the underwater vehicle, it is displayed almost instantaneously in the interface for each KML client as a sequence of color-coded samples. Clicking on a sample brings up a detail box which gives the exact value for the point. The color code and scale extrema are preset for each sensor, but can be easily modified. Figure 5-2 shows scalar data being displayed as it is streamed from an underwater vehicle. Photos, while not shown in the figure, are represented as camera icons. When clicked, the photo

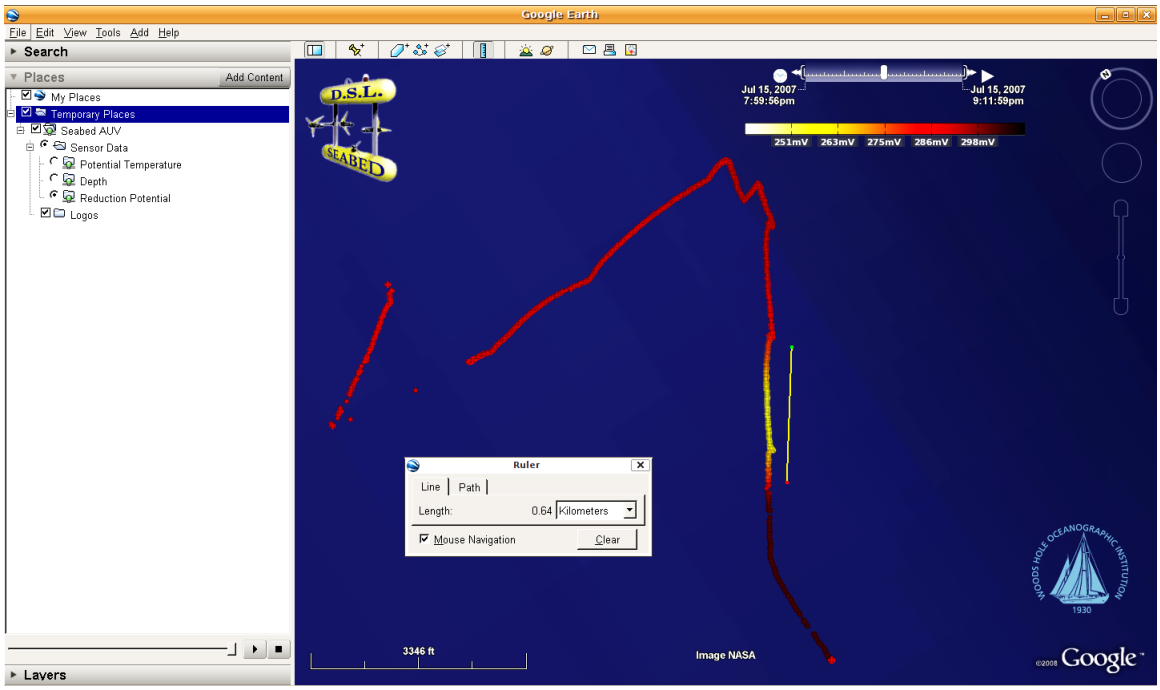


Figure 5-2: Streaming Eh telemetry from a volcanically active portion of the Arctic Ocean's Gakkel Ridge[51] being reviewed in Google Earth. Users can scan through time, view data from different sensors, or annotate it. Shown is the use of a ruler to measure the length of an interesting section of Eh data.

loads in a small popup dialog.

5.2 Architecture

The primary software consists of a pair of programs running on a server which receives raw telemetry from the AUV. The server need not be connected directly to the AUV's acoustic modem; typically it receives data via User Datagram Protocol (UDP) broadcasts from another computer setup for communication with the AUV. A standard setup is shown in Figure 5-3.

The server software is split into two seperate programs; one application decodes received telemetry from the AUV and stores it in a database, the second application generates and serves KML files via a built-in HTTP server. While the scripts need not run on the same server, the database (consisting of a single file) must be on a filesystem accessible to both servers.

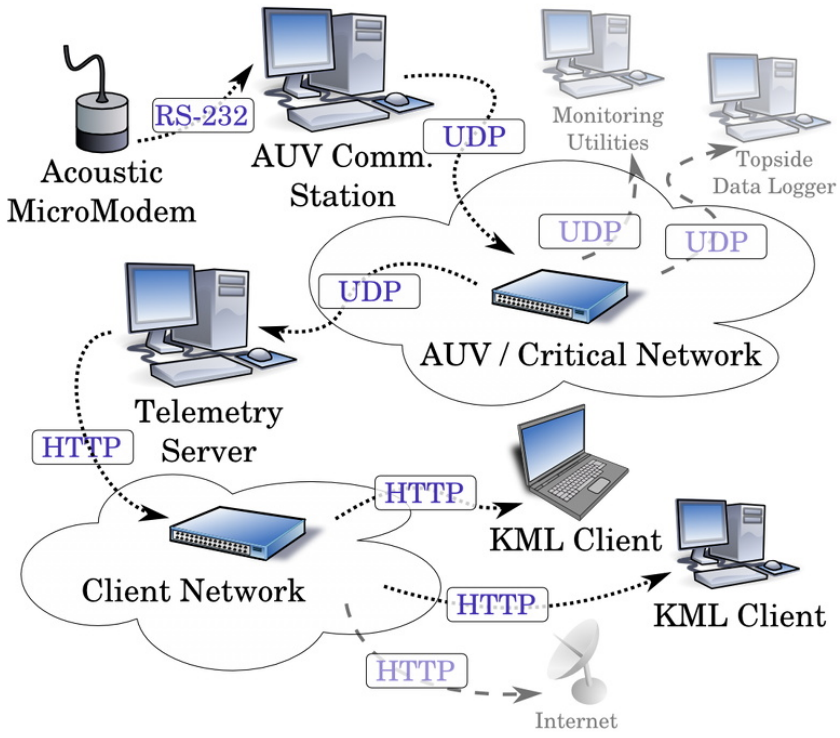


Figure 5-3: System diagram for typical operation of KML data server. Two independent networks can be used to isolate critical AUV network traffic from ship network traffic, but are not required.

5.2.1 Telemetry Decoding

The telemetry decoder consists of a Python[62] script which receives raw telemetry data over UDP. The telemetry is decoded using Numerical Python[4] and the Python Wavelet Toolbox[67]. As telemetry describing the position of the AUV (x, y, depth) is received, it is interpolated to the time-axis of each other sensor's data telemetry which has been received. In the current implementation, each packet contains samples up to the moment the packet was transmitted rather than the start of the TDMA cycle. Thus it is likely that at any given time there will be sensor data which has not yet been geo-referenced due to a lack of necessary position information.

