DESIGN AND IMPLEMENTATION OF AN ENERGY AWARE PROGRAMMING FRAMEWORK FOR AUTONOMOUS UNDERWATER VEHICLES

BY HANS CHRISTIAN WOITHE

A dissertation submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

in partial fulfillment of the requirements

for the degree of

Doctor of Philosophy

Graduate Program in Computer Science

Written under the direction of

Ulrich Kremer

and approved by

_____

_____

_____

_____

_____

New Brunswick, New Jersey

May, 2014

ABSTRACT OF THE DISSERTATION

Design and Implementation of an Energy Aware Programming Framework for Autonomous Underwater Vehicles

by Hans Christian Woithe

Dissertation Director: Ulrich Kremer

Autonomous underwater vehicles (AUVs) have become an indispensable tool for studying the oceans. They allow for the prolonged presence of scientific instruments in the ocean, enabling the collection of samples for several weeks or months at a time for a fraction of the cost of research vessels. These vehicles share common characteristics and constraints with other cyber-physical systems that include concerns for vehicle safety, a limited energy supply, the optimization and trade-off of resources, sporadic communication, and operation in extremely constrained environments. One such AUV is the Slocum Electric Glider. Although AUVs like the Slocum Glider have revolutionized the field of oceanography, many are difficult to program and thus limit their overall utility.

A new energy aware, domain specific programming framework for AUVs, called ALGAE (AUV Language for Greater Adaptability and Energy optimization), has been developed on the Slocum Glider. This framework enables scientists to easily create missions that use domain specific features to make trade-offs, such as sacrificing the

quality at which the environment is sampled for a gain in vehicle endurance. Novel methods used in the framework make the vehicle a more effective scientific instrument. The system was specifically designed to support a mission critical platform that operates in an extremely constrained environment. In the new infrastructure, missions can be tested in simulation, but more importantly, can be compiled directly for use on the target platform. To evaluate the framework, simulations and field trials off the coast of New Jersey were performed to showcase the practicality of the system. Furthermore, because the framework was designed around a common set of constraints and characteristics, the mechanisms and approaches developed are widely applicable to many autonomous systems.

Preface

Portions of this dissertation are based on work previously published or submitted for publication by the author [Woithe and Kremer, 2009; Woithe et al., 2010; Woithe and Kremer, 2010, 2011b,a; Woithe et al., 2011; Eichhorn et al., 2012; Woithe et al., 2012, 2013].

Acknowledgements

I would like to thank the committee and my advisor Ulrich Kremer. Although this work on gliders has been very laborious, it has also been very rewarding. I am grateful that throughout these years I have had the support and guidance of my advisor. I am certain that I have yet to fully grasp and appreciate all the ways that you have affected my life. It will likely take years of reflection to fully understand.

I would also like to thank my EEL lab mates Denitsa Tilkidjieva, Jerry Hom, John McCabe, and Pradip Hari for their help and support throughout my research and graduate school experience. I want to also thank my friends and collaborators at the department of Marine and Coastal Sciences and WinLAB of Rutgers University. Without their help and support much of this work would not be possible. I am also very grateful to my many friends for their love, support and great times. I hope the relationships we have built will last the rest of our lives.

Finally, I would like to thank all of my family, especially my wife Francesca. We have already spent much of our life happily together. Our first years in graduate school also marked our first years as a young couple on our own. Together, we have endured the frustrations of research and the joys of our accomplishments. We have both changed and grown so much over these years, but we will leave as we entered, side-by-side. Word cannot express how proud I am of you and how blessed I am to have you as my partner in life. Thank you.

Dedication

To my parents and my wife Francesca.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, autonomous underwater vehicles (AUVs) have become an indispensable tool for marine scientists to learn more about the world's oceans. Traditionally, the acquisition of oceanographic data involved lowering sensors from surface vessels. AUVs have replaced this laborious process and are capable of gathering orders of magnitude more data for a fraction of the overall cost [Creed et al., 2004; Schofield et al., 2007]. Not only are they more cost efficient, but they enable data to be collected in environments that were historically inaccessible or too dangerous [Kunz et al., 2008a].

One such underwater vehicle is the Slocum Electric Glider, illustrated in Figure 1.1, which is used as the implementation platform in this work. The vehicle is developed by Teledyne Webb Research (TWR). Unlike propeller driven vehicles [Hydroid, LLC.; Kunz et al., 2008b; Schulz et al., 2005, 1997], it belongs to a class of AUVs which achieve forward propulsion by changing its buoyancy [Teledyne Webb Research; Sherman et al.,

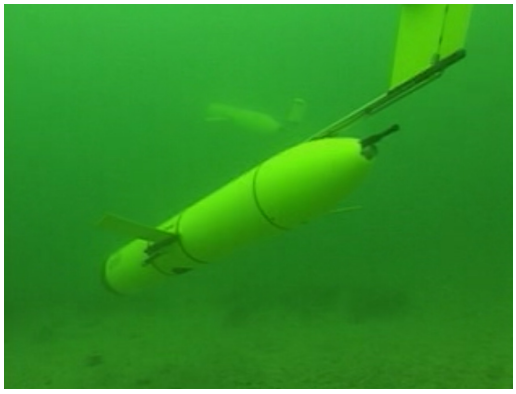


Figure 1.1: Two Slocum Gliders performing a sequence of dives and climbs.

2001; Eriksen et al., 2001; Davis et al., 2002]. The Slocum accomplishes this by moving a piston to take in and expel water. The pitch created by the change in buoyancy can be refined by adjusting the glider's center of gravity through the movement of an internal battery pack. The vehicle's wings and control services along with the change in buoyancy and center of gravity result in a sawtooth forward trajectory of approximately 35 cm/s [Graver et al., 2003]. Because the Slocum only operates its buoyancy engine at inflections points, it can achieve efficient flight lasting weeks to months compared to days or weeks for its propeller driven counterparts. Although efficient, glider operation relies solely on battery power like other autonomous systems, so managing this limited resource is of utmost importance.

The software control systems used to program many cyber-physical systems (CPSs), including the Slocum Glider AUV, are based on the layered control system [Brooks, 1986; Bellingham and Leonard, 1994; Gat et al., 1998]. Users, like marine scientists, specify the actions they wish the vehicle to perform through a set of *behaviors* written in mission files. Users and their AUVs are therefore limited to functionality provided by the set of behaviors the manufacturer supplies. Writing new behaviors for the AUVs from scratch is a difficult task, especially for non-expert programmers. It is often not clear how existing behaviors, let alone new behaviors, interact with each other within the layered control system. This leads to a programming approach where users generally limit themselves to the modification of existing mission parameters or changing priorities among existing behaviors. Even then, the resulting new mission requires the user to go through a lengthy and inherently unreliable trial-and-error validation process.

Like many AUVs, the Slocum Glider in its current state is also extremely static in that it cannot react dynamically to the environment it is observing. It can only be reprogrammed during its periodic surfacings via satellite or radio communication. This constraint of both limited and periodic communication is not unique to underwater

systems, as land, air, and space vehicles also operate in hostile environments where communication can be hampered. For example, autonomous aerial drones can have limited and intermittent communication with a control station. This is possibly caused by intentional radio silence in enemy territory or by "dead-zones" caused by geological features.

In these various domains (e.g., land, air, sea, and space), phenomena may be short lived and so the opportunity to observe them may have been lost by the time a remote operator is able to instruct the autonomous system to resurvey the area of interest. Finally, the number and complexity of sensors that can be installed and supported to study the environment is limited, in the case of the Slocum, by its two 16 MHz embedded processors. Thus a system is required that allows for the flexibility to dynamically react to the environment while supporting a variety of complex sensors and algorithms.

In order to reach an AUV's full potential, it is also necessary to allow scientists to express their mission objectives at a level of abstraction that makes sense to them. Thus, I have investigated and developed a new energy aware, domain specific programming framework for autonomous underwater vehicles using the Slocum Glider as the initial target platform. This framework allows for the specification of new and dynamic missions. It also includes domain specific features to encourage trade-offs. Because energy is a vital and limited resource, trade-offs concerning energy, such as sacrificing the quality of sensor readings to extend vehicle endurance, have been developed and can easily be expressed. Furthermore, the framework is practical and must safely support a mission critical system that operates in an extremely constrained environment with sporadic communication. Thus, the infrastructure must provide mechanisms to quickly evaluate and debug newly created missions before they are compiled and deployed for the actual vehicle.

Although the implementation is focused on a particular vehicle, many of the concepts and approaches are valid for other CPSs and AUVs. In collaboration with the Monterey Bay Aquarium Research Institute (MBARI), I have also integrated portions of the framework to work with their propeller driven vehicle Tethys. Other autonomous systems also share the same constraints and must be equipped to deal with vehicle safety, optimizations and trade-offs of resources because of a limited energy supply, intermittent communication, extendibility, practicality, and usability so that the technology can be adopted. The infrastructure will continue to be developed to become a heterogeneous CPS programming environment in the future. Even with this focus, this work is a significant contribution to the field as the Slocum Glider community consisting of hundreds of glider customers at research institutions around the world and the United States Navy.

1.1 Contributions

In this dissertation, I present the design and implementation of an energy aware programming infrastructure for autonomous underwater vehicles, with a focus on the Slocum Electric Glider. The main contributions of this dissertation include:

- The implementation of a power measurement infrastructure for the Slocum Glider to measure the power consumption of individual components of the vehicle. This infrastructure has been field tested and deployed in several missions off the coast of New Jersey.

- The creation of energy models for the vehicle using information from components specifications from manufacturers, benchtop measurements using oscilloscopes, and field measurements using the power measurement infrastructure mentioned above. The energy models can estimate the energy used by both previously flown as well as simulated deployments.

- The development of an energy aware simulation framework. The framework consists of two simulators, a model-based simulator and a software port simulator. The modeled simulator is built on several years of glider flight data, while the software port simulator is an adaptation of the glider's flight control software to run on commodity hardware. Furthermore, both simulators are capable of running faster-than-real-time and can present a virtual environment. This infrastructure has been instrumental in fixing vehicle software bugs and algorithms in the Slocum Glider.

- The integration of an advanced computing platform for the Slocum Glider that elevates the vehicle's capabilities to support advanced sensors and algorithms. The platform enables the creation of software services that can interoperate with the existing infrastructure in an energy efficient manner.

- The design and implementation of a domain specific language and compiler suite. This language is one of two tiers of programming supported by the framework. This tier provides a high level of abstraction targeted at non-expert programmers that exposes domain specific features to enable mission trade-offs. Scientists are the individuals who will consume the collected data so they should know best on what trade-offs they will need to make in order for sea trials to remain useful. The compiler uses the high level specification to produce missions capable of being run in the simulation infrastructure and on the actual vehicles. Thus, the system was designed to meet practical and real world constraints.

- The creation of a new service model infrastructure that enables the second tier of programming abstractions and allows flight engineers and programmers to more easily extend the functionality of the system. Users can develop new services in a variety of programming languages and can use existing primitive and advanced

services as building blocks. Moreover, mechanisms have been created to easily expose these services as language constructs using the compiler infrastructure. New services will expand the expressiveness of the domain specific language and will allow non-expert programmers to easily take advantage of the new functionality.

- The exploration of several domain specific applications and challenges that provided key insights and knowledge that influenced the design of the programming framework. Performing the triggering of advanced sensors to reduce the energy consumption of the vehicle while still capturing essential scientific data is such an example and has been field tested. This mechanism has also been integrated into the programming framework and allows users to energy manage sensors.

1.2 Organization

The rest of this dissertation is organized as follows: Chapter 2 discusses related literature in the field of programming frameworks and tools for autonomous underwater vehicles, and energy and performance trade-off systems. Chapter 3 details the infrastructure of the Slocum Glider and its programming issues. In Chapter 4, the additional hardware and software infrastructure integrated into the vehicle to enrich the capabilities of the glider are described and showcased. Chapter 5 presents the power measurement infrastructure that enabled the creation of energy models used by the energy aware simulators described in Chapter 6. Several applications and challenges during the development of the programming framework are presented in Chapter 7. The conclusion and future work are discussed in Chapter 8.

Chapter 2

Related Work

In this chapter, some of the related literature that has inspired aspects of the new programming framework described in this work are presented. In Section 2.1, literature that showcases systems that enable or apply trade-offs, for example, in performance, quality and energy are discussed. Users of the programming framework should use the infrastructure to adjust their missions by making domain specific trade-offs in a similar manner. Finally, in Section 2.2, related programming languages and environments used in the underwater domain are covered and their relation to the presented work is discussed.

2.1 Performance, Quality, and Energy Trade-Off Systems

2.1.1 Turducken and Triage

Mobile devices, such as AUVs, laptops and cellular phones, have limited lifetimes. The power requirements of such devices can vary greatly. The power requirements of PDA's, for example, is an order-of-magnitude smaller than a laptop's, and the requirements of sensors is an order-of-magnitude smaller than a PDAs [Sorber et al., 2005]. These reductions in power come at the price of a decrease in computational capabilities and functionality. The Turducken mobile device architecture addresses this issue and provides full device functionality while maintaining availability and extending the device's lifetime.

In Turducken, several optimized platforms are combined to create a single integrated system that can reduce the energy cost to maintain high levels of consistency. A Turducken system is designed in a strictly hierarchical manner where each subsystem is more powerful than the subsystem below. Each subsystem in this layout is capable of performing a task itself, perform service discovery, suspend if it is not needed to perform tasks, and bring its superior subsystem out of suspend mode if it is needed. For example, in the web cache application of [Sorber et al., 2005], a sensor is used to determine if a WiFi connection is available. If it is, the sensor can wake up a StrongARM based PDA which will fetch expired cached items. Later, when web requests are made from a laptop, the requests are routed and satisfied by the PDA. Thus, lower powered and more specialized platforms can be used in place of more power hungry platforms to perform simple tasks.

The follow on, and similar, work to Turducken is the Triage system [Banerjee et al., 2007]. In Triage a high-power and resource-rich platform in combination with a low-power and resource-constrained platform provide quality of service and energy efficiency. The architecture is broken down into two tiers, where tier-0 employs a low-power platform and tier-1 a high-power platform. In tier-0, requests can be serviced by using local cache information, by local execution, or by passing the request to tier-1 for execution. A profiler in Triage measures the energy requirements of tasks and a scheduler determines when and where the requests should be executed to meet service guarantees. The scheduler may decided, for example, to queue several requests and to keep tier-1 powered off until enough work is collected to justify the tier's power requirements.

A stock Slocum Glider contains two 16 Mhz computing platforms, one for flight control and the other for the collection of scientific data. As part of the new framework, the capabilities of the AUV have been extended with the integration of a Linux single board computer. The software on the glider's computing platforms have also

been retrofitted with a scripting framework that allows for the execution of programs without the need to flash new firmware. Together, these systems can create hierarchical power management techniques similar to those described in Turducken and Triage. For example, the low-power flight controller can be used to collect and filter data until enough information has been collected to justify the use of the more powerful computer while still maintaining quality of service guarantees.

2.1.2 PowerDial

Many applications can make trade-offs in the accuracy of results produced and the time it takes to compute the results. PowerDial is a system that can dynamically adjust application behavior upon changes of system load and power fluctuations [Hoffmann et al., 2011]. It creates a set of dynamic knobs that change the configuration of a running program. These knobs are application specific configuration variables that are identified by tracing program input parameters. For example, in the x264 H.264 encoder, a parameter can be specified that indicates the number of reference frames that should be used during motion estimation. The control variable in the program that is initialized by this parameter is identified and is considered a dynamic knob that the system can modify at runtime. These dynamic knobs are used to explore an accuracy versus performance trade-off space. Periodic heartbeats allow the system to identify when and how an application's dynamic knobs should be adjusted to maintain a given quality of service.

Similarly, pilots using the new programming infrastructure of this work can make adjustments to their programs and explore trade-off spaces with the provided energy aware simulation infrastructure. Such trade-offs may involve sacrificing the accuracy or spatiotemporal resolution of sensor data for the benefit of increased vehicle endurance.

Services can also, like the heartbeats of PowerDial, monitor the quality of sensor readings and dynamically change the sensor management scheme to satisfy the quality of the results expected by the mission programmer.

2.1.3 Loop Perforations

Loop perforation, [Sidiroglou-Douskos et al., 2011], is a technique that involves transforming a program's loops to only execute a subset of its iterations while still producing acceptable output. This is possible because many applications have some flexibility in the number of iterations that are required to produce output. A change in the number of iterations of a program can, however, also lead to a possible degradation in accuracy of the program's output. Because loop perforation can reduce the amount of computation, it also can lead to a reduction in energy requirements.

Sidiroglou-Douskos et al. [2011] have implemented a loop perforation system using the LLVM compiler framework. It identifies candidate loops in an application, perforates them, and executes the perforated program with training input. Perforated loops that do improve performance, cause application errors such as a crash, or produce output that fall out of a specified accuracy bound, are filtered out and not used. The remaining loops are exhaustively explored and their resulting speedup and accuracy are used to build a trade-off space. A developer can examine the perforations and gain insights into where they may find it acceptable to trade accuracy for performance or energy.

Conceptually, a similar mechanism can be used to manage sensors in the presented new programming framework. Depending on the flexibility of the application, a data set that contains sensors readings of an entire flight may not be required by an oceanographer. A glider flying in shallow waters may perform many inflections and create more snapshots of the water column in a short amount of time than is truly necessary. To

reduce the number of inflections, the flight angle of the vehicle could be adjusted. This, however, may cause improper or slow flight. Instead, the flight profile can remain the same, but the sensors are dynamically perforated to save energy. Thus, for some dives and climbs, analogous to loops, sensors are turned off. Depending on the quality of the results that are to be maintained, these *yo-perforations* may be changed dynamically, if for example, the environment in the area of operation is changing rapidly.

2.1.4 EnerJ

Another mechanism that can be used by programmers to choose energy and accuracy trade-offs in applications is through the use of approximate data types. The EnerJ programming language, [Sampson et al., 2011], is an extension to Java that enables approximate computing by the means of type qualifiers that distinguish between approximate and precise data types. Data that is annotated to be approximate data can be stored approximately or can be computed on with approximate instructions. This strategy requires the use of approximate aware hardware such as dynamic RAM that reduces its refresh rate on lines containing approximate data. Precise data, on the other hand, is computed on in the traditional sense. Explicit endorsements by the programmer from approximate to precise data certifies that approximate data in the program is handled intelligently and will not cause undesirable results.

The notion of approximate data types is evident in the domain specific language of the new programming framework presented. In EnerJ, data types are either approximate or precise. In the domain specific language, when a sensor is instructed to be enabled and to log all readings it may be considered precise. However, sensors in the language have additional qualifiers, rather than simply “approximate”, that more precisely describe how a flight engineer wishes to sacrifice sensor quality. Thus, the sensor specification of the language may be thought of in a similar manner as EnerJ’s

data type annotations.

2.2 AUV Programming Languages And Environments

2.2.1 MOOS-IvP

MOOS-IvP is a set of open source tools that provide autonomy for unmanned marine vehicles [Benjamin et al., 2010]. It is composed of two distinct components, the Mission Oriented Operating Suite (MOOS) and the IvP Helm. MOOS provides a publish-subscribe architecture and protocol where processes communicate through a single database in a star topology. The IvP Helm, short for interval programming, is one such process in MOOS and uses a behavior based architecture to implement autonomy.

Behaviors in MOOS-IvP are self contained expert systems that are dedicated to some aspect of an AUVs autonomy. One such behavior may be to guide the vehicle to a set of waypoints at a given speed. A mode specification determines which behaviors are active in each Helm control cycle. If multiple behaviors are active, the IvP solver is used to reconcile the behaviors using the objective functions generated by each of the active behaviors. These objective functions are piecewise linearly defined and are used by the behavior to influence the decisions made by the helm over some decision space. The decision space can be arbitrary but is typically composed of settings for speed, heading and depth.

Similar to MOOS, the new runtime system integrated into the glider and described later, also provides a publish-subscribe interface to services. These services use the runtime system to read and write to the glider's sensor memory which is analogous to the MOOS database (MOOSDB). Unlike MOOSDB, the sensor memory of the glider only supports numerical values and not strings. The string values for variables in the MOOSDB can be complex and could be encoded to contain several pieces of data. New

behaviors that make use of such variables must know how to decode these types of strings. Furthermore, the MOOS interface process that interacts with the vehicle must also translate MOOS variables into vehicle specific commands. For the Slocum Glider, this could be the command data structure generated by layered control as described in Chapter 3.

The MOOS-IvP system is one of the most widely adopted general programming infrastructures used by the AUV community. It has been used in several hundred hours of sea trial experiments on a variety of vehicles like the REMUS and Iver2. However, architecturally, MOOS-IvP requires that the vehicle control system and the autonomy system be on two distinct platforms. It would require a significant engineering effort to morph the existing infrastructure of the Slocum Glider into a conforming backseat driver paradigm. Furthermore, the system would not be backwards compatible, which is an import focal point of the new programming framework described.

To remain backwards compatible and to not discard the safety mechanism put into the current glider control system, existing components were used much as possible to create the new programming infrastructure instead of replacing it with a MOOS-IvP system. The domain specific language is also meant to be a high level specification and one not targeted for engineers or advanced programmers. The MOOS-IvP mission specification can become quite complex when considering the interactions of behaviors, mission modes, the generation of objective functions, the weights of the objective functions, and how the solver will resolve these functions to produce a vehicle command. The domain specific language and compiler could target MOOS-IvP in the future as long as the compiler generates a set of behaviors that will ensure the solver will produce the desired vehicle actions in accordance with the mission.

2.2.2 AUVW and AVCL

The Autonomous Unmanned Vehicle Workbench (AUVW) aims to bridge the gap between heterogeneous vehicles by providing a tool capable of mission planning, rehearsal and replay for arbitrary air, ground, surface and underwater vehicles [Davis and Brutzman, 2005; Davis, 2005]. A key component of AUVW is the Extensible Markup Language (XML) based Autonomous Vehicle Control Language (AVCL) that provides a common data model. The common data format along with a set of utilities that perform automatic data conversion to and from a vehicle specific format serves as the bridging element between the dissimilar vehicles. Besides AUVW, other mission planning tools have also chosen to use a XML based mission format [Dias et al., 2005, 2006; Godin et al., 2010].

Missions written in AVCL are translated to a vehicle specific format with the use of the Extensible Stylesheet Language for Transformations (XSLT). To parse vehicle missions into AVCL, a context-free grammar (CFG) definition must be created for each vehicle. Typically, a graphic interface in AUVW is used to create and edit AVCL missions as well as perform any required mission translations.

The benefit of having such a common data model is that one tool can more easily manage multiple vehicles. However, in AVCL, many tags exist that are only applicable for certain types of vehicles. For example, a *MoveRotate* command is used to rotate a vehicle in place using a body thruster. This maneuver is not currently possible on the Slocum Glider. Even general tasks can contain *meta* commands only used for certain vehicles, such as setting the navigation mode for the REMUS AUV [Davis and Brutzman, 2005]. Thus, missions that want to make use of a particular vehicle feature or capability may require both general and vehicle commands. The more meta commands that the framework is aware of, the more polluted and less general the common data model will become. Finally, although the programming framework described in this

work does not make use of AVCL, some of the AVCL behaviors are already supported or could be exposed by the domain specific language.