After the telemetry is decoded and geo-referenced, the resultant time-series data is stored in a Hierarchical Data Format (HDF) version 5 file. HDF[23] is a high-performance database designed for scientific datasets. Each sensor is stored in a seperate but identically defined table. The table has four columns for the time, value, interpolated latitude and interpolated longitude of each sensor reading. Our implementation used Python Tables[2] as a wrapper

around the HDF format to facilitate programming during capture and postprocessing.

5.2.2 KML Generation

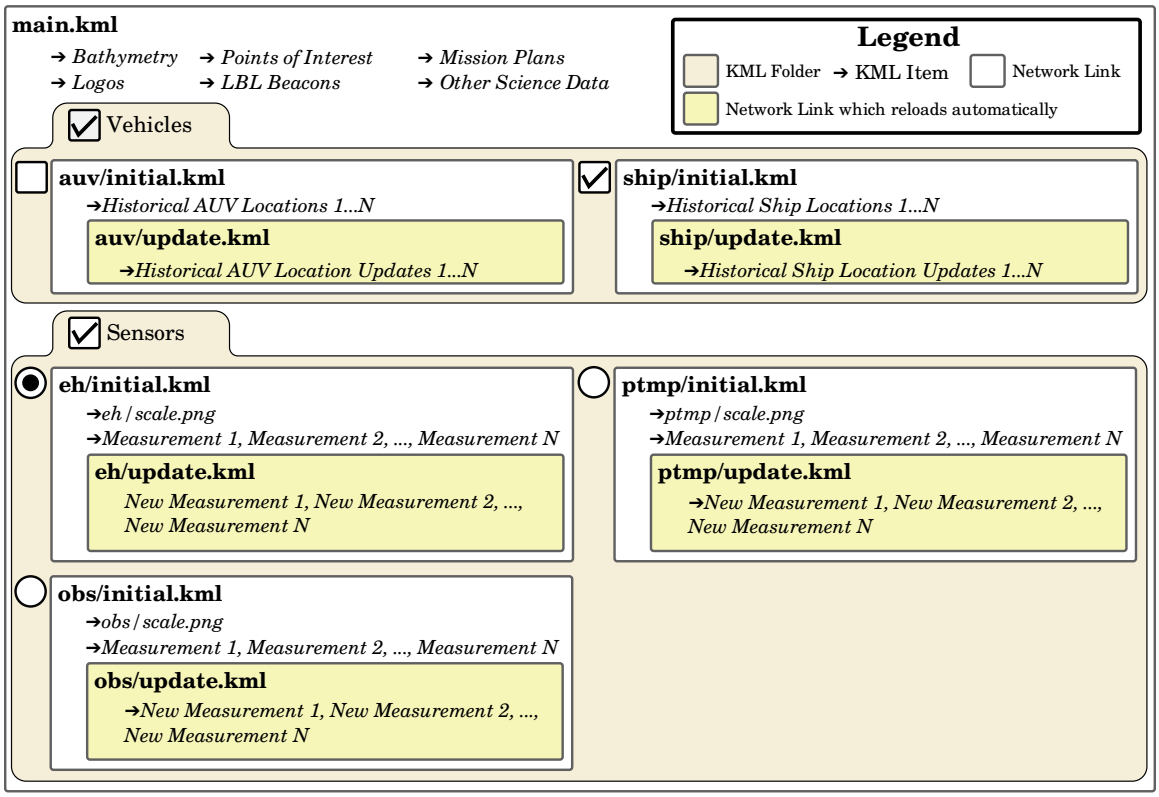


Figure 5-4: Structure of the KML files used by the topside system.

KML is dynamically generated by a WebPy[48] script which is called to handle incoming HTTP requests. There are a number of interrelated KML files which are generated, as depicted in Figure 5-4. A KML client will initially request a base KML document (main.kml in Figure 5-4) containing any pre-existing data such as the locations of LBL transducers, mission tracks, CTD highlights, and multibeam bathymetry. Additional geo-referenced information can be manually added by the user, if desired, after loading the telemetry KML file.

The KML file also contains network links for telemetry from each sensor, and for basic AUV and ship tracks. Only one of these network links is allowed to be active at a time as they cannot be viewed on top of each other, and attempting to do so would needlessly increase server load. Finally, when this initial file is requested a unique client 'cookie', or session ID, is generated for the KML client to use with all future HTTP requests. This

session ID is used to track the last data point sent to each client, which allows multiple clients to be updated asynchronously. New clients can be added at any time by requesting the base KML document from within the client.

The initial data file requested for each sensor or vehicle position (. . . /initial.kml in Figure 5-4) loads any data initially available for the sensor, along with any sensor-specific information that should be displayed to the user. Currently, a color scale is provided for each sensor. This scale is dynamically generated based on a predefined colormap and data limits for each sensor. Each initial data file also includes yet another network link to a document which provides updates when new data is received from the AUV or ship (. . . /update.kml in Figure 5-4). Each sensor or track which is being displayed will request this file at a regular interval; the rate at which the client checks for updates can be set by the server or overriden by each client individually. Each telemetered sample is represented by a single KML 'Placemark', and includes a timestamp for when the sample was acquired. KML clients which support timestamps, such as Google Earth shown in Figure 5-2, allow users to scan backwards and forwards in time, as well as switch between sensors and change viewpoints. Though the files generated by our topside system make full use of advanced KML features such as Network Links and timestamps, clients which do not support these features should degrade gracefully. Clients which do not support Network Links can still have each sensor added individually, and manually reload the files to obtain an updated version. Timestamps can be ignored entirely and only the ability to 'window' data to a specific time period will be lost.

KML clients that do support "Network Links", such as *Google Earth* or *ArcGIS Explorer*, will continue to receive updated data even as users explore the telemetry. Telemetry is presented as an easy-to-understand color-coded scatter plot on the map, and each point also has a timestamp to allow easy navigation or playback. Using KML for the telemetry format also allows the tools being used for data analysis on the ship to be reused to provide mission data to the general public afterwards, as part of outreach efforts.