2.2.3 Common Control Language

The motivations behind the Common Control Language (CCL) is to investigate a standard for communicating between AUVs and surface vehicles as well as human operators [Duarte and Werger, 2000; Eberbach et al., 2003; Mupparapu et al., 2004; Duarte et al., 2004, 2005; Komerska and Chappell, 2007a,b]. To support groups of heterogeneous vehicles, a set of generic or basic behaviors are defined common all AUV. These generic behaviors are categorized into nine broad classes. The focal point thus far has been on the categories of maneuvering, navigating, communicating, configuration, execution and monitoring. The behaviors are used as basic building blocks for AUV interaction and for mission files.

Each vehicle that wishes to support CCL requires a CCL interpreter to be incorporated. The CCL interpreter includes an embedded planner and requires a mechanism to interface between CCL and the vehicle's existing control software. In the Distributed Control Environment (DICE) for the Solar-powered AUV (SAUV), this was accomplished with a bridge behavior that interacted with the legacy SAUV controller [Duarte et al., 2005]. The embedded planner in CCL adaptively searches for the best sequence of basic vehicle behaviors for the directives it is set to accomplish.

The Common Control Language is part of several graphical user interfaces (GUIs) being developed [Duarte and Werger, 2000; Mupparapu et al., 2004; Duarte et al., 2005] for vehicle missions control. The data types and structures are precisely defined in the specification [Komerska and Chappell, 2007a,b]. The serialization and deserialization of a subset of CCL messages is also given to enable users to build and parse messages. Several human readable representations are also provided throughout the literature, for

example, one that is inspired by the Lisp and C programming languages.

By limiting the expressiveness and types of behaviors that the vehicle can make use of, it is unclear what optimization opportunities are lost. For example, the protocol may need to be extended to enable an operator to specify that a sensor should only be logged based on a certain environmental feature. Thus far, CCL has generally been used to operate a fleet of AUVs to accomplish a set of tasks while the presented framework currently focuses on enabling programmers to use domain specific features to make trade-offs for single vehicle instead of a group vehicles.

2.2.4 Compact Control Language

The Compact Control Language (C2L) is developed by the Woods Hole Oceanographic Institute (WHOI) and is designed to allow AUVs to communicate with each other or a central node using low-bandwidth acoustic links [Stokey et al., 2005; Stokey, 2005]. The C2L protocol is designed around the capabilities of the WHOI Micro-Modem and WHOI Utility Acoustic Modem and thus messages are extremely compact and encoded into 32 byte packets. Because of its original design for use with the REMUS AUV [Hydroid, LLC.], the protocol also contains some vehicle specific messages. However, C2L is intended to be sufficiently generic to be used with other AUVs.

Currently, the protocol specifies 21 supported message types. These message including vehicle command messages and more generic sensor messages, such as, bathymetry and conductivity, temperature and depth (CTD) data. Sensor data in messages are compressed so that multiple data points may be sent in one packet. For example in a bathymetry message, the altitude and depth resolutions are changed to 10 cm for depths less than 100 m, 20 cm for depths between 100–200 m and 50 cm for depths up to 1000 m and 1 m resolution for depths greater than 1000 m [Stokey et al., 2005]. Furthermore, latitude and longitude values are encoded into three byte values to provide

a resolution of several meters. This compression allows three data points to be sent in one bathymetry message.

Clearly, this language is not meant for direct communication between a human operator and an AUV. Rather, as is the case with the REMUS, a graphical user interface displays sensor and vehicle updates as they are received and decoded and generates new messages in the binary format on the user's behalf. The language also contains vehicle specific functionality that may not applicable to all vehicles. In order for new features for vehicles to extend C2L, a developer must coordinate with C2L creators to ensure future compliance. The programming framework presented, on the other hand, contains a domain specific language for human operators and is extensible with the implementation of new services.

2.2.5 Dynamic Compact Control Language

The Dynamic Compact Control Language (DCCL), [Schneider and Schmidt, 2010], builds on the ideas developed in C2L. In C2L, message were precisely defined. For instance, a bathymetry message contains three data points with a specific resolution for each of the variable types. In DCCL, the message structure is defined by a structure language based on XML. A library validates the definition and efficiently encodes the message for transmission over an acoustic channel. Thus, DCCL provides a mechanism to adapt the messaging protocol and is not strictly defined like C2L. Much like its predecessor it is a binary protocol targeted for acoustic communication.

2.2.6 AUVish

AUVish, like C2L and DCCL, is a language target for acoustic communication among several vehicles [Rajala et al., 2006]. Specifically, AUVish is designed to enable the cooperation of a group of AUVs to maintain complete coverage in underwater mine

countermeasures (MCM). In the MCM presented in [Rajala et al., 2006], a formation searches for mines in a lawnmower search pattern with a leader AUV and several swimmer and follower AUVs. The AUVs in the formation communicate regularly using the AUVish protocol. If the leader has determined that a swimmer in the formation has been lost, for example by detonating a mine, the leader elects a nearby follower to take the swimmers place.

AUVish was developed for the purpose of simulating cooperative behaviors and did not consider real-world constraints. AUVish-BBM is a dialect of AUVish made to work within the functional capabilities of the WHOI modem [Beidler et al., 2007] and takes advantage of smaller 13 bit chirp packets as well as 32 byte packets. Both languages are, however, *ad hoc* and restricted to the target application.

Chapter 3

Slocum Glider Infrastructure

The Slocum Glider has its roots in Massachusetts Institute of Technology's (MIT) Sea Grant Odessy AUV. Specifically, the layered control system described by Brooks [1986], has made its way from MIT's Artificial Intelligence (AI) Lab into many AUVs including the Slocum Glider [Zheng, 1992; Bellingham and Leonard, 1994; Godin et al., 2010]. The layered control system determines, for example, how an AUV should propagate to a target location and how specific sensors should be utilized to meet a mission's objectives. This chapter will provide a brief overview of the Slocum Glider's implementation of the layered control systems and how it is used to control the AUV. The glider's existing computing infrastructure and its limitations are also described. Finally, the chapter concludes with a discussion of the usability of the existing programming system and motivates the need for a new programming framework.

3.1 Layered Control

The subsumption control architecture proposed by Brooks [1986] differs from what was the traditional control system of robots at the time and is shown in Figure 3.1.

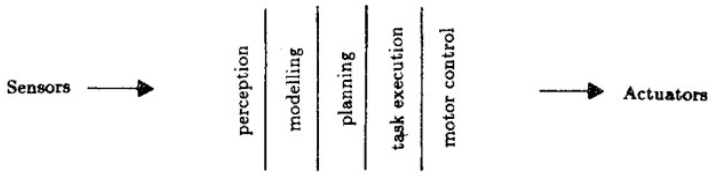


Figure 3.1: The traditional decomposition of a robot's control system [Brooks, 1986].

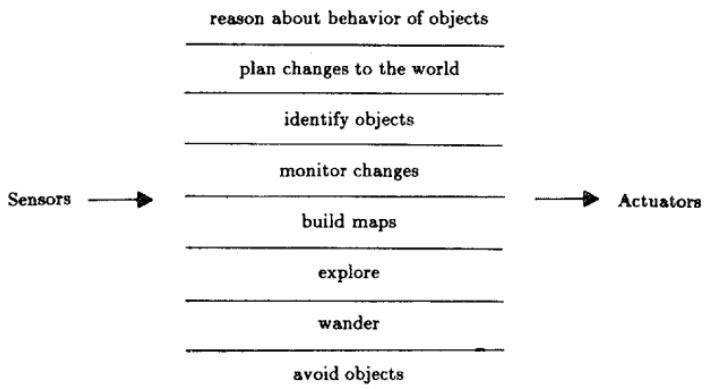


Figure 3.2: Brooks' design of task achieving behaviors [Brooks, 1986].

These control systems were decomposed into functional units while the layered control system introduced task-achieving behaviors, depicted in Figure 3.2, that could be built to perform complex tasks by increasing each layers level of competence.

The lowest level of this architecture should be rather simple. After it has been developed and debugged, it should not be changed. The next level of control would be built on top of the previous layer and is permitted to inject data into the lower level layer to suppress its normal output. The lowest layer in this system is unaware of the layer above it which may, if necessary, interfere with its output. Multiple of these layers may exist to achieve the overall task required by the robot.

Figure 3.2 illustrates this concept where a land based robot may be first programmed to avoid contact with objects at the lowest layer. Next, an additional layer of competence is added which allows the robot to wander. Another layer may provide the functionality to explore by trying to find places to go to. Other layers may then, build maps and routes, notice changes in the environment, reason and perform tasks on objects, format and execute plans, and finally reason about the behavior of objects and modify actions accordingly.

```
behavior: abend
  b_arg: overdepth(m)                    20
  b_arg: overtime(sec)                   600.0
  b_arg: samedepth_for(sec)              600.0
  b_arg: samedepth_for_sample_time(sec)  600.0

behavior: surface
  ...

behavior: set_heading
  ...

behavior: yo
  ...

behavior: prepare_to_dive
  ...
```

Figure 3.3: Skeleton of a sample mission file for the Slocum Glider.

The control systems of many AUVs have been built using the layered control concept. The Slocum Glider in particular uses this behavior driven architecture to determine what the task the vehicle should perform for each four second control cycle. These behaviors are specified as part of mission files by the user and are transmitted to the vehicle via satellite or radio.

3.1.1 Missions

Users program the Slocum Glider by creating mission files that will instantiate the control system. Figure 3.3 contains an overview of a sample mission. Behaviors provide the user the ability to control certain aspects of the vehicle. Each behavior has associated with it a set of arguments that allow the user to customize how the behavior operates. A *yo* behavior, for example, provides the functionality of diving and climbing a saw-toothed flight profile. The way in which the glider will dive or climb will depend on the arguments provided by the user through the behavior arguments. A pitch of 25–26° may be a desired behavior argument for the yo behavior [Graver et al., 2003].

Multiple behaviors of the same kind may also exist. A user should create mission programs in such a way that only a subset of behaviors are active at any given time in

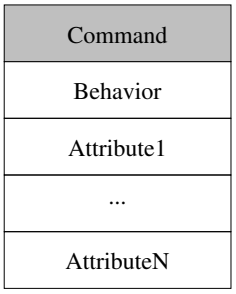


Figure 3.4: Command data structure.

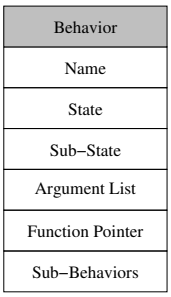


Figure 3.5: Behavior data structure.

order to receive the desired effect. If two yo behaviors exist in a mission, each specifying different diving and climbing angles as well as operating depth ranges, then only one yo behavior's goals should be sought after. The goals that are ultimately selected are determined by the subsumptions the behaviors apply to the final command in the layered control system.

3.1.2 Commands

The decision as to how the glider may react to the environment to achieve its goal is made every four seconds through the layered control system. The result of the system is to produce a final command that the glider should perform for the specific four second cycle. Figure 3.4 contains an abstraction of a command data structure. It consists of a behavior, as well as a set of attributes that the vehicle's drivers will use to manipulate its motors. For example, a pitch attribute may be set to modify the buoyancy pump's position. Although many instances of the command structure exist within the control system, only one final command will be produced to commandeer the vehicle.

3.1.3 Behaviors

Behaviors play a crucial role as part of the control system. They exists as part of command structure or as part of another behavior structure. As shown in Figure 3.5, each behavior instance will contain a name, state, sub-state, argument list, function

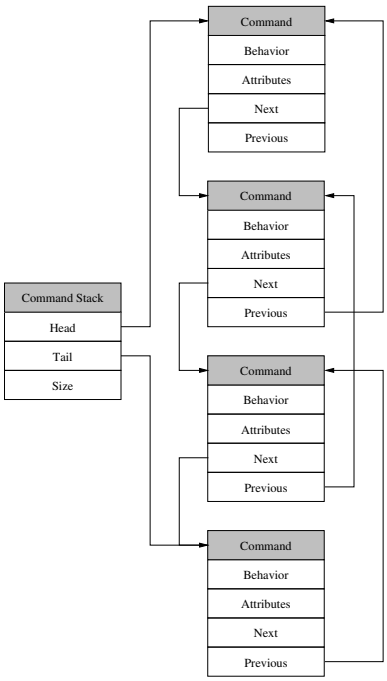


Figure 3.6: Command stack structure.

pointer, and a list of possible sub-behaviors. The state fields provide the necessary mechanisms for an instance of a behavior to continue proper execution across multiple cycles. Depending on the state of the behavior, the function pointer in the behavior is executed. The objective of the function, that the behavior calls, is specific to each kind of behavior. For example, a *sample* behavior function may turn on or off a sensor or change a sensor's sample rate. The specific sensor that the sample function should manipulate is specified by the user through the behavior's argument list. A function may also create other sub-behaviors and call their functions. Although a function can have multiple tasks to perform, the key role of most functions is to modify the attributes of the command that the behavior belongs to. A *climb_to* behavior would, for example, manipulate a command's attributes in such a way that would cause the buoyancy pump's device driver to expel water, thus changing the glider's buoyancy.

3.1.4 Command Stack

The existing programming architecture parses the user provided mission file, and creates an instantiation of a command stack structure as illustrated in Figure 3.6. The ordering of the behaviors listed in the mission will directly correspond to the ordering of the behaviors in the command stack. The behavior at the beginning of the file will be placed at the top, or head, of the command stack; the last behavior will be positioned at the bottom, or tail, of the stack.

At each cycle, the layered control system will use the command stack to produce the final command the glider will execute. To create this command, the stack's tail command attributes are first set to default values. If the tail command's behavior is active, then the behavior's function is executed and the attributes of that command may change. Next, the attributes of the tail command overwrite the attributes of the command in the next level in the stack. Again, if active, the attributes may be changed by this level's behavior. This procedure will be repeated until the head command completes its modifications on its attributes. The command at the head of the stack will then become the output command which will be used by the glider's drivers.

This control system allows the vehicle to negotiate the actions that would meet a mission's overall goals. Behaviors at higher layers may subsume or overwrite the attributes set by layers below. Behaviors of utmost importance, such as the *abend* (abnormal end) behavior should be placed at the beginning of the mission file to keep the vehicle safe. This will enable the more important behaviors to be the last to contribute to the resulting final command.

3.2 Glider Hardware

Teledyne Webb Research's development of the Slocum Glider began over a decade ago and some of the hardware components have begun to show their age. The computing

infrastructure, although reliable, are some of the oldest components on the vehicle today. The processing power on the Slocum Glider consists of two 16 MHz Motorolla 68338 Persistor processors, with 1 MB of flash and 512 KB RAM [Persistor Instruments Inc.]. The flight controller is located in the rear of the vehicle and is responsible for piloting the AUV. It executes the mission files and layered control system as described in the previous sections. The science computer, typically located in a center payload bay, acquires the scientific data from connected sensors using *proglet* drivers.

The two computing platforms in a stock Slocum Glider communicate via a RS-232 serial connection known as the *clothesline*. A driver on the flight controller and a proglet on the science computer speak the *superscience* protocol to communicate sensor data. Glider control software before the 7.0 release performed data logging only on the flight controller. All sensor data that needed to be logged had to traverse this clothesline. Since the 7.0 release, science data logging is possible and has significantly reduced the complexity and trade-offs users had to make to successfully log sensor data.

The existing control software is the largest to ever run on that particular processor type. However, it is struggling to meet the requirements needed by the glider's customers. For example, before science data logging, if too many sensors were requested to be logged by the flight controller, the clothesline could become a bottleneck and cause control cycle abnormalities. Science data logging has relieved some of the pressure, however, the number of sensors that can be sent to the flight controller are still limited. Other tasks and control sequences can also cause the vehicle to overrun the control cycle. Such limitations hinder the vehicle from performing dynamic tasks and optimizations that rely on sensor data, processing, and modeling. With projects such as acoustic data processing and acoustic communication among groups of gliders in the works, the requirement of a more modern and powerful computing system has arisen.

```
behavior: sensors_in
    b_arg: c_att_time(sec)                  -1
    b_arg: c_pressure_time(sec)             -1
    b_arg: c_alt_time(sec)                  -1
    b_arg: u_battery_time(sec)              -1
    b_arg: u_vacuum_time(sec)               -1
    b_arg: c_profile_on(sec)                -1
    ...

behavior: sample
    b_arg: args_from_file(enum)             -1
    b_arg: sensor_type(enum)                 0
    b_arg: state_to_sample(enum)             1
    b_arg: sample_time_after_state_change(s) 15
    b_arg: intersample_time(s)               2
```

Figure 3.7: The sensors_in and sample behaviors.

3.3 Programming Issues

The user's interface to the glider is in the form of mission files. Missions are thus an extremely important aspect of the whole system. Without an intuitive mechanism to specify a scientist's goal, the effectiveness of any tool is dramatically reduced. It should then be made easy to specify how a glider should use its sensors as well as how to propagate from one target location to the next. However, the existing glider programming infrastructure can make tasks, such as sensing, cumbersome and difficult to read and write.

The mechanism to specify which sensors should be active in a mission, and how samples should be taken are accomplished using the *sensors_in* and *sample* behaviors as shown in Figure 3.7. When using the sensors_in behavior, a user denotes each sensor they wish to customize by providing to the behavior a sensor specific argument as well as an integer with the desired sample rate. A value of negative one is used to disable a sensor, a zero to sample as quickly as possible, and an integer n to specify a sampling every n seconds.

A more customizable, and sometimes preferred behavior, *sample* allows a user to not only specify at what rate to sample (intersample time argument), but also when to

sample a sensor. Similar to Unix file permissions, an integer based on a bit-vector will allow for readings when the glider is at the surface (bit-vector value of 8), climbing (4), hovering (2) or diving (1). Thus, a state to sample value of 14 would sample a given sensor when at the surface, climbing and hovering, but not while diving.

A sample behavior must exist in the layered control system, and thus the mission file for each sensor. Unlike the sensors_in behavior where select sensors have their own arguments, the sample behavior requires the user to indicate which sensor the behavior should control by providing an integer value to the sensor type argument. The vehicle's documentation denotes which integer values correspond to which sensors. Interestingly enough, these values also happen to be directly used as indexes into an array. Not only is specifying an index to an array not very intuitive for end users, but it has also led to incompatibility and usability issues. Software updates have changed the array but the documentation has remained unchanged. For example, to indicate the sampling properties of the AUVB fluorometer sensor, the value of 27 should be used. However, because of changes to the array in the glider software, the correct value is 28. This inconsistency can be observed across multiple sensors. Missions which had previously been written and deployed may not function properly across software updates. Users were never informed of the possibility that changes may need to be made to their missions, or that multiple versions of the same mission may need to exist for each revision of the software stack.

The writing and modification of missions can be a non-trivial task. To properly program the AUV, one must appreciate the layered control system and how behaviors change their command's attributes and how the combination of attribute changes at each layer produces the desired control of the system. Most behaviors are activated and deactivated by providing the behavior with a *start_when* and *stop_when* arguments. An integer value is again used to symbolically represent different states or events that

should trigger the behavior. Many of such triggers are based on a command's attributes. One such start_when trigger value is 4, or when a command's up and down attributes are idle. Therefore, when adding a behavior which uses this value as a trigger, one must consider at what state and sub-states other behaviors are in and how each have contributed to the current behavior's command attributes. This requires an intimate knowledge by the user of how each behavior has been implemented and how it will function at any time in the mission. Once a user has determined how to properly activate a given behavior, the user must next consider how behaviors in higher layers will modify or subsume the attributes set by the current behavior. So, when adding a behavior into a mission it is necessary to account for both behavior activation and command attribute subsumption.

Missions files are the user's tool to program the system and have proven to be difficult to program. Multiple mechanisms, with different interfaces exist to change the sample rates and activation of sensors. Adding behaviors to create new or modify existing missions involves knowledge beyond the interests of its users. The lack of documentation describing behaviors and their arguments also often leads to inspections of the vehicle's source code. As a result, it has become common that only the missions provided by the manufacturer are used. Moreover, only a subset of the a mission's arguments and waypoints are typically changed. This severely limits the scope of the glider's use.

Finally, the present system is extremely static in that it cannot react dynamically to the environment. Once a mission begins execution, it can only be reprogrammed during its periodic surfacings, otherwise it will continue as instructed by the mission. Usual missions denote that the AUV should surface every 3–6 h to send a subset of sensor readings through the glider's communication infrastructure. At this time, a scientist may interpret the data and instruct the glider to resume, or execute a new

mission. No instrumentation currently exists which provides the functionality to react to the environment the vehicle is experiencing. If an interesting phenomena occurs while underwater, the glider will ignore it and continue as commanded. Only at the next resurfacing can a scientist react and send the vehicle back to collect more data points. Some phenomena are short-lived and so the opportunity to observe it may have already been lost. It has become evident that to bring about the glider's full potential, it is necessary to create a medium where scientists can express their requirements to the glider in a way that is specific to their domain.

Chapter 4

New Programming Framework

Autonomous underwater vehicles are powerful tools but can be difficult to program especially by non-experts. The goals of this dissertation is to create a new programming framework that enables AUV missions to be specified in a domain specific manner and use domain specific features to optimize a vehicle's deployment. The infrastructure should allow a user, such as an oceanographer, to easily express and make use of such features and be provided feedback to determine if the trade-offs are a worthwhile sacrifice. Furthermore, the safety of the vehicle is of utmost importance, and framework must be designed as such. The Slocum Glider is the target platform for the framework and is a widely deployed AUV with a large user community. However, much of the work, such as the language abstractions, compiler, and service infrastructure, is applicable to other vehicles both in the domain and outside the underwater domain.

This chapter describes several components of the programming framework. Although of equal importance, the simulation, power measurement, and energy model infrastructure are described separately. The highest level of abstraction if provided by the *ALGAE* domain specific language. The language's target audience are scientists that do not necessarily know or require detailed knowledge of the specific AUV. Services, like feature detection, are created in the framework and exposed via ALGAE. These services can be created in several languages and in the *GLOC* scripting engine created for the Slocum. Services can also vary in complexity and be implemented and executed on the *AVBot* single board computer integrated into the AUV to extend its

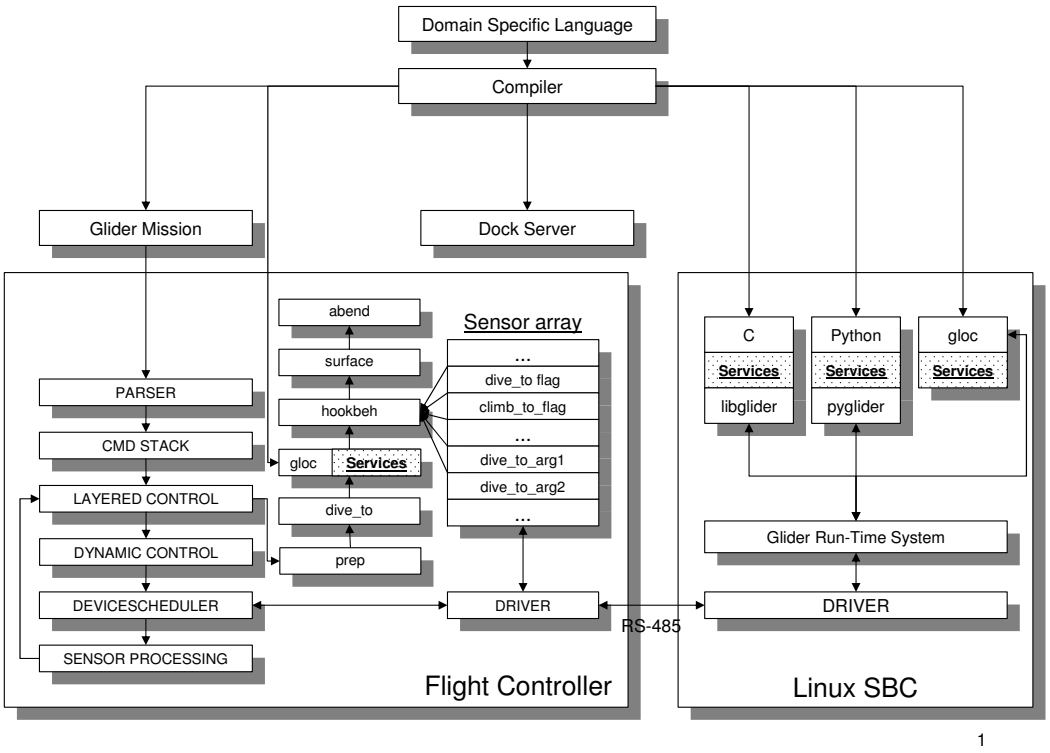


Figure 4.1: Overview of new programming infrastructure.

functionality. This is also the second level of abstraction provided by the programming framework and is targeted at users with more programming experience and domain knowledge. This separation of complexity to extend the vehicle is intentional and was driven by the need to ensure that the vehicle remains safe, robust, and practical.

4.1 Overview

An overview of the new programming framework is shown in Figure 4.1 and will be described in greater detail in the following sections. To create more dynamic functionality in the vehicle, we have added two behaviors into the layered control stack of the glider's control system. The first of these behaviors is the hook behavior that allows for the dynamic creation of existing glider behaviors. The second, the GLOC behavior, contains within it a small virtual machine that can be used to execute small programs. In combination these two behaviors allow for dynamic, although simple, control of stock

	TS-5500	TS-7800	TS-7260
CPU:	x86 at 133MHz	ARM9 at 300MHz or 500MHz	ARM9 200MHz
Memory:	64MB	128MB	64MB
Storage:	1 CF	1 FullSD, 1 MicroSD, 2 SATA	1 FullSD
USB:	2	2	2
Serial Ports:	3	Up to 10	3
Analog to Digital :	8x12-bit channels	5x10-bit channels	2x12bit channels
Active Power:	2.7W	3.42W (300MHz), 4.14W (500MHz)	2W
OS:	Linux 2.4	Linux 2.6	Linux 2.4

Table 4.1: SBCs that have been installed in the Slocum Glider.

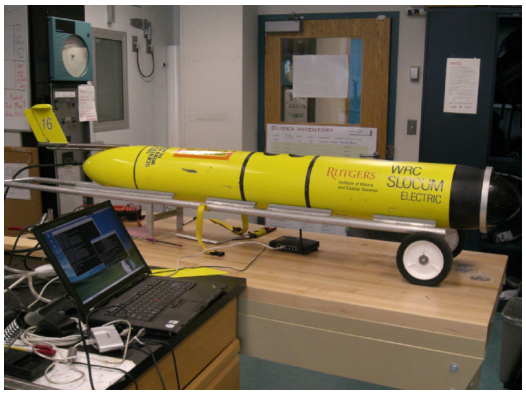
gliders.

To enable more advanced sensors and control, a Linux single board computer (SBC) has been integrated into the glider. The SBC communications with the main flight controller through a new glider device driver. This device driver allows the SBC to read and write into a portion of glider's main memory called the *Sensor Array*. Programs on the SBC, called services, can subscribe and publish values to the sensor array. Services running on the SBC are generally more complex than services running in the GLOC behavior in the glider.

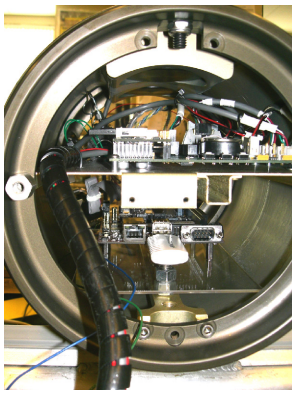
Finally, programs written in the domain specific language are transformed from high level constructs to a set of glider mission files, behaviors, *Dockserver* mission control scripts, and services. These components can together execute the high level programs specified by a user on the Slocum Glider target platform.

4.2 AVBot Hardware

To support the new software infrastructure as well as a set of more complex sensors and services, we have tested and integrated several single board computers with our target platform. A SBC integrated as part of the infrastructure for the glider is called an



(a) Vacuum sealed glider with science bay located under the access point.



(b) TS-7800 installed in the bottom half of the science bay of the glider.

Figure 4.2: Current hardware prototype.

Autonomous Vehicle Robot (AVBot). Table 4.1 contains the hardware specifications of the SBCs from Technologic Systems. The power measurements were taken under load on a Tektronix TDS 3014 oscilloscope. These SBCs were specifically chosen because they provide a wide variety of inputs for sensors and run a commodity operating system that provides for rich driver support as well as an easy development environment. Depending on the sensor, power, logging, as well as processing requirements, it may be advantageous to choose one board over another. For example, floating point operations are likely faster on the x86 than on the ARM9, while the ARM9 board has more RAM as well as serial interfaces. Other SBCs will likely be considered in the future which may be more appropriate for use with certain sets of sensors.

Of the SBCs, TS-7260 has been chosen as the default SBC for the prototype system as it strikes a nice balance of being relatively low power while still providing the necessary computational capabilities required by many applications in the domain. Power is provided to AVBot by the glider's battery or by an external power source. The TS-7260 has been fitted with a battery backup system to ensure that the AVBot software and operating system be can safely shutdown. While deployed, the battery backup will be recharged from the glider's batteries.

A Slocum Glider with the AVBot hardware installed can be seen in Figure 4.2. The science bay is located in the middle of the glider. It is the section of the glider which contains the science computer and all of its attached sensors. In Figure 4.2(a), it is located above the access point which is used to communicate with AVBot during benchtop testing. The TS-7800 is shown in Figure 4.2(b) completely installed in the science bay and connected to the remainder of the system.

Communication with the glider is accomplished through either an RS-485 connection to the glider's flight computer or through an RS-232 serial connection to the glider's science computer. When connected to the flight controller, flight data from the glider is more quickly accessible to the AVBot, while when connected to the science computer, science data is more quickly available.

In the initial design the SBC communicated with the vehicle through the science Persistor. A proglet, a program on the science computer which acquires data from sensors, was written to allow AVBot to act as an ordinary sensor to the existing system. The proglet acted as a proxy to allow AVBot to write into the glider's sensor memory by speaking the superscience protocol with the glider over a slow serial connection known as the clothesline. However, after several missions at sea, the communication latency experienced was too high when the AVBot was tasked to maneuver the vehicle's flight through this mechanism. Therefore, the RS-485 link connected directly to the glider's flight controller is preferred.

In the current incarnation of the AVBot prototype, a device driver was written for the vehicle that speaks the *gliderbus* protocol over the RS-485 serial connection. The AVBot device driver is shown in Figure 4.1 as part of the glider's flight controller software. The gliderbus protocol is used by other devices like the AUV's fin controller and coulomb counter. In this case, the protocol is used to perform a set of memory read and write transactions to the sensor variables. Not only does this approach provide

direct access to the vehicle, but it is also much more reliable than the science proglet approach.

With significantly more computing power in place, the glider is able to support a wider variety of sensors and applications. Sensor data can be sampled at higher rates and stored locally on the SBC. Processing, decision making, and acting on sensor data may now be possible but comes at cost of additional energy. The increase in energy required will likely lead to lower deployment times, which may be justified if a glider is able to accomplish enough data acquisition in one pass of an area, rather than requiring multiple passes to aggregate a similar set of data points. Other optimizations such as changing the glider's flight path, sensor triggering, or hierarchical power management may also be considered to subsidize the energy cost.

4.3 Hook behavior

With the creation of a reliable mechanism in the AUV's software to allow the reading and injection of data into the sensor array, the glider must also be instrumented to make use of such data to perform actions. For example, the glider could be tasked to detect and then track a physical feature in the ocean. To remain as non-intrusive as possible, the creation of a new behavior was chosen as the mechanism to provide the programming framework new dynamic capabilities. By using a behavior the overall changes that need to be made to the control system are isolated. This new behavior, named the *hook* behavior, is shown in Figure 4.1 as part of the layered control stack.

The approach of using a behavior has the benefit of being backward compatible. If a user wishes to use the new system, the behavior is specified in the mission file. If the behavior is not included in the mission, the behavior will never be part of the layered control system and the glider will function as an unmodified vehicle. Furthermore, higher priority behaviors can still overwrite any changes made to the command stack

by the hook behavior. For example, a higher priority abend behavior may overwrite invalid or unsafe commands written by the hook behavior.

At each four second control system cycle, when the hook behavior runs, it interprets a set of sensor array values set by the AVBot system to see if any actions need to take place. In Figure 4.1, for instance, if the *dive_to_flag* is set, the behavior dynamically creates a dive_to sub-behavior in the layered control system. The arguments for the sub-behavior are set by reading the corresponding argument list values specified by preselected set of sensor values in the sensor array. The sub-behaviors are executed and, optionally, destroyed. Depending on the flags set by the AVBot, multiple sub-behaviors may be created and executed in each cycle. Because the behavior uses existing behaviors, the extensive safety features in the current system do not go at a loss. It also becomes possible to react and change the glider's flight to react to phenomena in the environment.

4.3.1 Thermocline Tracking Experiment

Not being able to react to its surroundings can lead the glider to inefficiently study some ocean phenomena. To showcase the dynamic capabilities that have been added to the Slocum, the infrastructure has been used to track a thermocline. A thermocline is a layer of water where temperatures change drastically, typically within several meters.

To detect and track such a thermocline, a simple algorithm has been developed that observes when a threshold has been met in recent depth and temperature data points. Using fictitious data loosely based on real thermoclines observed by previous glider missions off the coast of New Jersey, and the stated algorithm, a benched glider in simulation mode was able to successfully track a thermocline. Figure 4.3 shows a glider's depth profile of such an experiment, where the thermocline is represented by the gradient. The warm surface water is assumed to be 24° C and the cool deep water

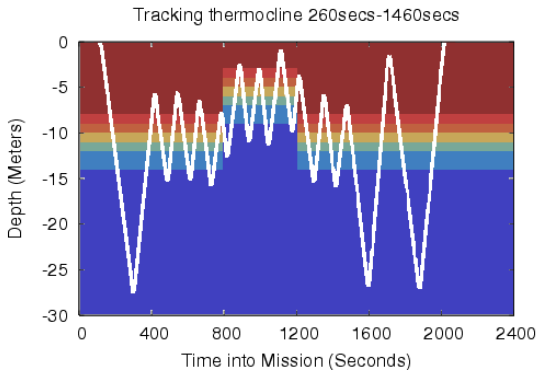


Figure 4.3: On bench thermocline tracking simulation.

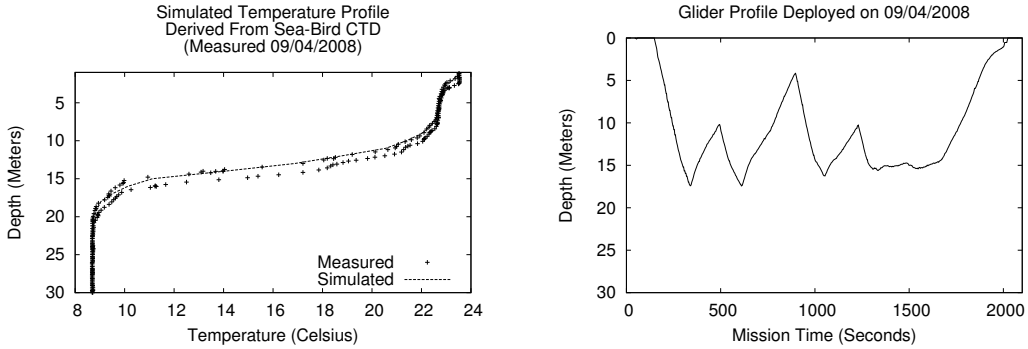
7° C. The simulated thermocline has one internal wave.

With the intent of remaining backward compatible, the glider was instructed to perform three dive and climb sequences with the traditional layered control system. The infrastructure was used to pilot the tracking of the thermocline from 260 s to 1460 s into the mission; after this it would return to finish its layered control mission. The profile depicted in Figure 4.3 confirms that, although simple, the tracking algorithm performs well.

As part of the experimental evaluation of the system, two deployments were performed in the Atlantic Ocean approximately 30 km off the coast of southern New Jersey. During these deployments, the glider was tethered to a buoy as a safety precaution. Although the tether may slightly impact the flight behavior of the AUV, it provides the advantage of a speedy recovery in the case of unplanned events.

The overall objectives of the first deployment were to assess if the architecture was sound and to determine if the simulations are reflective of true environmental conditions. During the first run of the day, the glider was programmed in the framework to change its target depth from 15 m to 25 m based on a specific sensor reading. It was successful in doing so.

The second run was the first attempt of tracking a thermocline. The AUV flew to



(a) A thermocline measured by a Sea-Bird CTD profiling sensor.

(b) Profile of a glider programmed using the infrastructure to track a thermocline.

Figure 4.4: Mission profile of thermocline (a) being tracked in the Atlantic Ocean by a Slocum Glider (b).

its desired depth but failed to detect the thermocline because the algorithm was not properly provisioned to deal with sensor readings that were not monotonically increasing (while climbing) or monotonically decreasing (during diving). Thus, temperature fluctuations caused the data window to be reset and caused the threshold to never be reached. Provided that the simulations did not reveal the error in the algorithm, it was still considered a prosperous deployment.

The focal point of the second deployment was to use the knowledge obtained from the first mission to track a thermocline. Minor changes were made to the tracking algorithm based on the data collected from the previous deployment. Figure 4.4(a) exhibits the day's water column temperatures as observed using a Sea-Bird CTD profiling sensor. The thermocline is present at approximately 10–18 m where dramatic temperature change can be observed. A mission nearly identical to the one in the first deployment was carried out. The vehicle's vertical profile in Figure 4.4(b) indicates that the algorithm successfully detected the thermocline and changed the glider's target depth range. A dramatic temperature change was not observed in the climb starting at approximately 700 s into the mission. The loss of the thermocline was not due to a defect in the algorithm, but due to the CTD sensor itself. It is possible that an old water

column was not flushed out from the CTD sensor. The dive immediately following the climb did however detect the thermocline again.

It is interesting to note that the vehicle leveled out at 15 m towards the end of the mission. Although the infrastructure was still in control for a small portion of this time, it piloted the vehicle to climb. For most of the hovering period however the glider's layered control system was in control and instructed a climb. This type of behavior has been observed in other deployments for brief periods of time. Another possibility is that the tether somehow restricted the glider's movement. Regardless, it has been shown that the framework has enabled the Slocum to react dynamically to changes in its environment which was not feasible in the existing software system. Coming up with a reliable implementation of an efficient thermocline tracking algorithm was however not the focus of these experiments, only to showcase its feasibility. Other thermocline tracking algorithms [Petillo et al., 2010; Cruz and Matos, 2010a; Zhang et al., 2012] have since been developed and are described in Section 7.7.2.

4.4 Embedded Scripting Engine

The objective thus far has been to provide the necessary groundwork to make the Slocum Glider a more effective tool for researchers, and to enable complex algorithms that cannot be performed on the current computing infrastructure. However, the addition of the AVBot SBC comes at the cost of an increase in power requirements. For example, if the SBC is used to power manage sensors, the employed management algorithm must ensure that the net energy cost justifies its use. For data processing, it may be advantageous for the SBC to be powered only when enough data becomes available to perform the calculations. Finally, if the SBC is used to control the glider's flight, there may still be autopiloting opportunities where it can be powered off to let

the legacy system control the vehicle. Thus, a system is needed that allows for the flexibility of the existing prototype system but at a lower energy cost so that the vehicle can continue to sustain long term deployments.

To fulfill this requirement a lightweight scripting engine, named GLOC, has been developed for the Slocum Glider. It is designed to operate on the glider's native processor, as well as on AVBot, so that researchers with standard vehicles can also develop new algorithms for the glider. These algorithms can be used as services in the programming framework and are capable of using a hierarchical management technique for energy efficiency [Sorber et al., 2005; Banerjee et al., 2007]. The GLOC engine running on the flight controller is implemented as a behavior for the vehicle's layered control system. This is to ensure that other higher priority behaviors can override the actions requested by a script.

Because the language of the scripting engine is rather low level, a BASIC-like high level programming language and compiler called *GBASIC* have been developed. This sample language illustrates the flexibility of the engine and is a reference point for other future languages that may be developed to program the vehicle. Using this infrastructure, dynamic feature tracking of a thermocline has been performed, which is not possible with a stock glider.

4.4.1 GLOC

The current programming environment for the Slocum Glider is limited since users are confined to the behaviors produced by TWR. Creating new mission files can also be cumbersome, because of the complex interactions between behaviors. The programming framework developed hopes to improve the programmability of the glider as well as add additional functionality that will make the vehicle an overall more effective tool. The aim of the scripting engine is to increase the scope of applications that can be performed

```
 1  nregs 8              16  label 1
 2  nlbls 7              17  loads 444,r4
 3  ninstr 27            18  loadd 5.0,r5
 4  label 0              19  cmplt r4,r5,r6
 5  loads 444,r0         20  cbr r6,5,6
 6  loadd 15.0,r1        21  label 5
 7  cmpgt r0,r1,r2       22  loadi 0,r7
 8  cbr r2,3,4           23  stores r7,1387
 9  label 3              24  jumpi 2
10  loadi 128,r3         25  label 6
11  stores r3,1387       26  yield
12  jumpi 1              27  jumpi 1
13  label 4              28  label 2
14  yield                29  yield
15  jumpi 0              30  jumpi 2
```

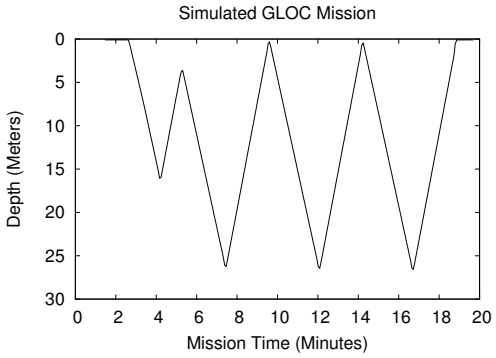


Figure 4.5: A sample three yo mission executing a GLOC script. The glider mission (not shown) is instructed to dive and climb three times between 2–25 m. The script behavior instead attempts to first fly a single yo between 5–15 m before relinquishing control and letting the other yo behavior complete its mission. Both behaviors are active at the same time, with the GLOC scripting engine at a higher priority in the layered control stack. The resulting simulated flight path is shown to the right.

on the vehicle, as part of the new framework or even a stock vehicle.

When used as part of the framework, GLOC is used to implement services, both simple and complex. However, more generally, it can be a standalone mechanism to easily develop new algorithms for the vehicle. This is particularly useful during the simulation and testing phases of newly designed algorithms where the engine can act as a test bed before the algorithm is independently created, for example, as its own behavior.

The scripting engine, like the hook behavior, is implemented as a behavior and resides in the layered control system. It is therefore able to take advantage of some of the safety features present in the layered control system. The engine, GLOC[1] , resembles a reduced instruction set computing (RISC) architecture [Hennessy and Patterson, 2007] assembly language and is inspired by ILOC[2] [Cooper and Torczon, 2008] developed at

[1] GLOC: Glider intermediate Language for Optimizing Compilers

[2] ILOC: Intermediate language for optimizing compilers

Rice University. Currently, GLOC has over thirty instructions, including instructions to load and store data to and from the glider's sensor list and the engine's registers, perform mathematical and logical operations, produce output, as well as perform jumps and conditional branches to labels.

Although an assembly level language is typically considered to be more difficult to program than a higher level language, it has several advantages. Parsers for complex languages often require a more complex set of tools and libraries to implement them. Some parsing techniques are also memory intensive and would use too much of this scarce resource on the vehicle's 1 MB Persistor processor. In addition, the size of the codebase may be a concern since larger codebases are typically harder to maintain and take longer to debug.

Due to the aforementioned reasons, the design decision was to make the scripting language very simple. The code base of the core of the engine itself is compact, measuring under 600 lines of code. Although this does not necessarily ensure reliability, the engine has thus far been easy to maintain. Another advantage is that it is simple to parse and interpret, and has a small memory and processing footprint. The exact memory and processing requirements are dependent on the script being executed, but all of the experiments so far have only required a few kilobytes of memory and have added at most 30 ms to the glider's four second control cycle.

A sample of a GLOC script is listed in Figure 4.5 along with the simulated flight profile of the vehicle's mission. The example illustrates the interactions between a mission's behaviors and the behaviors induced by a GLOC script. The glider mission specifies a sequence of three yos, where a yo consists of a dive and climb operation. Instead, the GLOC script first instructs the glider to perform a single yo between 5–15 m. In the layered control architecture, the script behavior is at a higher level than the yo behavior and thus supersedes the yo. As shown by the flight profile, the script

successfully accomplishes its task and then lets the glider proceed with the rest of the mission.

When a glider mission is executed with the GLOC behavior, the engine first loads and allocates the memory required by the specified script. The beginning of the script file, lines 1–3 of Figure 4.5, specifies the number of registers, labels and instructions that the script will use. This allows for the behavior to statically allocate all required private memory at one time, and makes deallocation of the memory easy at the end of a mission. This mechanism follows the general design pattern used for behaviors throughout the glider software.

Labels in GLOC are numbered and serve as targets for jump and conditional branch instructions. These instructions allow for control flow to occur in the scripting engine and are the building blocks for conditional statements such as an *if* and loops such as a *while*. The jump targets may be specified directly by a number or indirectly through a value contained inside a register. Conditional branch instructions jump to the first label when the condition holds true and jump to the second label if false.

A register contains data values such as floating point numbers or integers. Mathematical and logical instructions require their input values to be in registers and write their output to target registers. Registers can be populated with values using a number of load instructions. The *loadd* and *loadi* instructions of Figure 4.5 assign a floating point and integer values to their given target registers, respectfully. A *loads* instruction however loads a register with data from the glider's sensor array. The sensor array, is part of the pre-existing glider behavior programming architecture. The load instruction in line five assigns the value of the vehicle's current depth to register zero. This is because the 444th sensor variable in the glider's sensor array is designated for the depth information. Writing data to the sensor array is possible via the *stores* instruction.

The scripting engine is able to gain flight control of the vehicle by using the Hook

behavior. The engine, through the Hook behavior, can dynamically create and execute sub-behaviors by setting the appropriate flags and parameters in the glider's sensor array. Lines 10 and 11 correspond to such an interaction between the components. Sensor 1387 is a variable that is checked periodically by the hook behavior to see which sub-behaviors are to be created. A value of 128 activates a climb_to behavior whose parameters in this particular case have been predefined in the glider's mission file. Other behaviors can be simultaneously activated by setting appropriate flags through the Hook behavior's sensor interface.

The GLOC engine is lightweight and can quickly execute scripts as part of the layered control system. However, the exact overhead is reliant on the code being executed. It is currently the responsibility of the programmer to ensure that only a limited amount of code is executed as behaviors are not preempted by the glider software. In GLOC, the *yield* instruction informs the engine that the program wishes to relinquish execution for the current control cycle. It is in this manner that cooperative multitasking is achieved. The user must be aware that taking a large quantum of execution could lead to undesired control cycle overruns.