CHAPTER 6

Conclusions

In his epilogue to *The Silent World*[14], Oceanographic pioneer Jacques Yves Cousteau predicted that technology would eventually allow humans to break the "600 foot barrier" and explore to the edge of the continental shelf. While visiting those depths in person remains a challenge for the most technical of divers, submarines, ROVs, and AUVs have opened the full depths of the ocean to scientists for exploration and research. In this thesis, I have employed techniques from a variety of fields to deliver data from underwater vehicles to the surface, in the hopes of bringing scientists and vehicles even closer together. The complete system presented here has the capability to increase feedback between scientists and underwater vehicles, in a way that will hopefully continue to push exploration forward. In this thesis, I have:

- Identified the need for an end-to-end system capable of delivering science data from in-situ underwater vehicles to surface observers.
- Evaluated compression of scalar time-series data using a variety of lossy and lossless compression methods.
- Used data from recent AUV missions to compare lossy compression of scalar time-series data using the Discrete Wavelet Transform against compression with the Discrete Cosine Transform.
- Described and implemented a method for communicating compressed data to the surface using the Discrete Wavelet Transform.
- Compared the performance of two wavelet-based image compression techniques applicable to very low bitrates using photos captured during recent AUV missions to the results of vector quantization based techniques.

- Presented a novel user interface based on KML, designed to allow non-technical users to investigate telemetry as soon as it is has been telemetered.

Lossy compression methods, especially those based on the Discrete Wavelet Transform (DWT) hold great promise for delivering science data from underwater vehicles to topside observers. The bandwidth requirements for doing so are within conservative limits of current acoustic modems, even when telemetering multiple instruments. Delivering more detailed telemetry from underwater robots to scientists and engineers on the surface could enable new interactions with, and applications for, underwater robotics. This thesis investigates how best to transmit that telemetry, and how to display it once it's received. While there are numerous areas for further investigation, this thesis has presented one complete system for allowing such interactions in addition to the results of it being tested against data from recent AUV missions.

6.1 User Interface

This thesis focused on compressing oceanographic data, and providing that telemetry to surface observers to accelerate mission-level decision making. One area which has not been developed by this thesis is how to allow scientists to respond based upon received telemetry. What aspects of underwater missions should be mutable, and which should be fixed? How should new commands be transmitted to underwater vehicles? My experience remotely controlling AUVs on the AGAVE expedition described in section 3.1 suggests that these are equally important, yet difficult, questions. I believe that answering these questions will prove quite challenging, as, unlike this observation-based work, they will require careful coordination with scientists to ensure that changes to missions still allow oceanographic objectives to be achieved. Philosophical concerns aside, I believe there is great potential for applying traditional compression techniques to command and control telemetry as well. Missions, as previously mentioned, are typically high-level programs consisting of a short sequence of steps, each step containing a few numerical parameters. Development of a Huffman-coded representation for simple pre-programmed mission descriptions could allow even an entire mission to be transmitted in seconds.

Users should be able, additionally, to request higher detail for telemetry of interest. After receiving a timestamp, the underwater vehicle could send much higher resolution data

covering a shorter time period. For image telemetry, specific images could be requested, or thumbnails transmitted to allow a better overview of what is available. The wavelet-based compression methods presented in this thesis allow for progressive transmission; greater detail is transmitted with each byte, gradually increasing image quality. For images that were of particular interest, additional packets could be transmitted to refine the details seen in an initial image. I have begun development of a user interface that allows users to request greater detail for specific areas of telemetry, and presents some telemetry in greater detail. While this would not replace standard KML clients as the primary interface for telemetry, it would allow vehicle operators to interact more closely with the vehicle and monitor it more carefully.A prototype of the new user interface is shown in Figure 6-1.

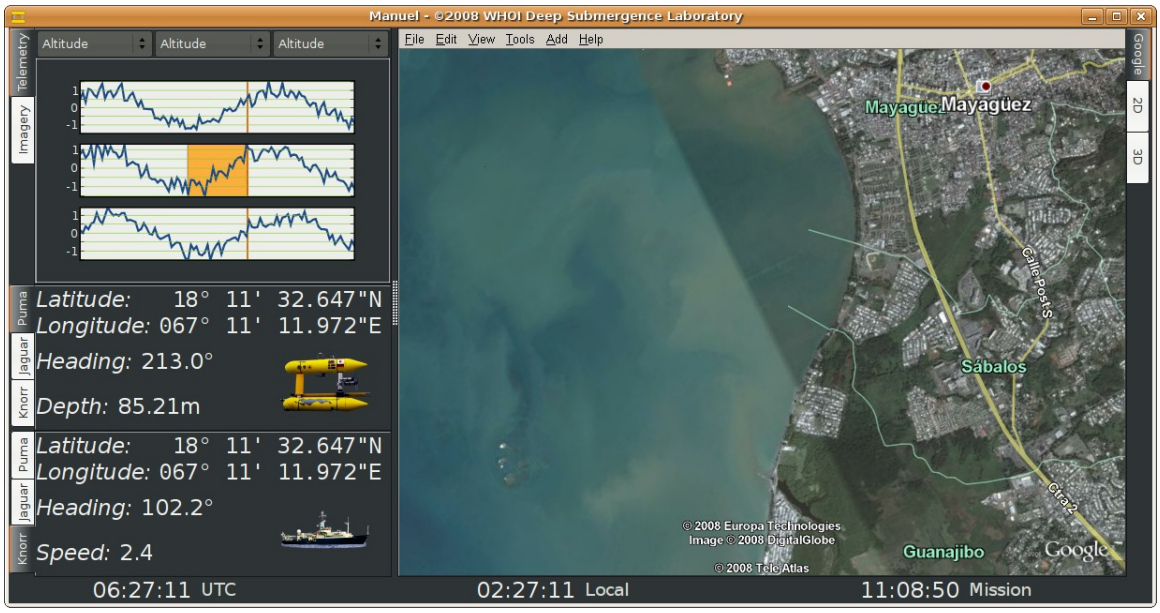


Figure 6-1: Prototype of future integration between Google Earth and other topside monitoring systems.

6.2 Data Compression

Lossy compression methods can clearly be applied to deliver useful science telemetry over extremely low bandwidth connections. This thesis has presented the foundations of a system capable of compressing both time-series and photographic oceanography data, but there are a number of other forms of data that have not been considered. Multibeam sonar imagery, or highly correlated vectors of values from instruments like an Acoustic Doppler Current

Profiler (ADCP), may benefit from different techniques than those presented.