The scripting environment is flexible and robust, and will increase the scope of applications that can be performed on the glider. The engine is also not restricted to performing tasks independently, but can collaborate on computation and data processing tasks with AVBot. GLOC can also reduce the energy consumption of the vehicle by alleviating the need to have the Linux SBC be powered at all times. Powering off AVBot may be desirable in many scenarios, for example, to be part of a hierarchical power management architecture [Sorber et al., 2005]. If an application on AVBot does not require a large processing workload for a portion of its execution, the task could instead be executed remotely by the scripting engine on the glider. In this scenario, the SBC can enter a low power mode or power off entirely until it is needed again.

```
1  label: state1
2  if m_depth > 15.0 then
3    SCI_RUHP_BEHS = 128
4    goto state2
5  endif
6  yield
7  goto state1
8  label: state2
9  if m_depth < 5.0 then
10   SCI_RUHP_BEHS = 0
11   goto state3
12 endif
13 yield
14 goto state2
15 label: state3
16 yield
17 goto state3
```

Figure 4.6: A sample GBASIC program to perform a single yo between 5–15 m. This program is the source of the compiler generated GLOC script shown in Figure 4.5

A transfer into low power mode may also be profitable when data processing is not worthwhile until a large data set has been acquired by onboard sensors.

In combination with AVBot, tasks can be performed concurrently with the scripting engine. The programming infrastructure can take full advantage of the scripting engine in the vehicle and could perform automatic code generation of lightweight tasks. Furthermore, scripts can also be dynamically generated, transferred, and executed.

4.4.2 GBASIC Language

To showcase the capabilities of the scripting engine and to improve the programmability of the scripting system a subset of a BASIC-like programming language, called GBASIC has been created. Although not a complex and feature rich language, the implementation of GBASIC can serve as a reference point for other language designs and compilers that target GLOC.

The compiler for GBASIC was implemented using Python and the Python Lex-Yacc (PLY) toolset. This toolset is comparable to standard Lex and Yacc tools used in

compiler construction [Aho et al., 2007]. The code base for the compiler is small with approximately 700 lines of source code. As the language develops and more BASIC inspired statements are added, the code size will slightly increase.

Currently, the compiler does not perform any optimizations on the GLOC code to reduce either its code size or memory consumption. However, type checking and casting is implemented since it is the responsibility of the compiler, or the programmer, to ensure that the GLOC code running on the actual vehicle is safe.

The GBASIC language has support for variables and one dimensional arrays of integer or floating point values. Like GLOC, GBASIC can express mathematical, relational and logical operations. The language contains the *label*, *goto* and *if* statements, useful to control the flow of execution. Although not yet implemented, the creation of *while/wend*, *do/loop*, and *for/next* loops should be trivial as they can be constructed from the already built constructs.

An example of a GBASIC program is listed in Figure 4.6. The GLOC code presented in Figure 4.5 is in fact the output code generated by the GBASIC compiler of the program in Figure 4.6. The higher level language is more readable and thus makes it easier to debug. Labels, for example, are not just numbers as in GLOC, but can have descriptive names. Built-in vehicle variables such as *m_depth* can also be called directly by their name, as specified in the glider's *Masterdata* documentation. Although GBASIC may not be the most appropriate language to develop programs for the Slocum Glider, it illustrates that higher level languages can be constructed for the scripting engine.

4.4.3 Thermocline Tracking Experiment

The GLOC engine adds functionality to the vehicle that is not available on a standard glider. To showcase its usefulness, the thermocline tracking algorithm of [Petillo et al.,

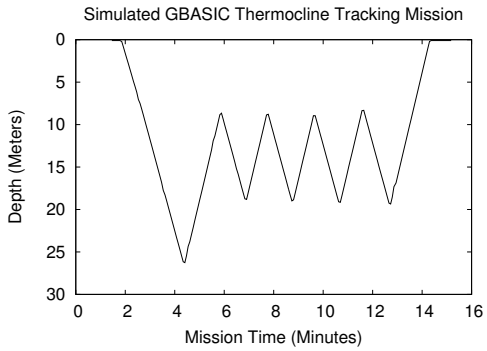


Figure 4.7: Flight profile of a simulated glider tracking the thermocline in Figure 4.4(a).

2010] has been implemented in GBASIC. The compiled GLOC code was executed by the scripting engine in a simulated thermocline tracking mission.

Petillo et al. [2010] developed a thermocline tracking algorithm for use within the MOOS-IvP autonomy system [Benjamin et al., 2010]. The algorithm collects temperature and depth data from a CTD sensor and places the readings into depth bins. In GBASIC implementation, one meter depth bins are used. When a dive or climb leg has been completed by the vehicle, the depth bins are averaged. The vertical derivatives, the change of temperature over the change of depth, are then calculated for each bin. The average of the vertical derivatives is used to determine the upper and lower bounds of the thermocline. Any depth bin whose vertical derivative is greater than the average derivative is considered to be part of the thermocline. The algorithm requires an initial dive profile and periodic resets of the depth bin data to ensure variations of the thermocline are successfully detected. For the evaluation, however, resets are not performed as the thermocline data is simulated.

The vertical temperature profile used as the basis for the data in the simulation is shown in Figure 4.4(a). The water column was measured using a Sea-Bird CTD sensor, and the thermocline shown was tracked using a Slocum Glider equipped with a previous revision of the prototype system. The GBASIC code to perform the simulated tracking mission was under 130 lines of code. This included data and GBASIC code

used to fabricate the simulated temperature profile of Figure 4.4(a). The non-optimized compiled code executed by the scripting engine was just over 300 lines of GLOC. The resulting flight profile which successfully performed tracking of the thermocline is shown in Figure 4.7. Algorithms, like the discussed thermocline tracking algorithm, can be easily implemented using the new scripting engine and opens the door to a world of new applications for the glider that were not possible before. These algorithms can be implemented as services in the new programming framework and exposed via language constructs.

4.5 Service Model

An essential component in the new programming architecture is the new service programming model. This service layer provides an additional development level for more advanced engineers and programmers that would like to extend the vehicle's functionality and therefore expand the expressiveness of ALGAE by exposing services as new language constructs. In this model, actions or behaviors are implemented as services to the system. Services may be simple, for example, by providing state information, such as if the AUV is commanded to dive or climb. Services may also grow to become complex by performing data analysis of sensor data that are then fed to computationally intensive models.

The existing infrastructure on many AUVs already contains the elements of such a model. In the Slocum, for example, the number of yos the vehicle has performed is exposed as part of the sensor array. This is a simple example of a counting service. Many such pseudo services are already provided and can be used by new services created using the infrastructure's GLOC or AVBOT.

The driver on the AVBot, shown in Figure 4.1 speaks the gliderbus protocol to read and write into the glider's sensor memory. The runtime system exposes the glider's

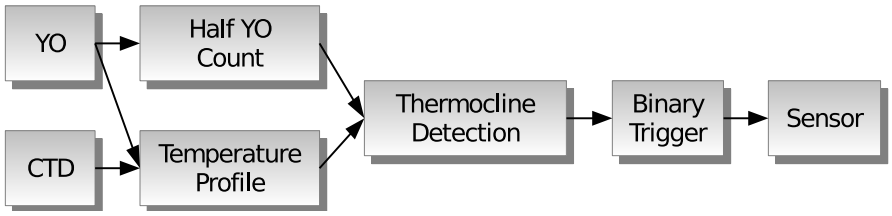


Figure 4.8: Simplified service model to trigger a sensor only within a thermocline.

sensor array for use by services on the AVBot using shared memory, network sockets, or a library. Typically at some level, a sensor is subscribed to by a service so that it may receive updates on that sensor from the glider. A service may also publish updates to the sensor, for example after performing some type of computation. This is similar to the publish and subscribe system used by processes in MOOS-IvP [Benjamin et al., 2010]. It is the responsibility of the runtime system and AVBot driver to interact with the vehicle to request and update any sensor data.

New services can implemented in a variety of programming languages on the AVBot itself. Since the vehicle's sensor memory is also exposed by a network socket, algorithms may be implemented on a remote network node. This is convenient for rapid prototyping and data injection for testing and the creation of fictitious environments for the AUV.

Depending on the complexity, in terms of memory and computation, that is required by a service it may be advantageous for it to be implemented in the GLOC scripting engine. Services implemented using GLOC can be executed on the flight controller, science computer, or on AVBot. Depending on where these services are executed they gain the advantage of data locality. Flight controller services have direct access with no latency to flight information, while GLOC services on the science Persistor have more direct access to the data produced by physical sensors. Finally, these services may also migrate their execution between the computing platforms on the vehicle.

A simplified overview of a service model to energy manage a sensor by triggering it on

only within a thermocline is shown in Figure 4.8. Such a sensor triggering mechanism implemented as services is discussed in more detail in Section 7.7. Existing glider behaviors, like the yo behavior, are pseudo services that, for example, cause state changes within the vehicles control software. The yo behavior produces as output the commanded and measured diving state. This state is observed later in the control cycle and is used to produce a count of the number of dives and climbs in the mission segment thus far.

New services that are created can interoperate with existing services provided by the vehicle and services within the new programming framework. A temperature profile service captures information from the CTD sensor as the vehicle flies through the water column. The thermocline feature detection service has multiple service dependencies needed to perform its duties. Not only does it require the temperature profile service to find the thermocline itself but it also needs to know the number of half-yos performed in the segment because the sensor is forced on for the first half yo of every mission segment.

The thermocline detection service produces binary output indicating whether the vehicle is currently within the thermocline. An instance of a primitive binary service is used to turn on a specific sensor when the detection service's output is true and off if it false. Furthermore, instances of logical and relational primitive services can be used to build more complex services when combined with traditional services. In this programming approach, existing services can be used as building blocks to create new services.

Figure 4.9 shows a service chain where several instances of primitive and more complex services are used to create a new service. The service created in Figure 4.9 triggers a sensor when the AUV is flying at night, or if during the day, only when the vehicle is deeper than 25 m. A Day Time service has service dependencies on the

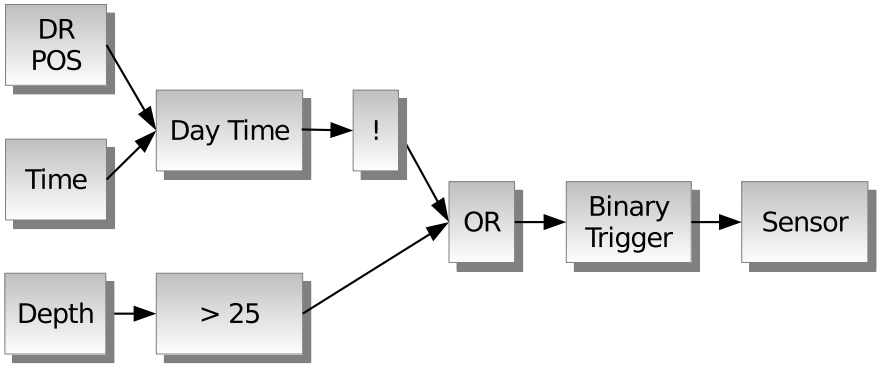


Figure 4.9: Primitive services used to build more complex services.

current dead reckoning position and the current time. The output produced by the service is negated by another service before being used by the OR service. If another sensor should be triggered in the same manner, another binary trigger service would need to be created to use the output produced by the OR service.

Redundant services also exist in the service framework. Such services may provide an alternate mechanism to produce the same or similar output or functionality [Noble et al., 1997; Sorber et al., 2007; Lachenmann et al., 2007; Baek and Chilimbi, 2010]. In the Green framework [Baek and Chilimbi, 2010], for example, several alternative functions can provide a task with varying levels of quality of service. The redundant task used at any given time could be dynamically switched at runtime to meet a user's expected quality of results given the current constraints. The user may in fact never be aware of the switching of services since they could occur automatically as long as the quality of service is maintained. Typically, redundant services provide some trade-off involving computation, memory, or energy requirements. For instance, to provide a notion of the water depth, an altimeter can be used to physically measure the distance to the sea floor and combine it with the glider's current depth. Alternatively, an algorithm could interpolate the water depth using a bathymetric dataset. Likewise, in Figure 4.8, several alternative implementations for thermocline tracking exist of which some are

not likely candidates for GLOC on the glider.

4.6 Domain Specific Programming Language And Compiler

The current programming model on a variety of AUVs including the Slocum Glider is limited and can be difficult to program especially for non-programmers like many oceanographers. To take advantage of the rich infrastructure that we have built, a high level programming language, called *ALGAE* (AUV Language for Greater Adaptability and Energy optimization), and an accompanying compiler have been developed that allows users to easily specify new and dynamic missions for the Slocum AUV. A key component of the language is to use domain specific features to allow oceanographers to naturally make trade-offs, for instance, in terms of sacrificing the quality of the sensing results for a gain in energy conservation to allow for extended deployments. Unlike the service tier of programming, this tier intentionally sacrifices flexibility for safety and usability. The design of the language, however, was guided by the requirements specified by AUV operators at Rutgers University. Although the Slocum is the target platform, many of the language features are applicable to other autonomous systems in other domains.

The most basic view of a mission involves that of states and state transitions. Typically a glider is tasked to perform an action for some time and should only transition to perform another action based on events. The user specifies a sequence of concise states that they wish the glider to be in during a deployment. The transitions between states in the language can only occur at the surface, which is how gliders are operated today. This not only allows users to verify or interrupt a state transition but also enables state transitions to occur based on shore-side information.

Figure 4.10 contains a sample program written in ALGAE containing only a single state. Each state contains a *route* that describes how the AUV should navigate

```
1  mission: simple_mission
2  state: state0
3  begin
4    route:
5      gotowaypoint(3927.0, -7415.0)
6    profile: yo(5, 0.454, 75, -0.454)
7    surface: interval : 10800
8    sensors:
9      ctd: interval 4 diving,
10     fire: interval 0 always
11   events:
12     case interval exit,
13     case waypoint exit
14 end
```

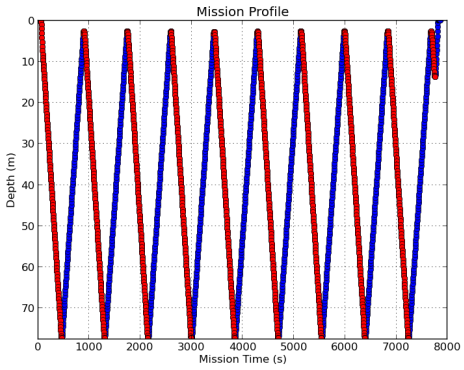


Figure 4.10: A single state program, written in the ALGAE language, that when compiled and executed in the programming infrastructure produces the flight profile on the right. The red data points indicate the CTD is commanded to be turned on, while the blue data points indicate the FIRe sensor to be turned on.

horizontally. This could be as simple as having no heading or flying to a waypoint, to more advanced maneuvers such as sweeping an area specified by a convex hull, chemical plume tracing [Farrell et al., 2005], or flight coordination with another AUV.

The *profile* details how to fly vertically through the water. Examples include a simple *yo* action or more elaborate movements such as flying within a thermocline [Wang et al., 2009; Woithe and Kremer, 2009; Petillo et al., 2010; Zhang et al., 2012; Cruz and Matos, 2010a]. In Figure 4.10 the vehicle is tasked to fly between 5–75 m at approximately 0.454 rad or 26°.

The *surface* section allows the user to specify common surface actions that the state should support. The vehicle could surface periodically, when there has been no recent communication with shore, or when nothing is being commanded to move vertically or horizontally. In the example program, the vehicle is requested to surface every 10800 s or every three hours.

A critical component in the state definition is the sensor specification. In this section, the user lists the sensors that should be logged while the state is active. In the

sample program of Figure 4.10, the CTD sensor, the red data points, should be logged in four second intervals while diving and the FIRe sensor, the blue data points, should be logged as fast as possible only while climbing. As will be illustrated further, services can effectively energy manage such sensors.

As mentioned in Section 2.1, the sensor specification has a similar notion as the type annotations of EnerJ [Sampson et al., 2011]. When a sensor is instructed to be enabled and to log all readings it may be considered precise, like the FIRe sensor in Figure 4.10. However, sensors in the language have additional qualifiers, rather than simply "approximate", that more precisely describe how a flight engineer wishes to sacrifice sensor quality. The CTD sensor's specification in Figure 4.10 is a simple qualifier indicating that it is only necessary to enable and log the sensor while diving at four second intervals. More complex annotations may specify that sensors should stay within some recall and precision or be enabled within an environmental feature as shown in Section 7.7. Some qualifiers implemented as services may be dependent on others, for example, thermocline feature detection is dependent on the collection of temperature and depth sensor data. Thus, some mechanism to "type-check" sensors within the state is an interesting concept left for future research.

Transitions between glider states occur based on *events*. Depending on the route, profile, or surface specified, a set of events can be raised. Similarly, the exceptions that can be raised by programs in other languages depend on the code at hand. The events of Figure 4.10 both perform *exit* transitions if the *gotowaypoint* has reached the waypoint or if the surface interval has been activated.

A mission with multiple states and its simulated flight profile is shown in Figure 4.11. The configuration section in the program is used to inform the compiler of mission options. For example, the compiler can generate a simulation environment for running the test missions, which enables a user to experiment with various trade-offs in the

```
1  mission: multiple_states
2  config:
3    s_ini_lon : -7412.8721
4    s_ini_lat : 3927.25266
5
6  state: state0
7  begin
8    route: heading(0.0)
9    profile:
10     yocnt(5, 0.454, 50, -0.454, 3)
11   surface: interval : 10800
12   sensors:
13     ctd: interval 4 diving,
14     fire: interval 0 always
15   events:
16     case interval state0,
17     case nopitch state1
18 end
19
20 state: state1
21 begin
22   route: heading(0.0)
23   profile:
24     yocnt(5, 0.454, 40, -0.454, 3)
25   surface: interval : 10800
26   sensors:
27     ctd: interval 4 climbing,
28     fire: interval 0 always
29   events:
30     case nopitch state2,
31     case interval state1
32 end
33
34 state: state2
35 begin
36   route: heading(0.0)
37   profile:
38     yocnt(5, 0.454, 30, -0.454, 3)
39   surface: interval : 10800
40   sensors:
41     ctd: interval 4 diving & climbing,
42     fire: interval 0 always
43   events:
44     case interval exit,
45     case nopitch exit
46 end
```

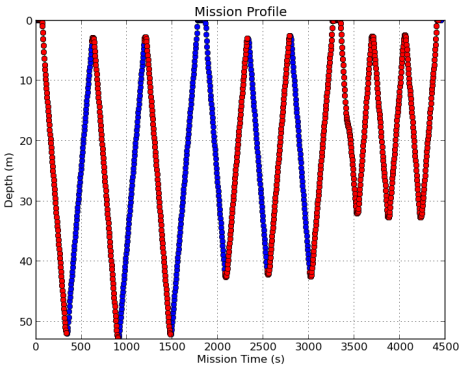


Figure 4.11: A program compiled and executed with multiple states. The red data points indicate CTD activity while the blue indicate FIRe sensor activity.

programs. This section may also provide vehicle options such as the sensors that are onboard. In the program of Figure 4.11, the configuration simply specifies the location that the AUV should have at the beginning of a simulated mission.

To task the vehicle to navigate horizontally in a cardinal direction a heading must be provided in the profile section of the state. The heading is provided in radian degrees with north defined as zero degrees. This is to stay consistent with the behavior based programming model.

A glider may be limited to a certain number of vertical profiles with a *yocnt* definition. Like yo, the target diving and climbing depths to be reached must be provided along with the profile's desired flight angles. The final parameter of yocnt limits the number yos that will be flown. Unlike yo, the yocnt profile will eventually complete and cause the vehicle to surface. Upon surfacing, a nopitch event occurs because of the completion of the yos and a state transition must occur. Although the route and profile statements thus far have been fairly simple, advanced AVBot services may be used that can also generate events used for transitions.

The program in Figure 4.11 contains three states with slight variations in each state. The route in all states directs the vehicle to navigate north. The profile of the states are all limited to three yos, however, the target depths are different. Each of the states also have a different sensor configuration where CTD activity is shown in Figure 4.11 in red and FIRe sensor activity in blue. Because the three yos in each of the states occur within the three hour time interval, all state transitions occur because of *nopitch* events. In the final third state, both the CTD and FIRe sensors are active throughout the flight and the mission is instructed to end on either of the two given events.

A simple single state ALGAE program in Figure 4.12 showcases how sensors can be energy managed by services written in the programming infrastructure. A trivial change in the sensor configuration allows a users, such as an oceanographer, to specify

```
1  mission: thermocline_mission
2  config:
3    s_ini_lon : -7412.8721
4    s_ini_lat : 3927.25266
5    sci_bb2flsV4_is_installed : 1
6
7  state: state0
8  begin
9    route: heading(0.0)
10   profile:
11     yocnt(5, 0.454, 50, -0.454, 6)
12   surface: interval : 3600
13   sensors:
14     ctd: interval 4 always,
15     bb2flsv4: interval 0 thermocline
16   events:
17     case interval exit,
18     case nopitch exit
19 end
```

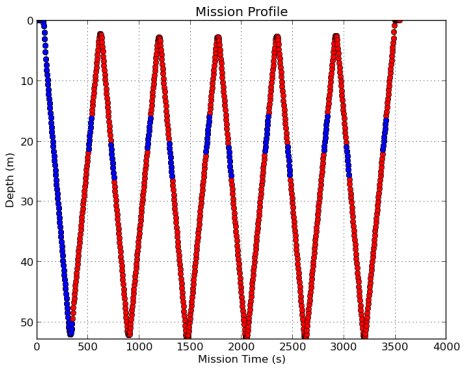


Figure 4.12: A program written in the ALGAE language that uses an AVBot service to only activate a sensor within a thermocline.

an energy conservation mechanism for a sensor. In the sample program, a sensor is activated only while the vehicle is within the thermocline of Figure 4.4(a). The services that accomplish this task are show in Figure 4.8. Domain specific features, such as a thermocline, allow service writers to create a rich trade-off space that are expressed in the language to enable users to explore and extend their missions and to collect useful scientific data.