In real AUV deployments, communication may not be constantly available. Transducers frequently need to be removed from the water when ships move, and seafloor bathymetry may simply prevent communication with underwater vehicles at times. Rather than selecting purely the largest coefficients, a telemetry system could weight coefficients by their age; old data would be transmitted for several TDMA cycles, but the most recent data would have a higher chance of including detail coefficients. Very large discontinuities would still be shown in old data, but recent data would be presented in greater detail.

While the scalar telemetry in this thesis was compressed using Daubechies wavelets, a formal study of wavelet families based on previously acquired data could help identify optimal wavelets for different sensors. Some time-series, such as X or Y position, may be smooth enough that a DCT representation is in fact superior to the wavelet representation used in this thesis. Quantization methods also have room for further study, identifying which methods provide the best compression with least error.

APPENDIX A

Wavelet Coding Source Code

telemeter.py

```
#!/usr/bin/env python
# encoding: utf-8
# This software is provided under the MIT Open Source License. For details,
# see the code ending, or http://www.opensource.org/licenses/mit-license.php
#
# In addition to the halffloat module supplied with this code (and its
# dependency on libILMbase), the following Python packages are required to run:
# * Numerical Python (numpy)
# * Scientific Python (scipy)
# * PyWavelets (pywt)
from numpy import *
import scipy.io
import math
import pywt
import halffloat

"""telemeter.py: A simple implementation of lossy wavelet-based encoding and
    decoding for AUV telemetry.\nNOTE: This code is not endian-safe; it was
    written for x86 processors."""

__version__    = "1.0"
__author__     = "Chris Murphy <cmurphy@whoi.edu>"
__copyright__  = "(c) 2008, Chris Murphy / Woods Hole Oceanographic Institution"
__license__    = "MIT"
__credits__    = "%s, Released with %s license." % (__copyright__, __license__ )
__date__       = "29 July 2008"

def __concatenatebits(integers, n_bits):
  """__concatenatebits(integers, n_bits) --> byte_array\n
  Returns an array of bytes, consisting of each value in 'integers' quantized
  to 'n_bits' bits and concatenated."""
  if any([x > 2**n_bits-1 for x in integers]):
    raise OverflowError("Not enough bits to represent "
                        "all values in iterable!")
  def bitify(x):
    return [(x & (1 << i)) > 0 for i in range(n_bits-1,-1,-1)]
  bools = concatenate(map(bitify, integers))
  return scipy.io.packbits(bools)

def __splitbits(bytes, n_bits):
  """ __splitbits(bytes, n_bits) --> integers\n
  Returns an array of integers, undoing the operation performed in
  __concatenatebits. Note that some extra values may be returned, as it can be
```

```
    unclear where the integer stream ends."""
    bits = scipy.io.unpackbits(bytes, 8*len(bytes))
    ii = range(n_bits, 8*len(bytes), n_bits)
    arrs = split(bits, ii)
    def unbitify(x):
      return sum([(1 << i) for i,val in enumerate(x[::-1]) if val])
    return map(unbitify, arrs)

def __n_index_bits(max_value):
    """__n_index_bits(max_value) -> n_bits\n
    Return the number of bits needed to represent values up to max_value."""
    return int(math.log(max_value, 2))+1

def wave_encode(t_pkt, x_pkt, opts):
    """wave_encode(t, x, config) -> encoded_bytes\n
    Wavelet-encodes a set of timeseries data as a fixed-length bytestring for
    transmission."""

    ### Fill the packet header ###
    # The time header consists of two items stuffed into one 24b unsigned integer.
    # The lower 21 bits represent the time of day in 20ths of a second. The top 3
    # bits determine the sample rate from among a predefined set.
    N      = len(x_pkt)
    t_end = (t_pkt[-1] % 86400.) # 24 hr/day * 60 min/hr * 60 sec/min = 86400.
    time_header = uint32(round(t_end * 20)) # 20 20ths of a second/sec
    t_step = median(diff(t_pkt)) # FIXME - hacky way to find t_step.
    idx = argmin(abs(t_step - opts['sample_rates']))
    time_header += idx << 21

    header = [uint8((time_header & 0x0000ff)),         # ending time and ...
              uint8((time_header & 0x00ff00) >> 8), # sample rate of ...
              uint8((time_header & 0xff0000) >> 16), # the dataset.
              uint16(N)]              # Number of total input datapoints

    ### Figure out which detail coefficients to encode ###
    # Wavelet Transform Data
    coeffs = pywt.wavedec(x_pkt, opts['wavelet'], opts['ext_mode'])
    # Calculate number of detail coeffs which can be sent with this packet length
    det_coeffs = concatenate(coeffs[1:])
    nbits = __n_index_bits(len(det_coeffs))
    n_bytes = opts['enc_length'] - 5 - len(coeffs[0])*4
    n_coeffs = int(n_bytes * 8 // (16+nbits))
    # Is header too big for packet? Assert that we can encode anything at all.
    assert(n_coeffs > 0)
    # Find the n_coeffs highest absolute value coefficients.
    ii = argsort(-abs(det_coeffs)) < n_coeffs

    ### Quantize detail coefficients and offsets ###
    if opts['quantization'] == 'halffloat':
      quantized = array([halffloat.bits(x) for x in det_coeffs[ii]], dtype=uint16)
    elif opts['quantization'] == '-1to1fixed16':
      inc_coeffs = det_coeffs[ii]
      quantized = uint16(round((inc_coeffs+1.) / 2. * 65535.))
    else:
      raise Exception('Unknown quantizer specified!')
    cmpindices = array(list(__concatenatebits(where(ii)[0], nbits)))

    print ">>> ", 4*coeffs[0].shape[0], cmpindices.shape[0], 2*quantized.shape[0]
```

```
    ### Pack data and convert everything to final representation ###
    # NOTE: Approx. coefficients could probably be packed better than as Float32's
    data = [float32(coeffs[0]),     # Approximation coefficients
            uint8(cmpindices),      # Compressed indices of included detail coeff's
            uint16(quantized)]      # Quantized detail coefficients

    # Convert lists of typed (uint8, uint16) values into bytestrings, concatenate.
    header = "".join(map(lambda x: x.tostring(), header))
    data = "".join(map(lambda x: x.tostring(), data))
    return "".join((header, data))

def wave_decode(packet, opts):
    """wave_decode(encoded_bytes, config) -> (time, data)\n
    Decode a wavelet-encoded bytestring, and return a 2-tuple (t, x) of the
    decoded section of timeseries data."""
    # N.B. for source code readers: wave_encode is easier to understand; read that
    # first, and understand that this is doing the inverse.

    ### Decode the packet header ###
    # Pull out the 24-bit time header, and decode it
    time_head = fromstring(packet[0:3]+"\x00", dtype=uint32)
    t_step = opts['sample_rates'][time_head >> 21]
    t_end = (0x1ffff & time_head) / 20.
    # Calculate array length for each wavelet level, for input of length N.
    # HACK: shouldn't have to run pywt.wavedec!
    N = fromstring(packet[3:5], dtype=uint16)
    coeff_arr_lens = map(len, pywt.wavedec(zeros(N), opts['wavelet'],
                                           opts['ext_mode']))
    ### Get Approximation coeffs, and create/fill array of detail coeffs.
    approx_coeff_end = 5 + coeff_arr_lens[0]*4
    approx_coeffs = fromstring(packet[5:approx_coeff_end], dtype=float32)
    # Create an array to store the detail coefficients
    det_coeffs = zeros(sum(coeff_arr_lens[1:]), dtype=float32)
    # Figure out how many detail coefficients were sent.
    bits_for_coeffs = 8*len(packet[approx_coeff_end:])
    nbits = __n_index_bits(len(det_coeffs))
    n_coeff_sent = bits_for_coeffs // (nbits + 16)
    # Get + decompress detail coefficient indices
    nbytes = int(math.ceil(nbits * n_coeff_sent / 8.))
    ind_end = approx_coeff_end+nbytes
    cmpinds = fromstring(packet[approx_coeff_end:ind_end], dtype=uint8)
    indices = list(__splitbits(cmpinds, nbits))[:n_coeff_sent]
    # Dequantize the coefficients
    coeff_uint16s = fromstring(packet[ind_end:], uint16)
    if opts['quantization'] == 'halffloat':
        coeffs = array(map(halffloat.half, coeff_uint16s))
    elif opts['quantization'] == '-1to1fixed16':
        coeffs = coeff_uint16s * 2. / 65535. - 1
    else:
        raise Exception('Unknown quantizer specified!')
    assert(coeffs.shape[0] == len(indices))
    # populate detail coefficient array with (sparse!) data
    for i, ind in enumerate(indices):
        det_coeffs[ind] = coeffs[i]
    # Split single detail coefficients array into multiple level-specific arrays
    arr_inds = cumsum(coeff_arr_lens)
    coeff_arrays = split(det_coeffs,
                         map(lambda x:x-len(approx_coeffs), arr_inds[1:-1]))
    # Prepend the approximation coefficients to the list of arrays
```

```
    coeff_arrays.insert(0, approx_coeffs)
    ## Perform wavelet recomposition, and calculate timestamps
    xi = pywt.waverec(coeff_arrays, opts['wavelet'], opts['ext_mode'])
    t_start = t_end - t_step * (len(xi)-1) - 0.5 * t_step
    ti = arange(t_end, t_start, -t_step)[::-1]
    return ti, xi

if __name__ == '__main__':
    print "<< Running brief lossy wavelet coding demo. >>"

    config = {
      'wavelet'       : pywt.Wavelet('db2'),
      'ext_mode'      : pywt.MODES.cpd,                    # Pad with edge values
      'sample_rates'  : [0.05, 0.1, 0.25, 0.5, 1, 2, 4, 10], # (seconds)
      'quantization'  : 'halffloat',
      'enc_length'    : 62
    }

    ### Generate 1/2 hr of 2Hz 'random' data
    t = arange(0, 1800, 0.5)
    # Start with some sinusoids
    x = (50 * sin(t/600.)) - (5 * sin(t/50. + 0.4))
    # Add some noise
    x += random.normal(0, 2, len(x))
    # Add a couple of interesting spikes
    x += minimum(400 / (0.6+abs(500-t)**0.6), 100)
    x -= minimum(300 / (0.6+abs(950-t)**0.3), 100)

    ### Encode and Decode the data
    packet = wave_encode(t, x, config)
    t_out, x_out = wave_decode(packet, config)

    ### Plot
    import pylab
    pylab.plot(t, x, 'r.', label='Original Data')
    pylab.plot(t_out, x_out, 'b.', label='Telemetered Data')
    pylab.title(u'Lossy Wavelet Encoding Demo - (c)2008, Chris Murphy')
    pylab.xlabel('Time')
    pylab.ylabel('Value')
    pylab.legend()
    pylab.grid()
    pylab.show()

#################### License ####################
# The MIT License
# Copyright (c) 2008 Chris Murphy, Woods Hole Oceanographic Institution
#
# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to
# deal in the Software without restriction, including without limitation the
# rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
# sell copies of the Software, and to permit persons to whom the Software is
# furnished to do so, subject to the following conditions:
#
# The above copyright notice and this permission notice shall be included in
# all copies or substantial portions of the Software.
#
# The software is provided "as is", without warranty of any kind, express or
# implied, including but not limited to the warranties of merchantability,
```

```
# fitness for a particular purpose and noninfringement. In no event shall the
# authors or copyright holders be liable for any claim, damages or other
# liability, whether in an action of contract, tort or otherwise, arising from,
# out of or in connection with the software or the use or other dealings in
# the software.
##########################################################
```

halffloat.cpp

```
#include "python2.5/Python.h"
#include <OpenEXR/half.h>
#include <stdlib.h>
#include <stdio.h>

/****
 * This software is provided under the MIT Open Source License. For details,
 * see the code ending, or http://www.opensource.org/licenses/mit-license.php
 ****/

/****
 * Functions for converting to / from half-precision floating-point values.
 *
 * This is a partial binding against the OpenEXR support provided by libILMbase.
 *
 * This file can be compiled on Linux with:
 *   g++ -lHalf --shared -o halffloat.so halffloat.cpp
 * And then tested in Python with:
 *   import halffloat
 *   from math import *
 *    halffloat.half( halffloat.bits(pi))
 *   --> 3.140625
 *    halffloat.half( halffloat.bits( (e**(-1j*pi)).real ))
 *   --> -1.0
 ****/

static PyObject * half_to_bits(PyObject *self, PyObject *args) {
  float float_in;
  unsigned short bits_out;

  /* Expects one Python Float, returns a C float, otherwise raise exception. */
  if (!PyArg_ParseTuple(args, "f", &float_in)) {
    return NULL;
  } else {
    /* Computer the half-precision representation of the floating value */
    half float16 (float_in);
    bits_out = float16.bits();
  }
  /* Return as Python integer */
  return PyInt_FromLong(bits_out);
}

static PyObject * bits_to_half(PyObject *self, PyObject *args) {
  unsigned short bits_in;
  float float_out;

  /* Takes one Python Int, return a C unsigned int. or raises an exception. */
  if (!PyArg_ParseTuple(args, "H", &bits_in)) {
    return NULL;
  } else {
    /* Compute the floating-point value of these bits as half-precision float */
    half float16 = half();
    float16.setBits(bits_in);
    float_out = (double)float16;
  }
```

```
  /* Return as Python Float */
  return PyFloat_FromDouble(float_out);
}

/* Method definitions */
PyDoc_STRVAR(bits__doc__, "bits(float) -> int\n\n"
  "Returns an unsigned short representing the bits of the floating-point value "
  "represented as a half-precision floating point number.");
PyDoc_STRVAR(half__doc__, "half(int) ->float\n\n"
  "Decodes the lower 16 bits of the integer passed in as a half-precision "
  "float, then returns the value as a Python Float.");

static PyMethodDef HalfFloatMethods[] = {
    {"bits", half_to_bits, METH_VARARGS, bits__doc__},
    {"half", bits_to_half, METH_VARARGS, half__doc__},
    {NULL, NULL, 0, NULL} /* Sentinel */
};

/* Useful constants */
static void set_constants(PyObject *mod) {
  PyObject *c;
  c = PyFloat_FromDouble(HALF_MIN);
  PyObject_SetAttrString(mod, "min", c);
  Py_DECREF(c);
  c = PyFloat_FromDouble(HALF_MAX);
  PyObject_SetAttrString(mod, "max", c);
  Py_DECREF(c);
  c = PyFloat_FromDouble(HALF_NRM_MIN);
  PyObject_SetAttrString(mod, "nrm_min", c);
  Py_DECREF(c);
  c = PyFloat_FromDouble(HALF_EPSILON);
  PyObject_SetAttrString(mod, "epsilon", c);
  Py_DECREF(c);
}

/* Module Initialization */
PyDoc_STRVAR(halffloat__doc__,
             "Provides support for half-precision floating-point (Float16) "
             "numbers. Binding for (and therefore requires) libILMbase.");
PyMODINIT_FUNC inithalffloat(void) {
  PyObject *m = Py_InitModule3("halffloat", HalfFloatMethods, halffloat__doc__);
  set_constants(m);
}

/****************** License ******************
* The MIT License
* Copyright (c) 2008 Chris Murphy, Woods Hole Oceanographic Institution
*
* Permission is hereby granted, free of charge, to any person obtaining a copy
* of this software and associated documentation files (the ''Software''), to
* deal in the Software without restriction, including without limitation the
* rights to use, copy, modify, merge, publish, distribute, sublicense, and/or
* sell copies of the Software, and to permit persons to whom the Software is
* furnished to do so, subject to the following conditions:
*
* The above copyright notice and this permission notice shall be included in
* all copies or substantial portions of the Software.
*
* The software is provided ''as is'', without warranty of any kind, express or
```

```
 * implied, including but not limited to the warranties of merchantability,
 * fitness for a particular purpose and noninfringement. In no event shall the
 * authors or copyright holders be liable for any claim, damages or other
 * liability , whether in an action of contract, tort or otherwise, arising from,
119 * out of or in connection with the software or the use or other dealings in
 * the software.
 ****************************************/
```