It is the responsibility of the ALGAE compiler to translate such a high level specification into an executable deployment. A service writer must extend the compiler, through plugins, and decide how to expose the services they have written with language constructs to end users. The service writer must ensure that the compiler uses, as much as possible, existing glider behaviors and services. Flight engineers already familiar with the vehicle can then inspect the generated missions before deploying the vehicle. This transparency is meant to build confidence and trust in the framework as the engineers are quite hesitant when too many new features are added to vehicle at

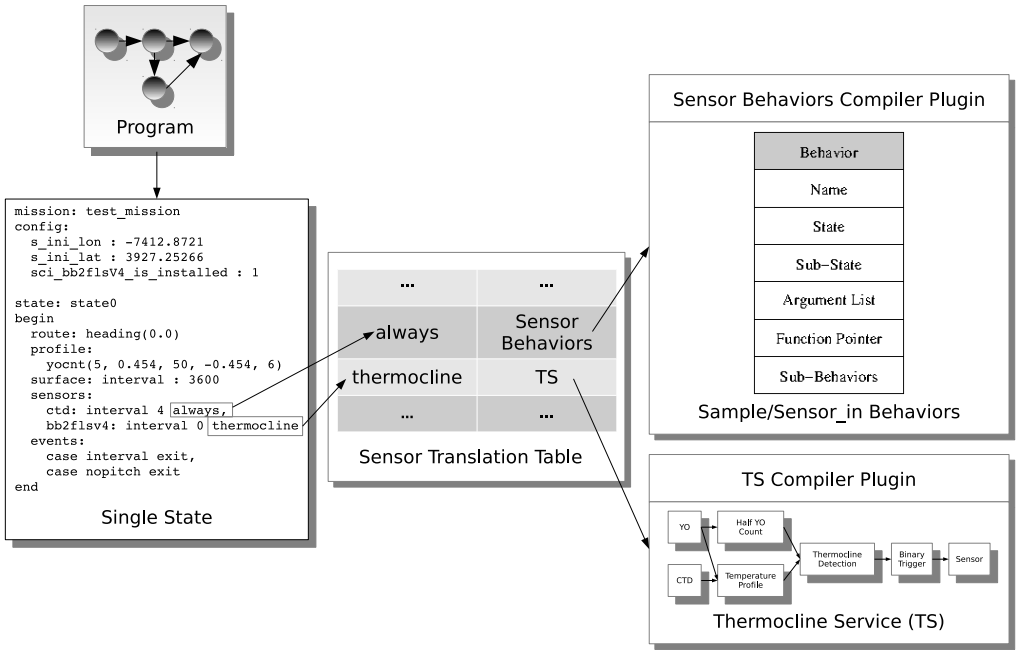


Figure 4.13: Translation of an ALGAE sensor specification to a service.

one time. Finally, because a service writer extends the language, they can decide what optimization and trade-offs their service can make based on any context gained from an analysis of the ALGAE program.

Figure 4.13 depicts, in part, how the ALGAE compiler will generate services for the state specification. These services can target the existing system or the new framework. In the case of the CTD sensor specification, the compiler will lookup the "always" token and use the sensor translation table to determine that *Sensor Behaviors* compiler plugin should be called. These compiler plugins are python modules that are imported and then called during the compilation to help verify, optimize, and generate code to enable or activate the services of the framework. These plugins, when executed, are provided with the state specification that will give the plugins the context to perform optimizations. For instance, in Figure 4.10, the compiler plugin should ensure that two separate sample behaviors are generated in the glider mission file. However, if both the CTD and FIRe sensors were tasked to always log, then the compiler could perform

the optimization of creating a single sensors_in behavior that would cover the logging of both sensors. This is because a sample behavior only enables the logging of a single sensor but is able to turn the sensor on and off when diving, climbing, hovering, at the surface, or always. The sensors_in behavior, on the to hand, allows for multiple sensors to be listed but has the constraint that causes the sensors to always be logged.

The "thermocline" sensor token of Figure 4.13, for the bb2flsv4 sensor, would also need to be looked up in the translation table. The thermocline service compiler plugin would then also be called by the compiler with the state's context. This plugin could take into account the dependent CTD sensor and verify that data resolution is sufficient for the thermocline tracking service to perform its duties. If the resolution is not sufficient, then the compiler will inform the user of the disparity with an error message. Because some services are redundant and can provide similar services with varying levels of quality of results, the compiler may choose a specific implementation of a service that uses less energy but still meets the user's quality requirements. Alternatively, depending on the complexity of the service, a management service may enable the switching of redundant services at runtime.

As mentioned, for each of the states in the program the compiler will generate a mission file to be flown by the AUV. If existing route, profile, and sample behaviors exist that match the semantics of the state, then they should be generated as part of the mission's layered control stack. Again, this is to ensure that as much of the existing components, like behaviors, that have been well tested and are trusted are used by the new framework. Thus, if sensor logging definitions are trivial, multiple corresponding sample and sensors_in behaviors will be created in the necessary order. For example, Figure 4.14 contains two mission files that the ALGAE compiler would generate for two similar states. The mission listed to the right corresponds to the state of Figure 4.12. The mission listed to the left corresponds to a similar state but would instead require

```
sensor: sci_generic_s(nodim) 0


behavior: abend
  ...
behavior: surface
  ...
behavior: surface
  ...
behavior: surface
  ...
behavior: surface
  ...
behavior: set_heading
  ...
behavior: yo
  ...
behavior: sample
  b_arg: sensor_type(enum) 1
  b_arg:
   state_to_sample(enum) 15
  b_arg: intersample_time(s) 4
behavior: sample
  b_arg: sensor_type(enum) 37
  b_arg:
   state_to_sample(enum) 15
  b_arg: intersample_time(s) 0
behavior: prepare_to_dive
  ...
```

```
sensor: sci_generic_s(nodim) 0
sensor: sci_generic_e(nodim) 1
sensor: c_avbot_power(bool) 0
behavior: abend
  ...
behavior: surface
  ...
behavior: surface
  ...
behavior: surface
  ...
behavior: surface
  ...
behavior: set_heading
  ...
behavior: yo
  ...
behavior: sample
  b_arg: sensor_type(enum) 1
  b_arg:
    state_to_sample(enum) 15
  b_arg: intersample_time(s) 4
behavior: gloc
    b_arg: filenum(nodim) 0


behavior: prepare_to_dive
  ...
```

Figure 4.14: Two mission files generated by the compiler for two similar states in ALGAE. The left mission file uses only existing behaviors while the mission on the right uses the new infrastructure to manage sensors.

that the bb2flsv4 sensor is always logged just like the CTD. The difference between the two missions is that a sample behavior has been replaced by a GLOC behavior that could perform the thermocline trigger service of Figure 4.8. The AVBot, in this case, has been disabled at the beginning of the mission but could also perform the triggering service. The GLOC script could transfer the control of the sensor triggering service at runtime by powering on the AVBot by setting the c_avbot_power sensor value to one. This would be an example of a hierarchical power management technique used in the framework. Other vehicle configuration files such as the vehicle sensor configuration,

simulation settings, and high density logging sensors can also be generated by the ALGAE compiler.

As part of the glider mission file generation the compiler ensures that a sensor variable is set to indicate the vehicle's current state. In Figure 4.14, the sci_generic_s variable is set to zero for a single state mission. This state variable is initialized when the AUV loads the mission file for execution. Upon surfacing, this state sensor is printed as part of the surface dialog message that summarizes the glider's status. A shore side piloting tool, also part of the infrastructure, interacts with the Dockserver mission control software to communicate with the glider. Because services in the framework may be provided outside of the glider, the existing Dockserver XML scripting engine is not used.

The *GPILOT* piloting tool communicates with Dockserver to execute the ALGAE program based on the configuration and script files generated by the compiler. When the tool recognizes that a surfacing has occurred, it will read the state variable in the surface dialog and determine which state in the program the vehicle is currently in. GPILOT will then need to determine the event that caused the surfacing, such as a communications timeout, and look up which state to transition to next. The event can be established by the mission dialog sent to Dockserver by glider behaviors, or by flags raised in sensor variables by services. To determine the next state, a state transition table also generated by the ALGAE compiler is used by GPILOT tool to task the vehicle to execute its next mission.

Early versions of the language allowed for state transitions to not only occur at the surface, but also within dive segments. This approach, however, had some issues. Programs written in the language were too reliant on the new programming framework to execute the mission rather than the existing system. The only way to successfully perform the deployment was to have a single template mission that used the Hook

and GLOC behaviors to generate all of the state's behaviors. Even simple states that could be represented with the current glider programming constructs would need to use the Hook and GLOC behaviors. So this approach directly violated one of the key requirements of being as transparent as possible so that flight engineers could inspect generated missions.

Another issue with this approach was that it could cause some confusion. If a state transition occurred while underwater, a user may or may not expect the surfacing timeout to be reset. If it is reset, then a program that often transitions from state to state may never get an opportunity to surface. If it does not reset and the new state has a lower surfacing timeout, the state may then immediately surface and possibly even transition to another state. This behavior may be unexpected by a user.

Shore-side services are an important element in the current design of ALGAE. Many services are envisioned that require processing and data input from sources outside the glider. For example, a path planning system could use ocean current information from weather prediction models. In the earlier language revisions that allowed for state transitions while underwater, a route specification that made use of such a path planning system would not be feasible. For instance, suppose a program is currently in a state that instructs the glider to head north, and while underwater, a transition occurs to another state with a path planning route, the system in this case has no capability to contact the shore-side service. The only alternative would be for the state to force a surfacing and contact the planning system for a waypoint list. Again, this behavior may be unexpected by a user who does not expect the vehicle to surface until a surface timeout. For the aforementioned reasons, the constraint of state transitions only occurring at the surface was chosen as the better design alternative. This approach is also similar to the way deployments are already executed by flight engineers, where missions are only changed or sequenced when communicating with the control center.

Thus, the framework provides a familiar interface to the end user.

The general programming abstractions of ALGAE are not limited to gliders or even AUVs, but are applicable to other autonomous cyber-physical systems. One can imagine non-expert programmers controlling an autonomous air robot, such as a drone, at a high level abstraction in a state and state transition based model. A drone, like an AUV, carries on board a limited supply of energy in fuel or batteries, so careful management of this resource is critical to a successful mission. They also have limited and only periodic communication with a control station. This may be caused by intentional radio silence in enemy territory or by "dead-zones" caused by geological features.

A drone, in ALGAE, may be tasked to perform a certain action for some time and only switch to performing another action when certain events have occurred. In each state, a drone will have to navigate a certain route, for example, using a waypoint list generated by a path planning system that makes use of favorable winds. Like in the underwater domain, a remote control center will be more effective in aggregating data and running models to create an optimal waypoint list than the drone would by itself. How the vehicle accomplishes the flight to the target waypoints is analogous to the profile specification and could determine the altitude profile the vehicle should fly. What is currently the surface specification would need to be renamed, however, it would still function as a way to express when or how the autonomous vehicle must make contact with a command center in the future. A drone, for example, may need to periodically fly into friendly territory or above a safe altitude before communicating. Thus, since other systems may also transition on events, including events based on a command center's service, the general programming approach is viable for other autonomous CPSs.

The new programming infrastructure created enables complex missions to be expressed at a high level of abstraction and uses domain specific features to encourage

trade-offs. Programs in the ALGAE language are compiled to produce vehicle missions files that make use of services on the glider as well as on AVBot. The GPILOT tool together with the Dockserver and vehicle execute the generated missions. An energy model service, Chapter 5, can be used in conjunction with the simulators, Chapter 6, to experiment with such trade-offs before the Slocum AUV is ever deployed.

Chapter 5

Power Measurement Infrastructure

The mission endurance of today's AUVs depends highly on the capacity and usage of the vehicle's batteries. Typically, missions for the Slocum Electric Glider last about 30 days [Teledyne Webb Research]. Longer missions, such as the 221 day mission to cross the Atlantic by RU27 from Rutgers University are possible through an increase in the number of batteries and through the careful planning of the usage of the vehicle's devices. Such planning is also crucial for shorter missions when gliders are equipped with advanced sensors such as an Acoustic Doppler Current Profiler or acoustic underwater communication.

With the recent integration of the coulomb meter into the glider, measuring the discharge of the battery has become more accurate. Knowing the rate at which energy is used and how much remains is vital to mission planning. However, the glider's coulomb meter only measures whole vehicle current. To perform more precise mission planning, being conscious of the energy consumption of individual components is necessary. As part of this work, a measurement infrastructure that captures the currents drawn from distinct components of the Slocum Glider has been developed. The infrastructure has been deployed in several missions off of the coast of New Jersey, and the data collected have been integrated into the Slocum Glider simulators. The measurement board and simulation framework can be used to assist in the planning and decision making of missions and shows possible trade-offs, for instance, between mission duration, speed, and energy consumption. These trade-offs and restrictions of energy are also incorporated

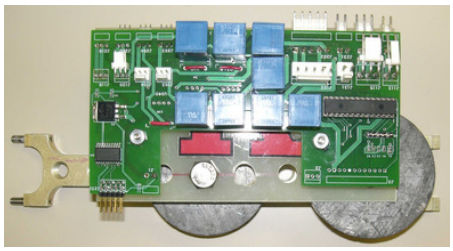


Figure 5.1: Measurement board mounted on a weight bar used for ballasting the Slocum Glider.

as part of the new programming specification of the vehicle via energy aware services.

The infrastructure consists of a measurement board and a data logger. The design philosophy in creating the infrastructure was to not compromise the safety of the vehicle, even if quality of the resulting measurements are affected. The glider components measured are: the main, external, and emergency power, the buoyancy pump and brake, and the pitch and fin servos.

The measurement board, shown in Figure 5.1, was intended to be housed above the glider's mainboard in the aft section of the vehicle. However, due to the different space constraints between different generations of gliders, the board was moved to the science payload bay. This allows the board to be quickly uninstalled and re-equipped onto another glider.

The board makes use of eight Hall Effect sensors which do not interfere with the vehicle's current flow. This ensures that in the event of sensor failure, the glider will continue to operate normally. Three 20 A sensors are used for the main, external, and emergency power, while two 5 A sensors are used for the buoyancy pump. Three 3 A sensors are used for the buoyancy pump brake, pitch servo and fin servo. The sensors were over-provisioned for safety, but still allow the capture of large spikes in the current.

Description	Channels	Power (Watts)
Deployed (02/2010)	8	0.76
Intermediate	6	0.58
Deployed (08/2010)	4	0.42

Table 5.1: Measurement board power consumption

Software	Clock Rate (kHz)	Power (Watts)
Stock	3680	0.19
Deployed (02/2010)	14720	0.71
Intermediate	3680	0.35
Intermediate	7360	0.49
Deployed (08/2010)	7360	0.47

Table 5.2: CF1 processor power consumption

As described, the original revision of the measurement board contained eight channels. However, throughout sea trials several of the Hall Effect sensors have been removed. For example, following the first long-term deployment equipped with the infrastructure, it was found that there was little use in measuring the external and emergency current. The external power supply is only active on the benchtop, so it is unnecessary for a board which will be deployed at sea. In the event of a emergency, the safe recovery of the vehicle is of higher priority than collecting good data. The presence of the emergency sensor could also not be justified for the additional power it consumes.

The microprocessor used in the design of the measurement board was the PIC16F767. The processor typically operates at less than 2 mA at 8 MHz. It contains eleven 10-bit analog-to-digital (A/D) channels of which eight are in use to measure the currents drawn by the glider using the Hall Effect sensors. The microprocessor has been programmed to use interrupts to generate constant samples at 32 ms intervals. These samples are transmitted to the glider's science bay processor for logging.

The board communicates its samples via a 9,600 baud serial connection to the science bay processor. The stock 6.38 software version of the glider's science computer software has been retrofitted to record the samples produced and transmitted by the

measurement board. The science processor is typically clocked to run at 3.68 MHz using the stock software, but is usually run at higher clock speeds when collecting data as part of the power measurement framework.

The power consumption of the measurement board and the science processor are shown in Table 5.1 and Table 5.2. As described, throughout the revisions of the board, some sensors were removed to reduce its power consumptions or were not needed as part of the mission requirements. An additional benefit is the fact that the board and the science processor now measure, transmit, and log less samples allowing for more data to be collected.

The CF1 science processor as programmed during the initial deployments and experiments, was clocked at 14.72 MHz. This consumed approximately 520 mW more power than the stock software release. Throughout the development optimizations in the logging process to reduce the overall energy consumption were developed. These improvements allowed the clock speed to be lowered down to 3.68 MHz provided that the mission specifications allow for the trade-off of four second, instead of two second, sampling from the CTD sensor. The CTD is a standard sensor on a Slocum Glider and as a precaution, deployments were still run at 7.36 MHz to ensure that neither the CTD data or power measurements needed to be compromised. With local science data logging now possible with newer glider software, this may no longer be necessary.

The measurement infrastructure has been extensively tested to ensure that recorded current samples are representative of the actual events. Figure 5.2 shows the results of a test where a current of 1 A was applied to one of the sensors for approximately six seconds. The event was measured and logged by a Tektronix MS04034 oscilloscope as well as a PC connected to the measurement board. The results of these experiments indicate that the samples collected are within the expected error of the sampling rate, A/D conversion and the sensors themselves.

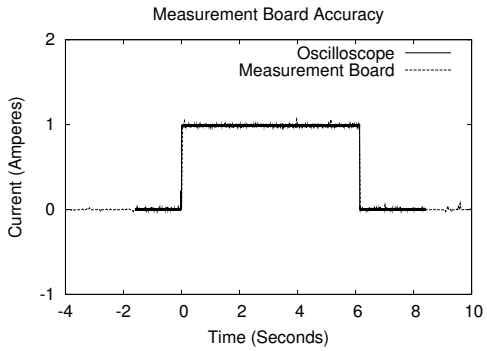
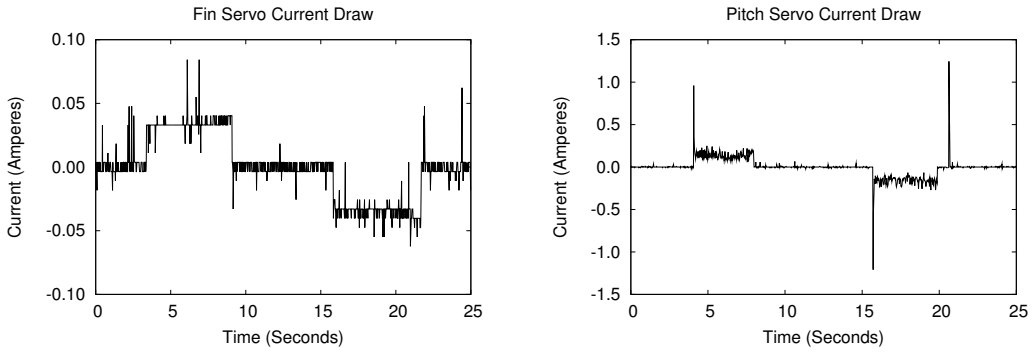


Figure 5.2: Assessment of the measurement board's accuracy using a Tektronix MSO4034 oscilloscope.

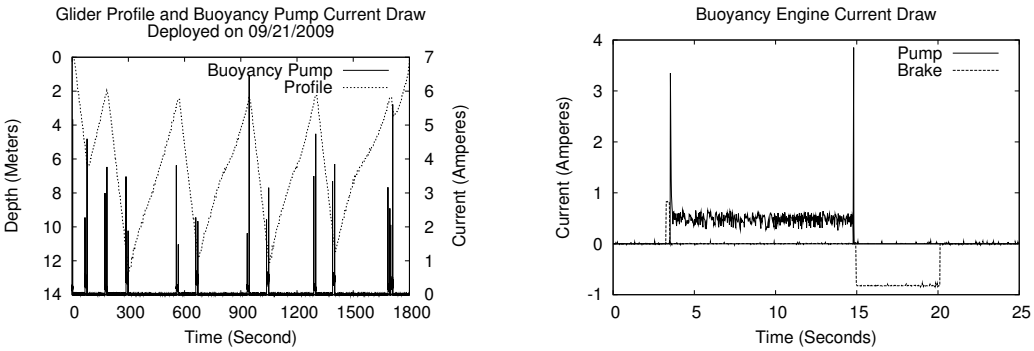
Without compression, data can be recorded for mission lasting up to 30 days. However, multi-week missions using the first revisions of the board with alkaline batteries were not feasible due to the significant energy overhead that was introduced. They may have been possible if lithium batteries were used instead. In future work, the power dissipation of the system will need to be significantly reduced if full length deployments with all channels are to be recorded. Meanwhile, the number of sensors were reduced to only those components of most interest to allow for extended deployments.

To ensure the vehicle components still performed up to par with the presence of the measurement board, the vehicle's motors were subjected to *wiggle* tests. This entails the moving of its motors through their full range of motion. Sample results of such a wiggle of the fin and pitch servos are depicted in Figure 5.3(a) and Figure 5.3(b), respectively. The Hall Effect sensors used for these devices are bipolar so the reported currents show the current flow in both directions as the servos move the opposite direction. The fin is used to steer the vehicle, and the pitch motor is used to fine tune the vehicle's commanded pitch by moving an internal battery pack. The power draw of these two motors is generally very low, and during a mission motor activities typically occur in brief bursts. Through wiggle, overnight, and weekend tests the system was deemed stable and reliable for sea trials. The measurement infrastructure has been installed



(a) (b)

Figure 5.3: Current draw of the fin (a) and pitch (b) servos during a wiggle test.

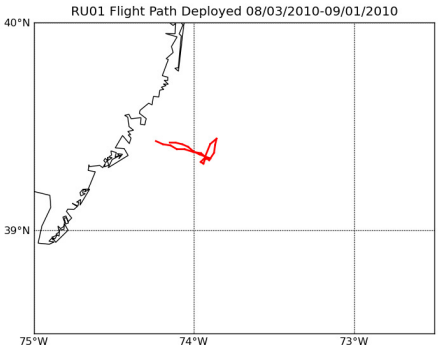


(a) (b)

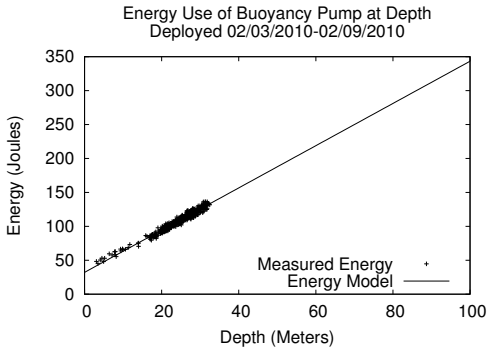
Figure 5.4: (a) The flight profile shown together with the current draw of the buoyancy engine during a deployment in September 2009. The activity of the buoyancy engine aligns with inflection points where it can be observed that the power consumption at depth is significantly higher than near the surface. (b) Current draw of the buoyancy engine during an inflection at approximately 12 m.

and deployed on three separate Slocum Gliders. The trials took place off the coast of New Jersey in September 2009, and in February and August 2010.

The first sea trial involved two short mission segments of approximately thirty minutes in length each. The glider was instructed to perform yos (sequences of dives and climbs) between 1–20 m. The glider's depth profile along with the current draw of the buoyancy pump of one mission of the segments are illustrated in Figure 5.4(a). The glider never reached a depth of 20 m because the ocean floor was not sufficiently deep enough at the deployment location. The experiences gained in the trials were used to



(a) Flight path of the mission.



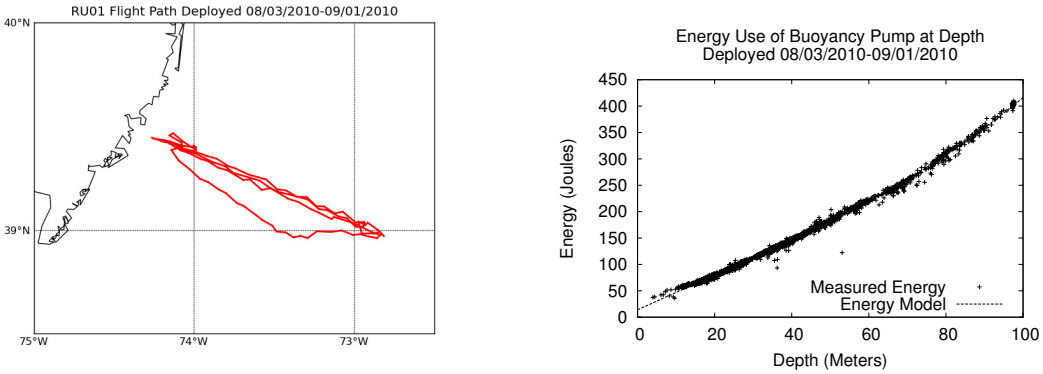
(b) Energy use of the buoyancy pump at depth.

Figure 5.5: February 2010 deployment with power measurement infrastructure.

prepare the infrastructure for longer term missions.

The second deployment was a 6.5 day mission in early February of 2010. A map of the glider's path is shown in Figure 5.5(a). The mission's goal was to fly to the continental shelf to gather buoyancy engine readings at depths of up to 100 m. The mission was however cut short due the combination of inclement weather and the high power consumption of the measurement infrastructure. After heading east toward the shelf for two days, the vehicle was commanded to head north because a Nor'easter storm was expected to push the vehicle south. After being forced south for two days, it was again commanded to head east towards the shelf to gather readings at deeper depths for a short time. Unfortunately, another Nor'easter was imminent so the mission was aborted and the glider spent the remaining time flying back to shore to be retrieved.

The buoyancy engine of the Slocum Electric Glider consists of a buoyancy pump and a brake mechanism. The pump moves a piston to change the vehicle's buoyancy by altering its displacement in water. The brake locks the pump's position in place which would otherwise be forced to retract due to water pressure. The current draw of the buoyancy engine is shown in Figure 5.4(b). When commanded to inflect from a dive to a climb, or from a climb to a dive, the brake first unlocks the pump. The pump



(a) Flight path of the mission.

(b) Energy use of the buoyancy pump at depth.

Figure 5.6: August 2010 deployment with power measurement infrastructure.

follows by moving the piston to the commanded position. When the desired position is reached, the brake again locks the pump's position in place.

The energy used by the buoyancy pump increases with depth because the pump must work harder to battle the additional water pressure. This was confirmed by the first sea trials, Figure 5.4(a), where inflections from a dive to a climb state used more energy when the inflections occurred at three, six and twelve meters. Figure 5.5(b) depicts the measured energy used by the pump during the deployment in February of 2010. The energy used for similar depths in the two seal trials were comparable considering that different gliders were used. In both missions, however, the energy necessary for the pump to perform inflections from a climb to a dive at shallow depths is at times less expensive than the cost associated with the brake.

Although the February 2010 deployment provided a good insight into the pump's energy use, much of a glider's time is spent at depths greater than 35 m. As a glider approaches the deeper waters towards the continental shelf off of the coast of New Jersey, it can experience very strong currents and may need to be carefully maneuvered to get the vehicle back to shore. During this critical period the more detailed the energy models of the pump and other components are, the better the prediction of the glider's

battery state will be.

In August 2010, another glider equipped with the measurement board was deployed. Whole vehicle and the buoyancy engine currents were logged during the flight. The mission objectives were to fulfill the buoyancy engine energy cost model up to 100 m, the standard buoyancy pump rating for many of the coastal gliders at Rutgers. Figure 5.6(a) shows the vehicle's flight path for this mission lasting four weeks. The observed energy readings from the measurement board for the use of the pump at depths is depicted in Figure 5.6(b). Previous sea trials suggest the increase in energy with depth should be linear. The data gathered from the August 2010 deployment however is not quite linear. This suggests that the efficiency of the motor decreases towards its motor's maximum rating. Having this kind of detailed knowledge of the cost of components is important when trying to optimize vehicle flights.

The energy measurements gathered from deployments, benchtop experiments and component specifications have been compiled into energy models for the Slocum Glider. The service that implements the models estimates the glider's energy usage by reading the state of the vehicle's devices through the sensor array. This allows the service to live on any of the computing platforms available on the glider. However, if the service is implemented as a device driver for the glider it could run multiple times a cycle and thus be more precise for many devices. Furthermore, the energy of previously flown missions can also be estimated by injecting the service with the logged sensor data at each control cycle.

Table 5.3 contains energy estimates of previous glider deployments measured with coulomb counters and estimated using the energy models. The measured energy column is calculated using the coulomb counter's integrated Ampere-hour readings and the battery voltage logged by the vehicles at every four second cycle. The modeled energy column is calculated by injecting into the energy model the values of all sensors of a

Glider	Pump Rating (m)	Sensors	Measured Energy (mJ)	Modeled Energy (mJ)
RU28	30	CTD, Optode	6.1	5.7 (93%)
RU28	30	CTD, Optode	4.7	4.3 (91%)
RU28	30	CTD ,Optode, FLBBCD	6.0	5.3 (88%)
RU16	100	CTD, Optode, BBFL2SLO	6.2	5.7 (92%)
RU16	100	CTD, Optode, BBFL2SLO	6.0	5.3 (88%)
RU16	100	CTD, Optode, BBFL2SLO	5.6	5.3 (95%)

Table 5.3: Energy estimates of glider deployments.

glider's log file at each cycle. Assuming that the coulomb counter has been correctly calibrated, the energy models slightly miss estimates the energy used by the two gliders of Table 5.3. RU28 is equipped with a 30 m and not a 100 m pump that the energy model is based on so some errors are expected. The power requirements of the FLBBCD and BBFL2SLO fluorometer and backscatter sensors were also estimated and based off of the measurements of similar sensors. Clearly, more measurements and field trials with different gliders and sensor packages are necessary to refine the energy models. Nevertheless, the current models can come quite close to energy measurements of real deployments and is a viable tool for mission planning.

Finally, the energy service can also output the energy used up to a given point into a mission for each of the devices it is monitoring. This enables the creation of energy aware services that could, for example, sacrifice functionality or data resolution for a gain in energy, such as in Levels or Green [Lachenmann et al., 2007; Baek and Chilimbi, 2010]. Along with the simulation infrastructure and an environmental model, the energy models can estimate the endurance of missions and provides a setting to explore any adjustments made by a mission programmer using the framework.

Chapter 6

Simulation

The longevity of missions performed by AUVs rely heavily on the limited energy resources the vehicle carries on board in its batteries. This resource limit can affect the quality of missions. Missions which require the vehicle to maintain a constant presence at a location or require traveling to an area of interest are constrained in the amount of information they can collect. Because of this energy reliance, a power measurement infrastructure was built and used to create a vehicle energy model. To assist in the planning and development of future missions, two Slocum Glider simulators have been developed as part of the programming framework that make use of the energy model; the first simulator is driven by a speed distribution model, while the second is a software port of the glider's control software.

The speed model based simulator incorporates energy, speed, seafloor and sea surface current models and is used to predict the flight path, longevity, and energy usage of a mission. The simulation environment has been validated against Teledyne Webb Research's *Shoebox* simulator and compared to a deployment off the coast of New Jersey. This simulation environment, although more coarse in many aspects than the software port simulator, still provides a good testbed for mission development.

The second simulator created for the programming framework is a partial software port of the Slocum Glider's control software. The simulator has been retrofitted to not only simulate in real-time like the *Pocket*, Shoebox, or a benchtop glider simulators, but also to simulate faster-than-real-time. This enables for rapid prototyping of new

Slocum Glider services and more importantly, it can quickly provide detailed feedback when exploring possible trade-offs in ALGAE. This simulator environment is essential in compatibility testing before deployment to ensure that safe and correct missions are generated by the systems for the vehicle.

The focus of the two simulators developed for the framework differ from other model based glider simulators [Bhatta and Leonard, 2002; Graver et al., 2003; Stante et al., 2007; Arima et al., 2009; Mahmoudian et al., 2010]. Unlike other simulators, both of the simulation environments incorporate the energy model to estimate vehicle endurance and sensor usage. The speed model simulator is simple in design can be used for quick approximations, while the software port simulator is more complex and includes Slocum Glider specific idiosyncrasies. However, components of the related work could be incorporated as alternative mechanisms in portions of the simulators, for example, to estimate the vehicle's motion.

The two energy aware simulators of the programming teamwork are described with greater detail in this chapter. Without such simulation environments the trade-offs specified in the domain specific language would be difficult to quantize and deployments more difficult to realize.

6.1 Speed Modeled Simulator

The speed modeled simulator takes a higher level and more coarse approach to performing a virtual deployment than the software port simulator. It is relative simplistic and incorporates a speed model derived from previously flown glider missions to fly the AUV. Because there is also a great concern for the vehicle's remaining energy throughout a deployment, the simulator is also equipped with the energy models.

The energy models were formulated from the samples recorded by the power measurement infrastructure along with the voltages reported by the glider. The simulator

uses models for the buoyancy pump, brake and steady state load, where no motor and most devices are not in use. The average observed cost associated with the brake is applied at every inflection point. The expense of inflections near the surface where the vehicle state changes from a climb to a diving state is modeled as a constant cost. Inflection performed at depth from a diving to a climbing state previously used a linear cost function. This function was initially fitted to the data points from the deployment in February 2010 and is shown in Figure 5.5(b). The model has since been updated to model the data acquired from a more recent deployment from August 2010, shown in Figure 5.6(b).

The energy used in simulated missions is dependent on the vehicle's pitch angle and speed. The pitch of the flight impacts the number of inflection points, and thus the use of the buoyancy engine. The speed also determines the amount of time required to complete the mission.

The simulation environment makes use of two types of speed models. The first is a model similar to that of the Slocum Glider's Shoebox simulator. The Shoebox, named after its physical similarities to a shoe box, contains the essential glider electronics to perform simulations in real-time. The software running in the Shoebox is the same software used during deployments but makes use of simulated device drivers. The speeds and missions generated by this model when used in the simulator should be similar to that of the commercial Shoebox. However, unlike the Shoebox, the modeled simulator is able to simulate missions significantly faster than real-time.

The second speed model integrated into the simulator is based on speed distributions which were empirically derived from over four years of glider flight data. The flights took place off the New Jersey coast between the years of 2003 and 2009. The resulting distributions are shown in Figure 6.1 and were constructed by measuring the distance covered in each dive segment and the time necessary to travel the segment.

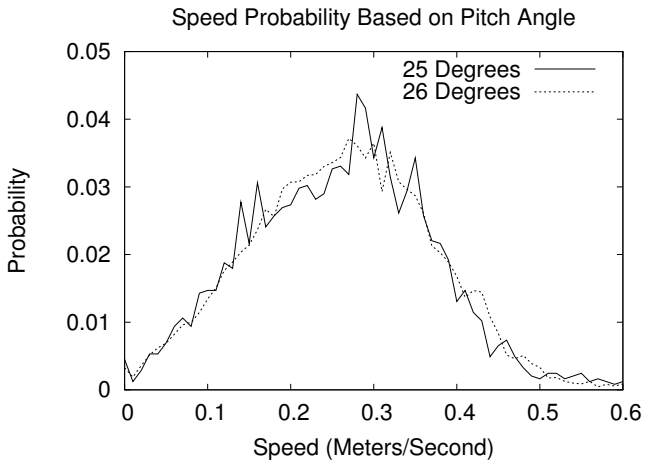


Figure 6.1: Speed distribution derived from over four years of glider flights.

A dive segment starts when a glider submerges and ends when it resurfaces. The 25° distribution was comprised of 2,539 segments, covering 6,263 km over 293 days, while the 26° distribution span 16,411 segments, 32,527 km and 3.48 years. Sufficient data to build speed distributions were available only for 25° and 26°, which are the most common angles used to fly the Slocum Glider. These speeds are sampled by the simulation environment to produce realistic over-the-ground speeds. Although very similar, the 26° distribution is slightly faster than the 25° distribution. Along with the dive and climb pitch angles specified by the mission, the depth rate is calculated and used to position the glider in space. The depth rate and the seafloor determines the number of inflections that occur during flight.

The simulation environment also supports the use of a seafloor terrain. The seafloor model used may be artificial, come from prior deployments as measured by a glider, or can be interpolated from NOAA's National Geophysical Data Center's (NGDC) bathymetric data set [National Oceanic and Atmospheric Administration]. The current data set (from the NGDC) used by the simulation environment is of the coast of New Jersey at a resolution of one arcminute. The addition of a seafloor model improves the quality of the vehicle's predicted energy usage especially in shallow waters. Simulated

open ocean deployments, or deployments where it is known that the glider will never reach the seafloor will not benefit from a seafloor model, and could therefore be removed for such missions.

Time dependent sea currents can significantly impact the flight of a glider, and are therefore modeled within the simulation framework. The currents may be artificially and dynamically generated, or can be interpolated much like the seafloor. The use of Coastal Ocean Dynamic applications Radar (CODAR) [Evans and Georges, 1979; CODAR Corporate Headquarters] data from Rutgers University has been integrated into the framework. This data describes the sea surface currents off the New Jersey area at a spatial resolution of six kilometers and a temporal resolution of one hour. The addition of sea currents adds another degree of realism which should improve the prediction quality.

The simulation framework can be used to analyze past glider flights, support active deployments, or help to plan future missions. CODAR information is valuable when simulating past flights and can be used in the decision making of active missions. For example, if recent sea surface current data is available, it can be used to predict the location of where the glider may resurface next. With the utilization of weather trend or prediction models, such as the Regional Oceanic Modeling System (ROMS) [Shchepetkin and McWilliams, 2005] and the Hybrid Coordinate Ocean Model (HYCOM) [Bleck et al., 2002], the simulator could also forecast the general outlook of missions.

6.1.1 Shoebox Simulator Validation

To validate the modeled simulation infrastructure, the predictions it produces are compared to that to that of TWR's Shoebox simulator. The mission executed on both the Shoebox and the simulation framework entailed three yos between 2–25 m.

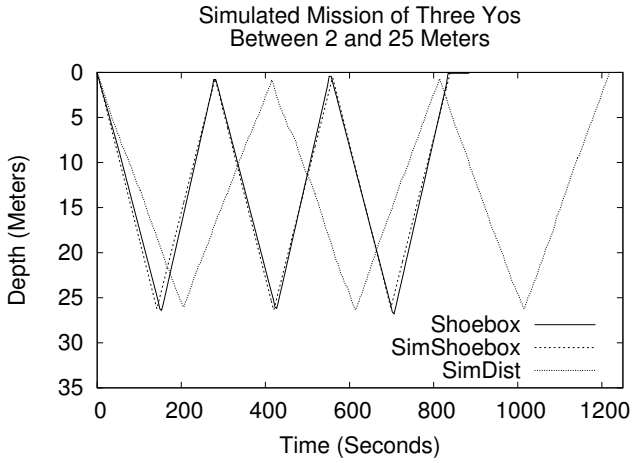


Figure 6.2: Validation of the simulation environment with respect to the Shoebox simulator.

The depth profile of the simulations are shown in Figure 6.2. The Shoebox profile describes the flight as performed by the Shoebox simulator. SimShoebox and SimDist are the flight profiles generated by the new simulation environment. SimShoebox generates speeds similar to that of the Shoebox, while SimDist samples speeds from the distribution in Figure 6.1.

The time necessary to complete the missions for Shoebox and SimShoebox are very similar. Like the Shoebox, the vehicle simulated using the new simulator also slightly overshoots the commanded depth limits. On average, the SimShoebox is slightly slower, taking several seconds longer to complete the mission. The results produced are however a reasonable representation of what may be generated by the manufacturer's simulator. The advantage of the new simulation framework lies in the runtime necessary to produce the simulated mission. The Shoebox took approximately 15 min to simulate the sample mission, while SimShoebox required only 0.35 s on a 2.2 GHz dual core processor.

The flight simulation which applies the speed distribution model, SimDist, requires an additional 380 s longer in mission time than both the Shoebox and SimShoebox. The simulation itself took only 0.5 s. This suggests that the vehicles speed is on

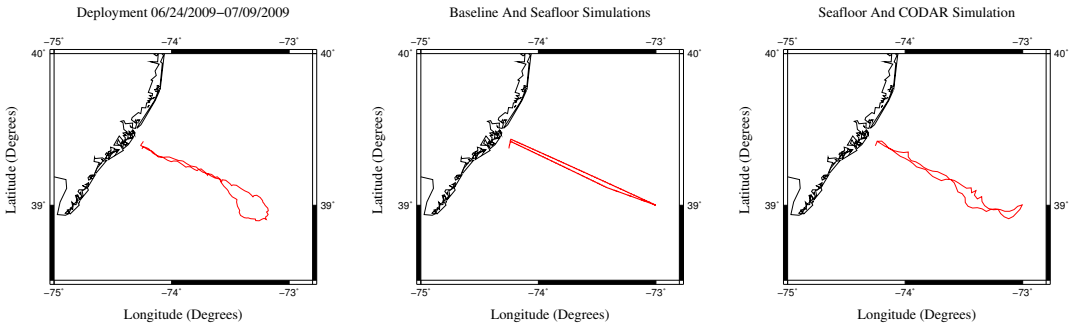


Figure 6.3: (a) The flight path of the mission being simulated. (b) The flight path of the baseline and seafloor simulations. (c) The simulated mission using both seafloor and CODAR data.

average slower using this model than that of the Shoebox. The speed model based on speed distributions is believed to be more accurate than the Shoebox model since it is derived from over fours years worth of vehicle flight time. The speed distribution model should then not be compared to that of the Shoebox simulator but against an actual deployment.

6.1.2 Deployment Validation

To continue to validate the simulator and its speed distribution model, a deployment from September 2009 is compared to similar flights in the new simulation environment. The goal of the original mission was to fly to the continental shelf from the coast of New Jersey and back with a 26° flight pitch angle. The flight path of the mission is illustrated in Figure 6.3(a). Due to strong currents for portions of the mission, the glider was pushed south preventing it from making steady forward progress towards the target waypoint. An operator interfered with the flight and changed the target waypoint due west back to shore before the vehicle reached the commanded waypoint near the continental shelf. Waypoints were changed further throughout the mission, causing the vehicle to reach none of the target waypoints except the last which was

Mission	Seafloor	Currents	Time (days)	Energy (kJ)	Runtime (min)
Actual	N/A	N/A	14.84	N/A	N/A
Baseline	No	No	11.5	785	1.4
Seafloor	Yes	No	11.5	984	5.7
CODAR	Yes	Yes	14.89	1,235	20

Table 6.1: Speed Distribution Simulation Results.

used to collect the vehicle. The total length of the deployment was 14.84 days. To validate the simulation framework a similar deployment length should be achieved. The energy costs in this section used the energy models of Figure 5.5(b).

Baseline

The baseline simulation assumes that no seafloor or currents exist in the environment. Consequently, the runtime needed to simulate the mission is small and also not be very accurate. The missions flown in the remainder of this section is inspired by Figure 6.3(a) except that the vehicle will be commanded to keep flying until it has reached all its waypoints. It is difficult to reenact the intentions or reasoning behind the operator's actions so they are ignored.

The SimShoebox simulation of the mission predicts a mission length of 7.9 days, with the energy usage of 707 kJ and a flight path as depicted in Figure 6.3(b). A runtime of 20 s was needed to simulate the mission. SimShoebox, which has been shown in the previous section to be fairly representative of the Shoebox simulator, would suggest a real-time simulation of 7.9 days. If the speed distribution is used instead in the simulation (SimDist), the mission length increases to 11.5 days, 785 kJ and a runtime of 86 s. SimShoebox in this scenario has erroneously estimated the mission length by 6.94 days while SimDist by 3.34 days. Unlike the previous validation experiment, the speed distribution produces a better estimate when compared to a real deployment.

Seafloor Model

To add a layer of realism, the simulation environment can use a seafloor as previously described. Instead of flying to the full commanded depth, the vehicle must inflect several meters above the seafloor to avoid impact. This will increase the total number of inflections points in the mission which directly translates into more energy use because the buoyancy engine is activated at each inflection. The mission length and flight path for both SimShoebox and SimDist remain nearly identical to the baseline but the energy usage increase to 892 kJ and 984 kJ, respectively. The modeling of the seafloor is paramount so that missions may be more accurately predicted and planned for.

Seafloor And CODAR Models

The final model supported by to modeled simulator is that of sea currents. The CODAR sea surface currents of the days surrounding the deployment of Figure 6.3(a) were integrated and applied to the simulated mission. The flight map of SimDist is shown in Figure 6.3(c). The SimDist mission flew for 14.89 days using 1,235 kJ of energy and required 12 min to simulate. SimShoebox's mission flew for only 8.88 days, used 986 kJ, and had a runtime of 5 min.

The presented simulation results indicate that the speed distribution model was more representative of the deployment in Figure 6.3(a) than that of the speed model which is similar to the Shoebox. A summary of the simulations for SimDist is listed in Table 6.1. The final SimDist mission using both the CODAR and seafloor resulted in a mission time slightly longer than that of the real deployment. This is however expected since the simulated mission flew a slightly different mission where the vehicle actually reached the waypoints and was not interrupted by an operator. Modeling the supervision as part of the mission is a difficult task because the intentions of the operator at the time are not known. Errors associated with the spatial and temporal

resolution of the seafloor and CODAR data also add to the difficulty of recreating the original mission.

6.2 Software Port Simulator

The Slocum Glider uses a layered-control programming architecture that determines mission sensing and control actions in cycles. Typically, the duration of a cycle is four seconds. During these four seconds, a rather complex interaction among different drivers for sensors, motors and software components is performed. These complex software and hardware interactions make it difficult to design a high level behavior model of all activities. Instead of modeling the software and its behavior on the vehicle, a significant portion of the glider's software system has been ported to actually perform these complex tasks purely in software. The software ported simulator is capable of running faster-than-real-time and includes vehicular, environmental, and energy models to determine the glider's behavior during mission execution and is capable of running on commodity hardware.

A typical Slocum simulator is either a physical glider on a bench top running in simulation mode, a Shoebox simulator, or a Pocket simulator. A Shoebox simulator contains much of the electronics of a glider contained in shoebox sized container, while the Pocket simulator contains the bare minimum amount of electronics to run the glider's software. These simulators run in real-time, so testing long term missions can be cumbersome, if not infeasible. Although the new speed model based simulator described is also capable of running faster-than-real-time, the simulations are coarser and do not include detailed platform information, for example, device driver behavior.

The port of the Slocum Glider software was motivated by the fact that much of the work while creating the new programming framework involved developing and integrating new functionalities, such as new services, into the vanilla software system. Having

modern tools, such as debuggers, available eases development and testing. Synthetic, as well as real data, from previous deployments can also be inject to test the vehicle's software. Thus, much of the interoperability can be accomplished on a modern desktop before ever testing it on an actual vehicle. Most importantly, however, was that a stable infrastructure to enable users to test and debug missions created using the programming framework was need. Authors writing ALGAE programs that are compiled for the Slocum Glider can be tested using this simulation environment and can be quickly provided with feedback.

Because the glider software is no longer tied to the glider's hardware and development stack, it can be easily extended to include additional features. In particular, one useful extension that has been added is the ability to run the simulator in a faster-than-real-time mode. Depending on the specifications of the host computer running the simulator, up to 30 mission hours in one minute have been simulated, a three order of magnitude (1800x) speed-up over a Pocket or Shoebox simulator. This enables long-term missions to be easily and quickly tested.

Furthermore, a hybrid mode that simulates faster-than-real-time while underwater, and real-time while at the surface has been implemented. In this mode of operation, a glider pilot can conveniently interact with the simulated glider while at the surface, for example to change mission parameters, while quickly simulating the underwater flight segment where no satellite communication is possible. This mode of operation is also used by the GPILOT tool when interacting with the simulator to perform state transitions.

An important aspect to any deployment is to monitor and estimate a vehicle's energy consumption. Like the glider, the simulator generates log files which can be used by the energy model after the simulated mission is completed. Additionally, the models have also integrated as a service into the simulator that will execute the energy models

during flight and present them as glider sensors to facilitate live energy evaluations. This enables energy aware services to be produced that can then be exposed to the higher level language.