Bibliography

[1] Ian F. Akyildiz, Dario Pompili, and Tomasso Melodia. Underwater Acoustic Sensor Networks: Research Challenges. In *Ad Hoc Networks*, volume 3, pages 257 – 279. Elsevier, March 2005.

[2] Francesc Altet, Ivan Vilata, Scott Prater, Vicent Mas, Tom Hedley, Antonio Valentino, Jeffrey Whitaker, et al. PyTables: Hierarchical datasets in Python, 2002–.

[3] Roy A. Armstrong, Hanumant Singh, Juan Torres, Richard S. Nemeth, Ali Can, Chris Roman, Ryan Eustice, Lauren Riggs, and Graciela Garcia-Moliner. Characterizing the deep insular shelf coral reef habitat of the hind bank marine conservation district (us virgin islands) using the seabed autonomous underwater vehicle. *Continental Shelf Research*, 26(2):194–205, February 2006.

[4] David Ascher, Paul F. Dubois, Konrad Hinsen, Jim Hugunin, and Travis Oliphant. *Numerical Python*. Lawrence Livermore National Laboratory, Livermore, California, USA, 2001. Available at http://www.pfdubois.com/numpy/.

[5] Robert D. Ballard, Dana R. Yoerger, W. Kenneth Stewart, and Andy Bowen. ARGO/JASON: a remotely operated survey and sampling system for full-ocean depth. In *OCEANS, 1991. Proceedings of MTS/IEEE*, pages 71–75, 1991.

[6] Jim Bellingham, Max Deffenbaugh, John J. Leonard, and Josko Catipovic. Arctic under-ice survey operations. *Unmanned Systems*, 12:24–29, 1994.

[7] Teledyne Benthos. Telesonar underwater acoustic modems, August 2008. http://www.benthos.com/pdf/telesonar2007.pdf.

[8] Andy Bowen, Dana Yoerger, and Louis Whitcomb. Hybrid ROV for 11,000 Meter Operations. In *Symposium on Underwater Technology and Workshop on Scientific Use of Submarine Cables and Related Technologies*, April 2007.

[9] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Research Report 124, Digital Equipment Corporation Systems Research Center, May 1994.

[10] Martin Burtscher and Paruj Ratanaworabhan. High throughput compression of double-precision floating-point data. In *2007 Data Compression Conference (DCC'07)*, 2007.

[11] Richard Camilli, Brian Bingham, Michael V. Jakuba, Hanumant Singh, and Jean Whelan. Integrating in-situ chemical sampling with auv control systems. In *Oceans 2004, Proceedings of the MTS/IEEE*, Kobe, Japan, 2004.