Many advanced services that can be implemented will likely need to be executed on the AVBot single board computer. To enable these services to be used in the simulator an AVBot compatibility driver was created . When compiling the glider source, if the target platform is not the Persistor processor, then the compatibility driver will be compiled into the glider executable in place of the typical driver that interacts with the physical SBC hardware. Instead of using an RS-485 serial connection, as shown in Figure 4.1, the alternate driver uses a TCP/IP connection to speak the gliderbus protocol to the glider's runtime system. The runtime and its services can be execute on the AVBot SBC or another computing platform.

Like the model based simulator, the software port simulator also supports data inject either directly or via services. Ocean current, temperature and salinity data in NetCDF [Rew and Davis, 1990] format from the HYCOM prediction model can be loaded directly into the simulation. A bathymetry map can also be loaded and will use the vehicle's location to set the AUV's perception of the water depth. Services, however, can also inject data to create an virtual environment by producing output to sensor variables in the flight controller. Retrofiring the simulator does, nonetheless, offer additional flexibility on when and how often the artificial environment is updated in the control cycle.

To port the vehicles control software to work outside the Persistor processor many changes were made to the code base. The system software contains hardware specific code throughout the code base along with many hardware drivers. Compatibility or simulated drivers and functions were created as drop-in replacements to keep the system's source code as close to the original as possible. For example, the interface

to register and create periodic interrupts for the hardware and operating system is implemented with user level signals.

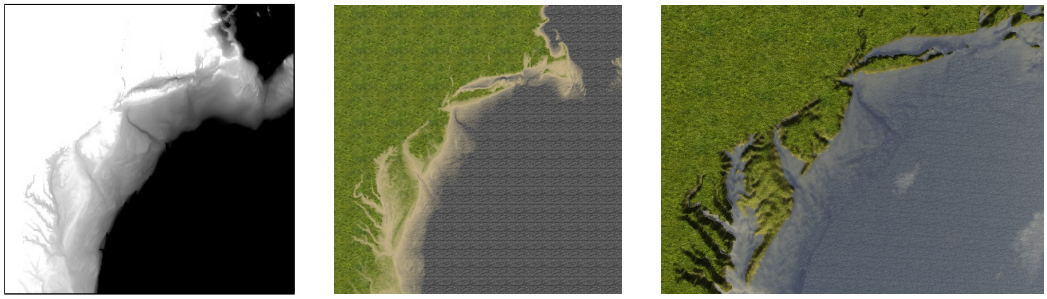
Having the capability to simulated a glider's flight using the same control software running on the actual vehicle is vital. It can provide detailed feedback to users when even the slightest changes are made to missions. It also increases the confidence of AUV engineers that the code generated for the vehicle can be trusted because it has gone through the rigors of the simulation process. Furthermore, it would also be useful as a general tool for planning future deployments and debugging ongoing mission.

6.3 Graphical Interface

Missions performed in the simulator can generate output about the vehicle as well as the environment. Typical focal points are on the glider's vertical profile (depth, pitch, roll) and geographic position. Much of these data can be graphed using tools such as matplotlib, Matlab, and the Generic Mapping Tools. We have nonetheless found the need to be able to visualize the Slocum Glider in three dimensional (3D) space for both simulated and real sea trials.

Google Earth is a powerful 3D tool that allows for the display of data on a world model [Google Inc.]. It is used extensively in the deployment process, from the initial planning stages to tracking the vehicle's flight. Weather forecast images are often overlaid to aid in the selection of new waypoints. The simulators presented have also been retrofitted to create KML files for Google Earth. Despite its usefulness, it also has its shortcomings. First, current licensing scheme limit how the content produced using the software can be distributed. Secondly, we would like to be able to easily customize the 3D environment to allow for live interaction as well integration with simulation environments.

To fulfill this gap, an interactive graphical environment for the Slocum Glider was



(a) (b) (c)

Figure 6.4: A heightmap (a), generated from NGDC's ETOPO1 model, is used to create a texture (b) for the 3D object (c) of the terrain in SimGUI.

created that is able to mesh with the existing framework. In the simulation front, it can be utilized in the development and debugging of algorithms, for example in glider swarming and coordination. With regard to actual deployments, it can be applied to the whole deployment process from waypoint selection, like Google Earth, to a 3D inspection of a vehicle's flight, such as that RU27's when it suffered from biofouling [Rutgers University]. This graphical user interface will be referred to as *SimGUI* although it is completely independent from the simulator.

The SimGUI makes use of Panda3D, an open sourced, BSD licensed, cross-platform 3D game engine with origins from Disney. It is a mature and stable engine used in commercial products. The core of the engine is written in C++, for efficiency, and its primary programming interface is exposed through Python. This eases integration with the simulators as many of their components are also exposed via the Python language.

The terrain information in SimGUI, as in the simulator, is based on a dataset by NGDC. A heightmap, an image representing the height of the terrain, must first be generated. This is accomplished by interpolating the height of a physical location and mapping it to a corresponding pixel in the image. A heightmap generated using this process of the coast of New Jersey is shown in Figure 6.4(a). The presented heightmap has been limited to maximum altitude of 25 m and maximum depth of 125 m. These

limits were chosen to give enough detail about the immediate landscape for orientation, and to provide enough depth for the simulations of a 100 m glider. Different parameters can be provided to the heightmap generation script, but details about the terrain may need to be sacrificed because most heightmaps are bound to eight bits of resolution.

Once the heightmap of the area of interest has been generated, a texture must be generated for the terrain. A set of images, such as sand, rocks and grass are repeated, or tiled until they have the dimensions of the desired image texture. Each of the tiled images are then blended together in a process called texture splatting. In this process, a pixel's transparency, or alpha, is calculated for each image depending on its altitude. For example, a pixel representing a deep ocean depth should have little or no contribution of the grass texture applied to it. Figure 6.4(b) shows the result of the texture splatting process performed using the NGDC dataset.

The heightmap is also used to generate a 3D object of the terrain. This is achieved in Panda3D using the *GeoMipTerrain* class. The terrain texture can then be applied to the 3D object. The resulting product, with lighting, is shown in Figure 6.4(c). Water, as well as a sky box surrounding the environment have been created and can be seen by the reflection of clouds in the lower right corner of Figure 6.4(c).

Depending on the graphical capabilities of the hosting machine, the level of detail (LOD) in SimGUI can be adjusted. The LOD of the terrain could be set to use a high number of polygons to shape the nearby scenery, while progressively decreasing the polygon count into the horizon as the distance from the engine's camera increases. Smaller textures for the terrain and the reduction of non-essential models can also aid in the programs performance but comes at the cost of the user's overall experience. If desired, SimGUI can optionally be loaded without a terrain object. This is certainly useful in non-coastal areas while simulating a shallow water glider because it is unlikely that the vehicle will ever reach the ocean floor.

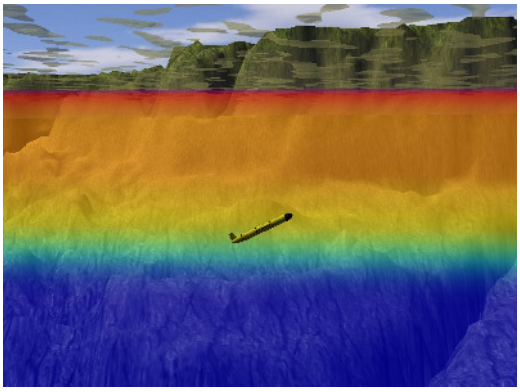


Figure 6.5: Glider replay of a thermocline tracking mission in SimGUI.

As previously stated, SimGUI is independent of the simulation infrastructure and intentionally so. This allows the simulation and the graphical environment to be developed separately and leaves open the possibilities for simulators from other vehicles to be integrated into SimGUI. Three dimensional objects must simply be imported and their respective simulators must provide the necessary information to SimGUI to place the vehicle in the scene.

The simulator and SimGUI typically run as separate operating system processes, but can be run as one. Running as one process gives the benefit of simplicity in that the memory is shared between the two components of the infrastructure. However, this can also lead to performance and engine frame rate issues because the process will be busy computing the simulation instead of refreshing the scene. This can be particularly be the case if multiple vehicles are in the environment concurrently.

When running as separate processes, the components can exchange information through pipes, shared memory or other forms of interprocess communication. Using UDP/TCP it is also possible for the simulation node to be on a completely different node than that of SimGUI. A cluster of computers can also run vehicle simulations while communicating to a central coordinating node. The coordinator ensures that other nodes have synchronized the global environment and updates the display state of

SimGUI to provide feedback of the distributed simulation.

The SimGUI and the simulation framework together are useful in other applications such as in the replay and analysis of missions and in the development and testing of algorithms. In particular it has been used to create a multi-vehicle coordination algorithm, to visualize the thermocline tracking mission as in Fig 6.5, and even as an education tool using a Wii Remote Controller.

Chapter 7

Applications

While developing the new programming framework a number of domain applications and challenges were explored to gain familiarity with the field. The results and lessons learned during the investigations have been integrated into the framework and have enabled it to mature and become a practical system.

In this section, I will describe the following applications where the programming framework was applied:

1. The feasibility of acoustic communications is explored on the Slocum Glider by having an AVBot service perform a simple action based on an acoustic signal.

2. Multi-vehicle collaborative sensing is investigated that would make use of an acoustic network. This work could lead to language features that allow a user to indicate which vehicle a glider should follow as part of a swarm.

3. An analysis of dead reckoning (DR) navigation and how a Doppler Velocity Log (DVL) can be used to improve an AUVs localization is examined. This comes at the expense of an increase in power. However, having a more precise notion of the vehicle's position may be worthwhile its cost while navigating in adverse weather conditions or when spatially tagging sensor readings is important.

4. The software ported simulator is used to investigate an issue with the heading algorithm implemented in the Slocum Glider. This heading algorithm is used

as part of the dead reckoning algorithm and can affect vehicle navigation. An alternative algorithm is implemented and compare to the existing algorithm.

5. Vehicle navigation by human pilots and by an automatic path planning system is explored. The comparison is a critical step to develop and gain confidence in an automatic path planning solution. Services in the framework could implemented various planning solutions with a variety of different goals and prioritize.

6. Services can be simple and run on the glider's processors, or can be rather complex and execute on more powerful devices like AVBot. An analyze on how a multi-core computing platform, that is flexible in its performance and energy requirements, could be used in future AUV sea trials is presented.

7. AUVs are deployed to perform sensing and have limited endurance. Thus, energy should not be wasted on collecting and sensing irrelevant data. An investigation of services that can be implemented as trigger chains to fire higher power sensors based on observations by lower power sensors is given.

Most of the technologies presented have not only been implemented in simulation but also at sea. By making these concepts available in the programming framework, for example by keywords in the domain specific language, these advanced features can become accessible to a larger audience. Users of the language can explore how their missions are affected by simple changes in the mission specification using the simulation infrastructures. Once satisfied, they can be translated to become real world missions.

7.1 Underwater Communication

As AUV platforms continue to mature, the demand for multi-node collaborative sensing will increase. A vital component needed to perform collaborative sensing is the ability for nodes in the sensor network to effectively communicate. To showcase the feasibility

(a)

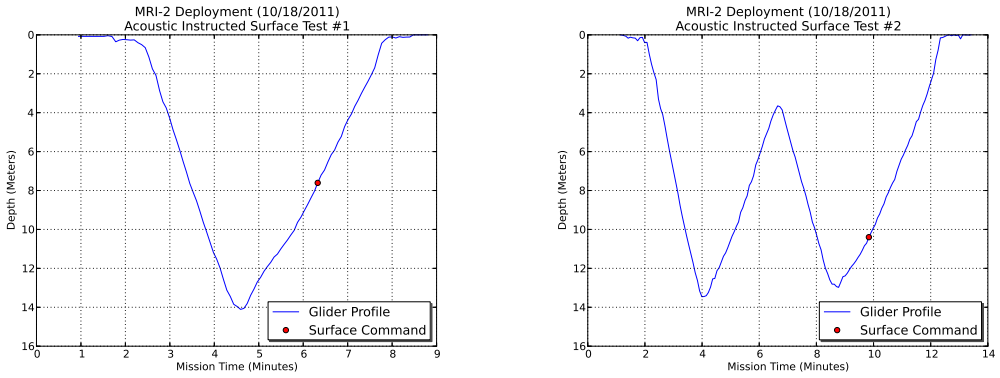
(b)

Figure 7.1: A Slocum Glider equipped with an acoustic modem (a) was tasked to surface upon a signal sent by a tow-fish (b) lowered from a surface vessel.

of multi-glider sensing the new infrastructure was used to induce a glider surfacing action caused by an acoustic signal sent from a surface vessel.

A Slocum Glider equipped with a Woods Hole Institute (WHOI) Micro-modem [Woods Hole Oceanographic Institution; Freitag et al., 2005] is depicted in Figure 7.1(a). The transducer of the modem is the blue cylinder in the center payload bay of the vehicle. A driver for the modem was written for AVBot and a service was implemented to send a signal to the glider's flight controller when a command is received by the modem. When this signal is received by the flight controller it activates a surfacing behavior that causes the vehicle to override other active behaviors and immediately surface. A tow-fish modem, shown in Figure 7.1(b), or another vehicle could be used to send such a command.

Using the glider and tow-fish in Figure 7.1, an acoustic communication sea trial was performed off the coast of New Jersey in October 2011. The glider was tasked to continuously sample the water column until it received a signal to surface. Two segments of the trial are shown in Figure 7.2. In both of the segments, the vehicle flew the same mission. Unfortunately, both signals were received by the vehicle while it was climbing, therefore it appears as though the glider would have surfaced anyway in the segments. However, based on Figure 7.2(b), the first segment would have indeed



(a) (b)

Figure 7.2: Two flights segments where the Slocum Glider was commanded to surface if it received an acoustic signal.

dove again if it did not receive the surfacing command. Log files on both the glider and AVBot confirm that the command was received and did indeed cause the surface behavior to activate and override the lower level behaviors in the layered control stack.

Although only two segments were presented, the sea trial does show that mutli-glider sensing is feasible using the infrastructure. Although a tow-fish modem instead of another glider was used, a communication link between two nodes was established and used to induce a glider action. Clearly, additional deployments are required to increase the robustness of the system. This is especially the case if another AUV were to be incorporated.

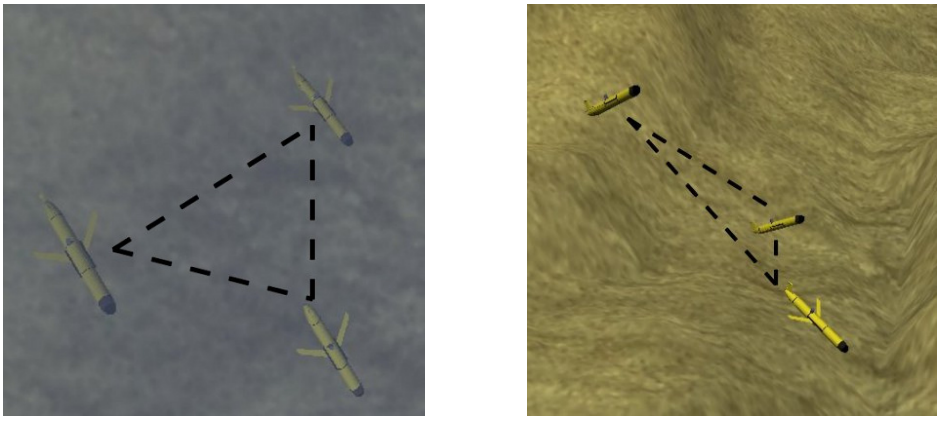
Instead of the AVBot service causing the vehicle to surface, one can imagine a service providing the flight controller with a heading to follow another vehicle to execute a swarming algorithm. This could also be incorporate into the domain specific language by specifying a *followglider* route action as part of the state specification. The hook behavior could then use the heading value to create and execute a *set_heading* sub-behavior.

7.2 Multi-Vehicle Coordination

Formation flight is a form of swarming where AUVs maintain particular positions relative to each other. It allows the observation of ocean conditions and phenomena at a high spatiotemporal resolution. In addition, formation flight enables AUVs with different sensor payloads to conceptually act as a single science instrument allowing enhanced sensing capabilities to be implemented by groups of lower cost, small AUVs instead of higher cost, large AUVs. This section investigates how such swarming algorithms may be implemented as it may become more common place in the future to see such multi-vehicle deployments. The study will also provide some insight and appreciation of the programming issues that may arise when programming such a swarm. The evaluation could contribute to design decisions for future revision of the domain specific language.

The prevailing approach to formation flight coordinates AUVs by using periodic surfacings to acquire and communicate new GPS positions to the command center, which calculates a new set of waypoints to recoordinate the vehicles [Paley and Leonard, 2008; Fiorelli et al., 2004]. Instead, a simple but effective coordination strategy for formation flight is proposed and evaluated which monitors the formation quality using low-bandwidth underwater communication. If an AUV drifts out of formation, using acoustic modem ranging techniques [Singh et al., 2006], all vehicles are instructed through underwater communication to resurface in order to reestablish the formation. Although this strategy may increase the frequency of surfacing, it has the advantage of: reducing the times needed at the surface to reestablish the formation since AUVs are not allowed to drift significantly out of formation, and acquiring more scientific data at the desired spatiotemporal resolution since larger portions of a mission are flown in formation.

The effectiveness of the proposed new strategy was evaluated by simulating multi-day missions involving three Slocum Gliders flying in a leader/follower formation using



(a) (b)

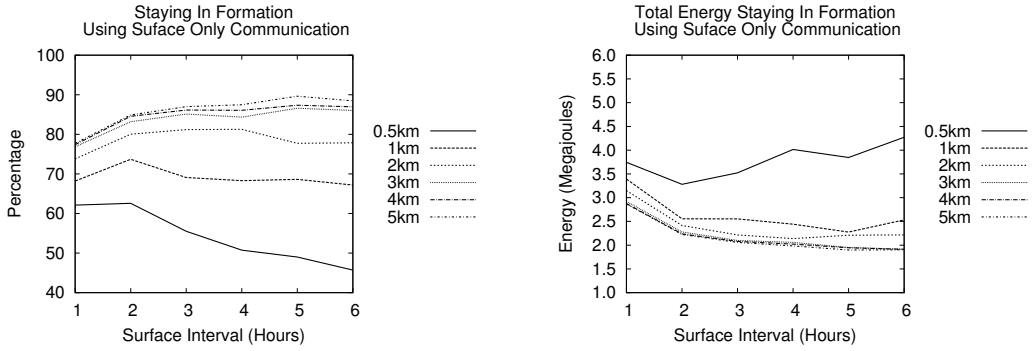
Figure 7.3: A top (a) and side (b) profile of a fleet of three Slocum Gliders coordinating to form a triangle formation using the simulation infrastructure and display through SimGUI.

the model based glider simulation infrastructure as shown in Figure 7.3 using SimGUI. The simulator uses the energy models obtained from the operation and communication of previous glider deployments off the coast of New Jersey as described in Chapter 5. The simulated mission is based on a deployment from June 2009 and includes satellite and radar information to model surface and underwater currents.

The simulations rely on two important parameters, namely the surface interval time and the formation distance. The surface interval describes at what interval of time, in hours, the gliders should resurface; the formation distance parameter specifies the size of the diameter the following gliders have to be in from the leader to be considered within the formation. If any of the gliders fall out of the circle, the formation is broken and is not counted as being part of useful scientific data.

Figure 7.6 and Figure 7.5 show the results of the simulations. In Figure 7.6 gliders could only communicate at the surface, while in Figure 7.5 gliders could communicate both at the surface and while underwater. Each data point is the average of 100 simulations performed on a 80 processor computing cluster.

The results of the simulations suggest that it is more difficult to maintain vehicle



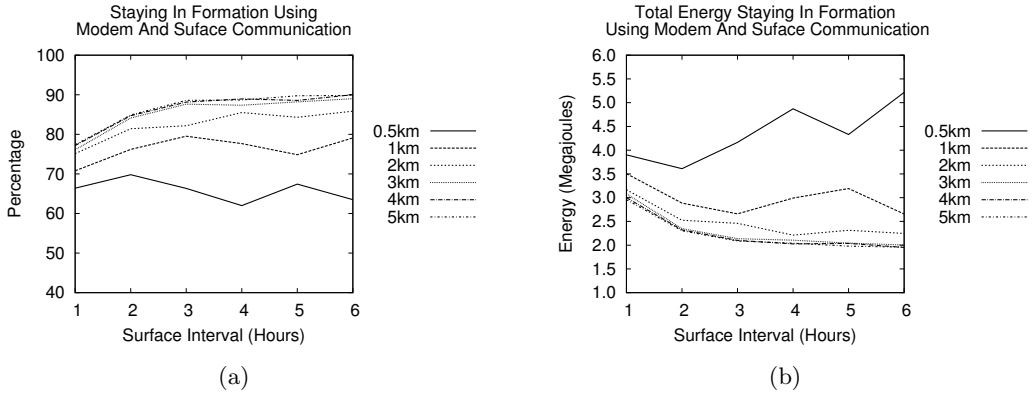
(a) (b)

Figure 7.4: Coordination strategy with surface communication only. (a) Quality of formation as percentage of mission time where formation was maintained; (b) Overall energy usage across all gliders in the swarm.

formation if the gliders are to sample more closely to one another. Increasing the time in between surfacing also has a negative impact on the coordinated sampling especially for short distances because the group may not realize they are out of formation and continue their flight path as if they were. By increasing the distance considered to be within the formation, the likelihood of the vehicles maintaining formation increases. However, by increasing the surfacing intervals, the vehicles are more likely to drift out of place and take more time to rendezvous when they do fall out of formation.

The surface interval also affects the total energy because it increases the number of times the vehicles surfaces which can be a relatively expensive operation. By decreasing the distance considered to be within formation the likelihood for vehicles to be out of formation increases. When a glider is determined to be out of formation, it causes other gliders in the simulations to remain at the surface while they wait to recoordinate. However, a shorter formation distance also means that the stray gliders will catch up more quickly (currents permitting).

Overall, the new strategy is able to keep a glider formation longer, but at the cost of an additional energy overhead. For small formation granularities, the trade-off is clearly in favor of the new strategy, in the best case increasing the flight time in



(a) (b)

Figure 7.5: Coordination strategy with surface and underwater communication (a) Quality of formation as percentage of mission time where formation was maintained; (b) Overall energy usage across all gliders in the swarm.

formation by 49% while only requiring 13% more energy. This corresponds to 25% decrease in energy cost per scientific data sample in the formation compared to surface only communication. For larger formations this is no longer the case since maintaining the formation and the corresponding energy budget are comparable across the two strategies.

7.3 Improving Dead Reckoning Using a Doppler Velocity Log

An AUV's ability to predict its location is important for a number of crucial tasks [Whitcomb et al., 1999]. Path planning algorithms, for example, need to be able to track the vehicle's location so that it can surface as close to the target waypoint as possible. Additionally, some sensors may need to be tagged with high spatial accuracy. Most AUVs currently use a dead reckoning (DR) algorithm to predict their location. In the case of the Slocum Glider, it is computed from measurements of pitch, heading, and depth change. Though easy to implement, this form of dead reckoning can produce inaccurate estimates, especially for long dive segments. The lack of water current measurements, imprecise sensor readings, and other effects can influence an AUV's

flight path and inaccurately cause the glider to drift away from its true location during missions [Leonard et al., 1998].