[12] Richard Camilli and Anthony Duryea. Characterizing marine hydrocarbons with in-situ mass spectrometry. In *Oceans 2007, Proceedings of the MTS/IEEE*, pages 1–7, Oct 2007.

[13] Richard Camilli, Oscar Pizarro, and Luis Camilli. Rapid swath mapping of reef ecology and associated water column chemistry in the gulf of chiriqui, panama. In *Oceans 2007, Proceedings of the MTS/IEEE*, pages 1–8, Oct 2007.

[14] Captain Jacques Yves Cousteau and Frédéric Dumas. *The Silent World*. Time Incorporated, 1950.

[15] L Peter Deutsch. RFC 1951: DEFLATE compressed data format specification version 1.3, May 1996. Status: INFORMATIONAL.

[16] R. A. DeVore and B. J. Lucier. Wavelets. In *Acta Numerica 1*, pages 1–56. Cambridge University Press, 1992.

[17] Jose A. Diaz and Raul E. Torres. Classification of underwater color images with applications in the study of deep coral reefs. In *Circuits and Systems, 2006. MWSCAS '06. 49th IEEE International Midwest Symposium on*, volume 2, pages 610–613, San Juan, PR, August 2006.

[18] David L. Donoho, Martin Vetterli, R.A. DeVore, and Ingrid Daubechies. Data Compression and Harmonic Analysis. *IEEE Transactions on Information Theory*, 44(6), October 1998.

[19] Robert L. Eastwood, Lee E. Freitag, and Josko A. Catipovic. Compression techniques for improving underwater acoustic transmission of images and data. In *Proceedings of MTS/IEEE OCEANS 1996: Prospects for the 21st Century*, Fort Lauderdale, FL, September 1996.

[20] B. Elder, A.D. Bowen, M. Heintz, M. Naiman, C. Taylor, W. Seller, J.C. Howland, and L.L. Whitcomb. Jason 2: a review of capabilities. In *EOS: Trans. Amer. Geophysical Union Fall Meeting Supplement*, 2003. Abstract.

[21] B.P. Foley, K. DellaPorta, D. Sakellariou, B. Bingham, R. Camilli, R. Eustice, D. Evagelistis, V. Ferrini, M. Hansson, K. Katsaros, D. Kourkoumelis, A. Mallios, P. Micha, D. Mindell, C. Roman, H. Singh, D. Switzer, and T. Theodoulou. New methods for underwater archaeology: the 2005 chios ancient shipwreck survey. *Hesperia*. Accepted, To Appear.

[22] Lee Freitag, Matt Grund, Sandipa Singh, James Partan, Peter Koski, and Keenan Ball. The WHOI Micro-Modem: An Acoustic Communcations and Navigation System for Multiple Platforms. In *Oceans 2005, Proceedings of the MTS/IEEE*, volume 2, pages 1086 – 1092, 2005.

[23] HDF Group. HDF5, 2008. http://hdf.ncsa.uiuc.edu/HDF5/index.html.

[24] WHOI Acoustic Communications Group. WHOI Acoustic Communications: Micro-Modem, July 2008. http://acomms.whoi.edu/umodem/.

[25] David F. Hoag and Vinay K. Ingle. Underwater image compression using the wavelet transform. In *OCEANS '94. Proceedings*, volume 2, pages 533–537, Sep 1994.

[26] David F. Hoag, Vinay K. Ingle, and Richard J. Gaudette. Low-bit-rate coding of underwater video using wavelet-based compression algorithms. *IEEE Journal of Oceanic Engineering*, 22(2):393–400, April 1997.

[27] David A. Huffman. A method for the construction of minimum-redundancy codes. In *Proceedings of the Institute of Radio Engineers*, volume 40, pages 1098–1101, September 1952.

[28] Matthew Johnson-Roberson, Suresh Kumar, and Stefan Williams. Segmentation and classification of coral for oceanographic surveys: A semi-supervised machine learning approach. In *Oceans 2006 – Asia Pacific*, Singapore, May 2006.

[29] Jeff W. Kaeli, Hanumant Singh, and Roy A. Armstrong. An automated morphological image processing based methodology for quantifying coral cover in deeper-reef zones. In *Oceans 2006, Proceedings of the MTS/IEEE*, pages 1–6, Boston, September 2006.

[30] James C. Kinsey and Louis L. Whitcomb. Preliminary field experience with the DVL-NAV integrated navigation system for oceanographic submersibles. *Control Engineering Practice*, 12(12):1541–1548, December 2004. Invited Paper.

[31] Jon Kristensen and Karstein Vestgard. Hugin-an untethered underwater vehicle for seabed surveying. In *OCEANS, 1998. Proceedings of MTS/IEEE*, volume 1, pages 118–123, September 1998.

[32] Clayton Kunz, Chris Murphy, Hanumant Singh, Claire Willis, Robert A. Sohn nd Sandipa Singh, Taichi Sato, Koichi Nakamura, Michael Jakuba, Ryan Eustice, Richard Camilli, and John Bailey. Toward extraplanetary under-ice exploration: Robotic steps in the arctic. *Journal of Field Robotics Special Issue on Space Robotics*, submitted.

[33] Peter Lindstrom and Martin Isenburg. Fast and efficient compression of floating-point data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5), September 2006.

[34] Inc. LinkQuest. Linkquest underwater acoustic modems, uwm3000h specifications, August 2008. http://www.link-quest.com/html/uwm3000h.htm.

[35] Stephen C. Martin, Louis L. Whitcomb, Roland Arsenault, Matthew Plumlee, and Colin Ware. A system for real-time spatio-temporal 3-d data visualization in underwater robotic exploration. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (ICRA)*, Barcelona, Spain, April 2005.

[36] P. H. Milne. *Underwater Acoustic Positioning Systems*. Gulf Publishing Co., Houston, TX, 1983.

[37] Koichi Nakamura. Personal communications, 2008.

[38] Open Geospatial Consortium. KML Standard. http://www.opengeospatial.org/standards/kml/.

[39] Alan V. Oppenheim, Roland W. Schafer, and John R. Buck. *Discrete-Time Signal Processing*, chapter 8. Prentice-Hall, 2 edition, 1999.

[40] Costas Pelekanakis, Milica Stojanovic, and Lee Freitag. High rate acoustic link for underwater video transmission. *OCEANS 2003. Proceedings*, 2:1091–1097, Sep 2003.

[41] John G. Proakis. *DigitalCommunications*. McGraw-Hill, 4 edition, 2000.

[42] Mark Probst. Metapixel, 1999–2008. http://www.complang.tuwien.ac.at/schani/.

[43] Paruj Ratanaworabhan, Jian Ke, and Martin Burtscher. Fast lossless compression of scientific floating-point data. In *2006 Data Compression Conference (DCC'06)*, 2006.

[44] Subhasis Saha. Image Compression - from DCT to Wavelets : A Review. *ACM Crossroads: Data Compression*, Spring 2000.

[45] Amir Said and William A. Pearlman. A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *Circuits and Systems for Video Technology, IEEE Transactions on*, 6(3):243–0250, Jun 1996.

[46] David Salomon. *Data Compression: The Complete Reference*, chapter 4.5. Springer, 4 edition, 2007.

[47] Craig Sayers, Angela Lai, and Richard Paul. Visual imagery for subsea teleprogramming. In *Proc. IEEE Robotics and Automation Conference*, 2005.

[48] Aaron Schwartz et al. WebPy, 2006–. http://www.webpy.org.

[49] H. Singh, A. Can, R. Eustice, S. Lerner, N. McPhee, O. Pizarro, and C. Roman. Seabed AUV offers new platform for high-resolution imaging. *EOS, Transactions of the AGU*, 85(31):289,294–295, August 2004.

[50] Sandipa Singh, Matthew Grund, Brian Bingham, Ryan Eustice, Hanumant Singh, and Lee Freitag. Underwater Acoustic Navigation with the WHOI Micro Modem. In *Proc. IEEE/MTS Oceans Conference and Exhibition*, Boston, USA, 2006.

[51] Robert Sohn, Claire Willis, Susan Humphris, Timothy Shank, Hanumant Singh, Henrietta Edmonds, Clayton Kunz, Ulf Hedman, Elisabeth Helmke, Michael Jakuba, Bengt Liljebladh, Julia Linder, Christopher Murphy, Koichi Nakamura, Taichi Sato, Vera Schlindwein, Christian Stranne, Maria Tausenfreund, Lucia Upchurch, Peter Winsor, Martin Jakosson, and Adam Soule. Explosive volcanism on the ultraslow-spreading Gakkel ridge, Arctic Ocean. *Nature*, 453(7199):1236–1238, June 2008.

[52] Milica Stojanović. Recent advances in highspeed underwater acoustic communications. *IEEE Journal of Oceanic Engineering*, 21(2):125–136, April 1996.

[53] Milica Stojanović. On the relationship between capacity and distance in an underwater acoustic communication channel. In *SIGMOBILE Mobile Computing and Communications Review*, volume 11, pages 34 – 43, October 2007.

[54] Roger Stokey, Lee Freitag, and Matthew Grund. A Compact Control Language for auv acoustic communication. *Oceans 2005 - Europe*, 2:1133–1137, June 2005.

[55] Roger P. Stokey. A compact control language for autonomous underwater vehicles. Specification, Woods Hole Oceanographic Institution, April 2005. http://acomms.whoi.edu/40x%20Specifications/401100%20Compact%20Control%20Language/CCL%20April%202005%20Public%20Release%201.0.pdf.

[56] Roger P. Stokey, Alexander Roup, Chris von Alt, Ben Allen, Ned Forrester Tom, Austin, Rob Goldsborough, Mike Purcell, Fred Jaffre, Greg Packard, and Amy Kukulya. Development of the remus 600 autonomous underwater vehicle. In *OCEANS, 2005. Proceedings of MTS/IEEE*, volume 2, pages 1301–1304, Washington, 2005.

[57] Ronald A Stone. Roughness and reflection in machine vision. Technical Report CMU-RI-TR-94-41, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 1994.

[58] Gilbert Strang and Truong Nguyen. *Wavelets and Filter Banks*, chapter 8. Wellesley-Cambridge Press, 1996.

[59] M. Suzuki, T. Sasaki, and T. Tsuchiya. Digital acoustic image transmission system for deep-sea research submersible. *Oceans 1992, Proceedings of the MTS/IEEE*, 2:567–570, Oct 1992.

[60] Nicholas Tolimieri, M. E. Clarke, Hanumant Singh, and Chris Goldfinger. Development of the seabed auv to monitor populations of westcoast groundfish in untrawlable habitat: an example with rosethorn rockfish, sebastes helvomaculatus. In Press.

[61] Robert J. Urick. *Principles of Underwater Sound*. McGraw-Hill, 1975.

[62] Guido van Rossum and Fred L. Drake, editors. *Python Reference Manual*. PythonLabs, Virginia, USA, 2001. Available at http://www.python.org.

[63] Martin Vetterli. Wavelets, approximation, and compression. *IEEE Signal Processing Magazine*, pages 59 – 73, September 2001.

[64] Christopher von Alt. Autonomous Underwater Vehicles. In *Autonomous and Lagrangian Platforms and Sensors*, 2003.

[65] Gregory K. Wallace. The jpeg still picture compression standard. *Communications of the ACM*, 34(4):30–44, 1991.

[66] Colin Ware, Matthew Plumlee, Roland Arsenault, Larry Mayer, and Shep Smith. Geozui3d: Data fusion for interpreting oceanographic data. In *MTS/IEEE Oceans 2001 Conference and Exhibition*, volume 3, pages 1960–1964, Honolulu, HI, November 2001.

[67] Filip Wasilewski. *PyWavelets – Discrete Wavelet Transform in Python*. Poland, 2006. Available at http://www.pybytes.com/pywavelets/.

[68] Terry A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, June 1984.

[69] Paul Wessell and W.H.F Smith. Free software helps map and display data. In *EOS, Transactions of the AGU*, volume 72, page 441, 1991.

[70] ISO/IEC JTC1/SC29 WG1. Jpeg 2000 part i final committee draft version 1.0. Specification, Joint Photographic Experts Group, April 2000.

[71] Wikipedia. Keyhole markup language — wikipedia, the free encyclopedia, 2008. [Online; accessed 3-August-2008].

[72] Dana R. Yoerger, Michael V. Jakuba, Albert M. Bradley, and Brian Bingham. Techniques for deep sea near bottom survey using an autonomous vehicle. *International Journal of Robotics Research*, 26(1):41–54, 2007.

[73] Jacob Ziv and Abraham Lempel. A universal algorithm for data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, May 1977.