More accurate DR localization strategies for the glider could take into account Acoustic Doppler Current Profiler or Doppler Velocity Log sensor data [Teledyne RD Instruments]. A DVL, for example, is able track the bottom of the ocean floor, allowing it to calculate the relative motion of the vehicle to the floor[Whitcomb et al., 1999; Fong and Jones, 2006]. The sensor's reported speeds could then be replaced by the traditionally calculated speeds during the DR process. This section will explore such a strategy for the Slocum Glider and demonstrate how it can dramatically improve the vehicle's localization estimates. This approach evaluated by comparing a glider's flight segments from a 12 day deployment off the coast of New Jersey both with and without DVL assisted dead reckoning (DVLDR).

7.3.1 Background

Drivers written for scientific sensors for the glider's science bay computer are known as proglets. Proglets, like the DVL, can require knowledge of other sensor values to perform their measurements. This information may be ascertained from other sensors connected to the science processor or from the flight controller. A serial connection (RS-232) between the two processors provides the necessary hardware infrastructure for data transmission. A software protocol known as the superscience protocol controls how the processors interact with one another to perform the actual sensor data exchange. For example, the computers can request from one another a sensor value to be sent only once, when changed, or when touched (timestamp update on the sensor value).

The DVL proglet on the science process requires five such sensors from the flight controller to be sent upon every change: current water depth, vehicle depth, pitch, roll, and heading. The most current view that the science computer has of these values is

sent to the instrument upon each measurement request to update the DVL's view of the environment. Since the sensor requires these data to perform accurate readings, it is critical that they be as up-to-date as possible. Delays in the transmission of the sensor data could effect the bottom tracking reported by the sensor which propagates to the DVLDR strategy.

The traditional dead reckoning algorithm on the Slocum Glider is quite simple and calculates the vehicle's estimated position at every four second control cycle. The algorithm requires input from two onboard sensors, namely the pressure and attitude sensors. The pressure sensor allows the vehicle to determine its depth (d) in the water, while the attitude sensor measures the vehicle's pitch (θ), roll and heading (h). The following describes the basis of the algorithm as it calculates its new location in the local mission coordinates (LMC) system:

$$ws = \frac{-\Delta d}{\tan\theta} \tag{7.1}$$

$$\begin{aligned} vx &= (ws * \sin h) * wvx \\ vy &= (ws * \cos h) * wvy \end{aligned} \tag{7.2}$$

$$\begin{aligned} \Delta x &= vx * \Delta t \\ \Delta y &= vy * \Delta t \end{aligned} \tag{7.3}$$

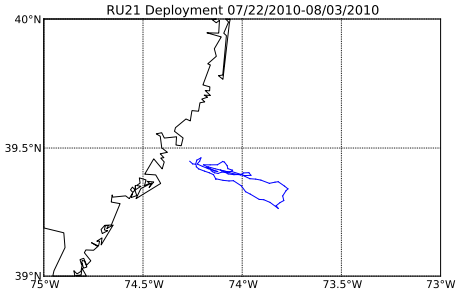
$$\begin{aligned} lmcx &= lmcx + \Delta x \\ lmcy &= lmcy + \Delta y \end{aligned} \tag{7.4}$$

$$dist = \sqrt{\Delta x^2 + \Delta y^2} \tag{7.5}$$

The LMC system is the internal navigation system used by the Slocum Glider. It describes the distance in meters the vehicle has moved north and east since the start of the current mission. Equation (7.1) determines the vehicle's speed through the water using the current pitch θ and the change in depth Δd since the last control cycle.

Next, in Equation (7.2), the glider's velocities are determined using the current heading (h) and an optional water velocity component for current correction. Vx denotes the eastward velocity while vy denotes the northward velocity. These components are converted to meters in Equation (7.3) by multiplying them with the time since the last control cycle. The new DR position is then determined in Equation (7.4) by updating the last calculated location with newly made LMC progress during the current cycle. The final horizontal distance covered is determined in Equation (7.5).

For short dive segments the described technique works quite well. However, for longer dive segments, or segments where the currents are strong, errors in the estimation can accumulate over time. For many applications, a highly accurate DR position may not be required and the existing approach is still valid. Some scenarios like path planning and navigation through shipping lanes require more accurate DR predictions. The data produced by a DVL performing bottom tracking could be integrated into the existing DR algorithm to assist the vehicle in underwater navigation.



(a)

(b)

Figure 7.6: The flight path of a glider deployment (RU21) from July 22, 2010 to August 3, 2010, off the coast of New Jersey (a). The vehicle was equipped with a DVL hat performed bottom tracking throughout the mission. The glider being recovered from the deployment (b). The DVL is part of the aft sensor payload bay in the center of the vehicle. The wings of the glider were removed during recovery to allow the glider to be pulled onto the boat without damaging the sensors.

7.3.2 Evaluation

A glider's DR error can be influenced by a variety of effects; a DVL, however, may provide the necessary sensory input to greatly reduce this error. To gain a quantitative perspective as to how much a DVL may assist in the DR of a glider, DVLDR is evaluated with a previously flown deployment equipped with a DVL. The original flight was flown without DVLDR, however this section will detail how much closer a DVLDR implementation could come to the actual surface location of the vehicle for each dive segment.

Deployment

The deployment used as the basis for the evaluation of the DVLDR took place off the coast of New Jersey from late July 2010 to early August 2010, lasting 12 days. The overall objective of the mission was to test a new glider equipped with a DVL as well as other new sensors. The flight path taken by the vehicle over the short mission is shown in Figure 7.6(a), and a picture of the recovery of the vehicle is shown in Figure 7.6(b). The dual science payload bays in the middle of the hull contain the glider's scientific sensors; the DVL was carried in the aft science bay for the deployment.

Since the exact performance of the newly tested sensors under field conditions were not yet known, the vehicle was commanded to remain relatively close to shore to ensure a quick recovery if required. The water depth during all dive segments were within 35 m, which was well within the maximal (theoretical) 60 m bottom tracking range of the DVL. Dive segments during the deployment were also kept quite short at an average of two hours with about 20 min for data transmission to shore while at the surface. On average, when a glider surfaced and gained a GPS fix, it was approximately 0.5 km off its estimated DR surface location. The minimum and maximum distances from the DR position were 23 m and 1.4 km respectively. Thus, although not detrimental, the

dive segments were rather short so room for improvement to reduce the error of the DR position exists.

As previously described, the proglets on the science computer communicate with sensors. Their data can then be logged locally on the science computer, or sent to the glider flight controller to be recorded. Traditionally, before the 7.0 series release of the Slocum Glider software, only logging on the flight Persistor was possible. Although the vehicle for the said mission was running a release in the 7.0 series of the codebase, no science data logging occurred. This was intentional, as the feature was still maturing and the functionality of other components, like the DVL, were the focus of this mission.

The decision to not perform data logging on the science computer also had its faults as it not only impacted the DVL data being logged but also the measurements themselves. The serial connection between the science and glider Persistors is slow, running at 9,600 baud rate on releases before the 7.0 series and 4,800 baud rate since the 7.0 series release. Science data logging reduces the traffic on the serial connection and is likely the reason why the baud rate was lowered. However, sensors like the DVL send a large amount of data which increases the load on the communications system.

Five sensors are needed by the DVL to perform its measurements. Not only are the five input values regularly sent across the serial link, the results produced by the DVL are also sent back. Currently, during bottom tracking, the DVL can produce up to 46 output sensor values. Although not all values must be sent, since they may not have been updated, there is still contention on the serial connection since it is shared by other proglets including the CTD sensor.

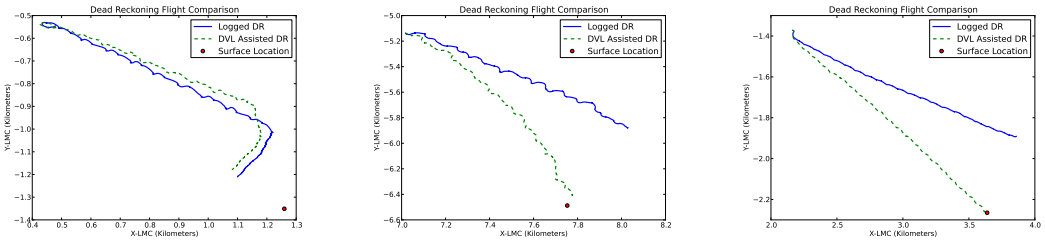
From a science Persistor point of view, the snapshot of the vehicle's physical orientation it receives from the glider may be inconsistent. For example, when updating the DVL's viewpoint of the environment to prepare the sensor for a measurement, a new pitch value may have been received by the proglet. However, an older depth value from

a previous glider cycle will be used if it has not been updated in time. In this scenario, the DVL may perform a measurement while diving that lags behind by several meters in depth which could affect the results. The opposite also holds true. The readings logged by the flight controller may be a conglomerate of both new and stale values that are several cycles old. Thus, although the data used in the evaluation is not always the most up-to-date, it can still prove useful in improving DR. Future deployments that enable science data logging should show even more promise since there should be less contention on the serial link.

Methodology

To evaluate how a DVL can assist in dead reckoning, a metric to compare the two DR methods must be established. Ideally, an underwater localization algorithm should be without error and be capable of calculating the true position of the vehicle at any time. However, the glider's DR algorithm, for example, has sensory input and computation limitations which can cause errors to accumulate over time. This becomes apparent when the vehicle surfaces after a dive segment, as the DR location usually differs from newly acquired GPS position. The distance between the DR and the GPS locations is used as the metric for the evaluation. The closer that the DR comes to the GPS fixed surface location, the better the algorithm was able to estimate the vehicle's position.

During the deployment, the glider only performed the standard DR algorithm so DVLDR must be simulated. To ensure that the results produced by a simulated DVLDR can be trusted, and the retrofitted glider software would produce similar results, DVLDR should be based on a similar algorithm to the one in the glider. The glider's DR algorithm has been ported to run independently from the vehicle and has been adapted to make use of data recorded during previous deployments. When running the ported DR algorithm with the dive segments of the sample mission, the difference



(a) (b) (c)

Figure 7.7: Comparison of flight paths of logged dead reckoning flights against DVL assisted dead reckoning (DVLDR) flights. In general, DVLDR significantly improves estimated vehicle position as in (b) and (c), while at times the traditional DR estimates are more accurate (a).

between the ported and vehicle's logged DR position were negligible, usually within one meter. This is due to the difference in the glider's state information used for the DR calculation. While the true vehicle's state is evolving during a control cycle, the ported algorithm instead uses a state snapshot that is recorded for each cycle.

To simulate DVLDR, the ported algorithm is altered to take into account the DVL. When valid data is produced from the sensor, the northward and eastward velocities are used in place of the calculated velocities of Equation (7.3) in Section 7.3.1. When the sensor is not able to perform bottom tracking, or its data is determined stale, the DVLDR will fall back to the traditional DR strategy. The sensor may not be able to perform bottom tracking, for example, if the water depth is too deep or the DVL cannot reliably detect the ocean floor. Using a combination of both strategies produces significantly better results compared to the standard approach.

Results and Discussion

By replaying the mission with recorded data from the deployment, and using the velocities gathered from the DVL, DVLDR is able to significantly reduce the DR error compared to the traditional approach in most cases. Three sample segments from over 130 segments of the deployment are shown in Figure 7.7. The figure compares algorithm

performance from the worst case scenario to the best.

Figure 7.7(a) presents the worst case of the ten segments where DVLDR did not outperform the standard DR algorithm. The logged DR position placed the vehicle 212 m from the GPS fixed position, while DVLDR estimated 246 m, an additional error of 34 m. The standard deviation of the ten segments was less than 13 m, which in the overall scheme of the deployment including the successful segments is not significant.

The flights paths of Figure 7.7(b) and Figure 7.7(c) showcase an average and one of the best segments that was improved. The errors, such as in Figure 7.7(b), are likely caused by the vehicle losing bottom tracking and the algorithm falling back to the traditional DR method, or possibly by errors in the measurements themselves. As stated this may have been caused by the delay in sensory input either to the DVL or from the DVL to the flight controller.

For all segments of the mission, DVLDR reduced the average surface location DR error from over half a kilometer to under a quarter of a kilometer. The minimum and maximum error were 6.5 m and 1 km respectively, compared to the 23 m and 1.4 km by the traditional approach. Overall, usage of a DVL during DR has a great impact on reducing the error of the glider's localization strategy. Using this new algorithm during deployments could improve overall vehicle navigation. Having a better sense of the vehicle's location translates into being able to calculate more accurate heading corrections towards the target waypoints. Sensors that require more accurate tagging would also be benefited from this method. If missions or applications can cope with the additional energy required by the DVL sensor, then the use of DVLDR is worthwhile.

In terms of the glider, the DVL is a relatively energy expensive sensor. Algorithms or services in the programming framework should schedule and manage its usage to get the most utility out of the sensor. For example, a service may determine that after a dive segment without the DVL powered that a significant drift is detected. The DVL

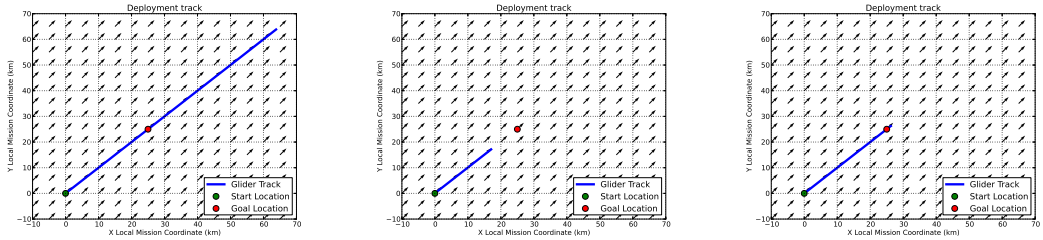
should then be used for some time during the next profile. The opposite could also be true; if the currents do not appear to affect the vehicle as much, an algorithm may choose not to use the DVL. Shore based services that use weather prediction models could also play a part in the decision making and could relay new instructions to the glider while it is at the surface. The power measurement infrastructure or energy service can quantify the energy usage and provide feedback to other services so that they may make any necessary adjustments.

7.4 Current Correction System

During typical glider operations, the AUV is typically tasked to fly to a list of waypoints. Pilots may choose to navigate with the feature of current correction (CC) enabled or disabled. When enabled, the vehicle's software will use its estimates of the water current components in its dead reckoning (DR) and heading calculations. When disabled, the water current is not considered in either the DR or heading calculation.

Based on the feedback from several experienced glider pilots, many missions are flown, in part, without the use of CC. These pilots expressed that, at times, they had difficulty navigating the vehicle while the CC feature was enabled and therefore fly without it in some situations. It would, however, be useful to navigate with CC to ensure that the vehicle follows the track if, for example, the vehicle makes use of a path planning system.

This section investigates the CC algorithm implemented in the Slocum using the software port simulator flying in the faster-than-real-time mode. Two nearly identical mission files were created to fly the vehicle from a starting location to a north-east target waypoint approximately 35 km away. The mission files only differ in that one has CC enabled while the other does not. A favorable current in the direction towards the target waypoint was set in the simulator with a speed of 50 cm/s, which is larger



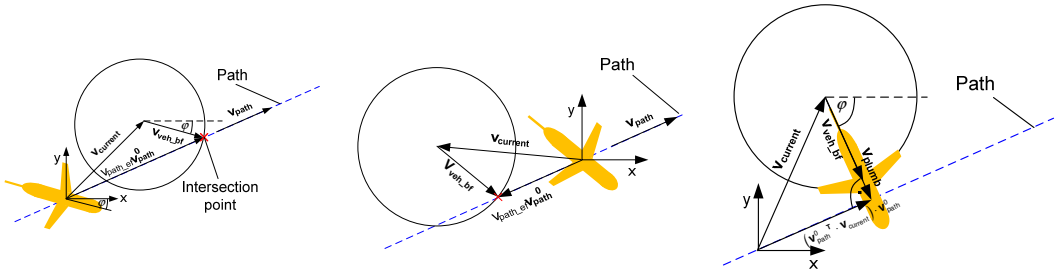
(a) Current correction disabled. (b) Default current correction algorithm. (c) Current correction enabled with the new heading correction algorithm.

Figure 7.8: The path taken by three flights flying a sample mission tasked to fly to a waypoint 35 km north-west with a current of 50 cm/s.

than the average speed of the glider. Three missions were flown: with CC disabled; with CC enabled; and with CC enabled but with the alternative heading algorithm described later.

The flight tracks of the simulated Slocum Glider's missions are shown in Figure 7.8. These tracks are of the actual paths of the vehicles and not their DR paths. In Figure 7.8(a), with current correction disabled, the glider vastly overshoots the target waypoint. When disabling CC, the glider assumes in its DR calculations that the water current components are both zero. Therefore, in strong currents, the DR position of the glider can grow to be significantly different than that of the AUV's actual position. In the sample mission, the vehicle will continue to fly and only surface to complete the mission when it believes it has arrived at the waypoint. Thus, if the AUV's DR positioning is highly inaccurate it cannot effectively navigate itself to the target.

Figure 7.8(b) shows the flight path with the standard CC system enabled. The heading algorithm is executed periodically to adjust the glider's flight path towards the target waypoint. The algorithm first calculates the expected flight time to reach the waypoint using the AUV's average speed with no consideration of the sea current. Then, using this flight time, the algorithm offsets the target waypoint by the displacement caused by the current during that time. The heading to the new offset waypoint is



(a) Definition of the velocities (b) Negative v_{path_ef} (c) Negative discriminant

Figure 7.9: Velocity relationships.

calculated and used to fly the glider. As seen in Figure 7.8(b), the glider falls short of reaching its intended target. Because the current speed is greater than the vehicle's average speed, the heading algorithm actually caused the AUV to fly directly straight into the current, i.e., in the opposite direction away from the target. This simple scenario highlights the problem with standard CC system.

We implement an alternative heading algorithm to adjust the glider's course in the simulator. The path taken by the vehicle with the modified CC system is shown in Figure 7.8(c). The vehicle maintained a direct heading towards the target and arrived at the waypoint in the shortest time of the three flight scenarios.

As was demonstrated, the existing heading algorithm incorrectly guides the AUV to fly directly against a favorable current. The modified algorithm implemented in the simulator is also used for the automated path planning flights in Section 7.6.3. Though the path planning system described offers the ability to generate waypoints offset by the water current conditions, a more general solution was explored for future inclusion into the glider's software. The algorithm described was developed in collaboration with Dr. Eichhorn and is heavily based on his previous work in path planning [Eichhorn et al., 2010; Eichhorn, 2009].

The glider heading φ to follow a defined path can be calculated to include the

ocean current vector $\mathbf{v}_{current}$ and the path/course vector $\mathbf{v}_{path}$. This path vector can be described by a magnitude and a direction. The direction is defined by a unit vector $\mathbf{v}^0_{path}$ of a point subtraction of the target waypoint and the current vehicle position. The magnitude of the path vector is the speed which the glider travels on the path in relation to a fixed world coordinate system. This speed v_{path_ef} depends on the vehicle speed through the water v_{veh_bf} (cruising speed), the magnitude, and the direction of the ocean current vector, as well as the direction of the path $\mathbf{v}^0_{path}$. This speed can be determined by the intersection point between a line and a circle (2D) and/or sphere (3D) [Schneider and Eberly, 2003], based on Figure 7.9(a), according to the following relation (7.6):

$$\begin{aligned} &line\text{: } \mathbf{x}\left(v_{path\_ef}\right) = v_{path\_ef}\mathbf{v}^0_{path} \\ &circle/spheres\text{: } v^2_{veh\_bf} = \|\mathbf{x} - \mathbf{v}_{current}\|^2 \end{aligned} \tag{7.6}$$

$$\begin{aligned} disc = &(\mathbf{v}^0_{path}{}^{\mathbf{T}} \cdot \mathbf{v}_{current})^2 \\ &+ v^2_{veh\_bf} - \mathbf{v}_{current}{}^{\mathbf{T}} \cdot \mathbf{v}_{current} \end{aligned} \tag{7.7}$$

If the discriminant $disc$ in Equation (7.7) is positive, the glider heading φ can be calculated using the following equations (7.8):

$$\begin{aligned} &\text{if } disc > 0 \\ &v_{path\_ef} = \mathbf{v}^0_{path}{}^{\mathbf{T}} \cdot \mathbf{v}_{current} + \sqrt{disc} \\ &\mathbf{v}_{veh\_bf} = v_{path\_ef}\mathbf{v}^0_{path} - \mathbf{v}_{current} \\ &\mathbf{v}_{veh\_bf} = \begin{bmatrix} x_{v\_veh\_bf} \\ \\ y_{v\_veh\_bf} \end{bmatrix} \\ &\varphi = \text{atan2}(y_{v\_veh\_bf}, x_{v\_veh\_bf}) \end{aligned} \tag{7.8}$$

If the speed v_{path_ef} is negative, the vehicle is still on the path, however, it is moving backwards. This scenario is shown in Figure 7.9(b).

If the discriminant $disc$ in Equation (7.7) becomes negative, v_{path_ef} does not have a real solution. This means that the vehicle cannot be held in that path and so the path is not feasible. This scenario is depicted in Figure 7.9(c). In this case, the calculated heading results in a "closest point on the line" calculation. The resulting glider heading is perpendicular to the path so that the drift to the desired path is minimal. This can be calculated using the following equations (7.9):

$$
\begin{aligned}
&\text{if } disc \leq 0 \\
&\mathbf{v}_{plumb} = \left({\mathbf{v}^0_{path}}^{\mathbf{T}} \cdot \mathbf{v}_{current} \right) \cdot \mathbf{v}^0_{path} - \mathbf{v}_{current} \\
&\mathbf{v}_{plumb} = \begin{bmatrix} x_{v\_plumb} \\ \\ y_{v\_plumb} \end{bmatrix} \\
&\varphi = \text{atan2}(y_{v\_plumb}, x_{v\_plumb})
\end{aligned}
\tag{7.9}
$$

Using the software port simulator from the programming framework an existing issue with the Slocum Glider's heading algorithm was easily verified through data injection and testing. An alternate heading algorithm was implemented and tested with initial simulation results indicating that it is a worthwhile candidate to replace the existing algorithm. However, several field trials and feedback from flight engineers would be required before it is integrated into the general glider codebase.

7.5 Assessing Automated and Human Path Planning

Most AUVs are operated by pilots who are able to interpret environmental information to make effective mission decisions. However, managing a group of glider can become rather complex even if a single AUV is under duress. To enable oceanographers and pilots to more easily manage a fleet of gliders, new mechanisms are needed to ease the

burden of AUV operation. An automated path planning system is one such tool that could free operators from the tedious task of waypoint selection, and would allow them to focus on scientific and mission critical aspects of managing groups of AUVs.

AUV path planning involves selecting a set of waypoints to guide an AUV from a starting location to a destination location while considering obstacles such as shipping lanes, ocean currents, or limited battery resources. Offloading operational tasks to an automatic tool is only feasible if the decisions made by the tool are considered reasonable and can be trusted. To assess the flight paths and energy consumptions of an AUV guided by an automated path planning system and by human pilots, a testbed environment has been developed. The testbed is based on the software port glider flight simulator. Four pilots with varying backgrounds and glider flight experiences were asked to fly a Slocum Glider through a simulated Gulf Stream modeled current field. The same challenge was posted to an automatic path planning system currently in development.

In the scope of the overall programming infrastructure, the automated path planning system developed for this assessment is intended to provide a set of guided route service. The service in this case would be dependent on shore-side processing to calculate the intermediate waypoints to be flown. Various path planning services may have different priorities and thus produce different routes, for example, to navigate a convex hull of an area of interest. However, the technologies involved to perform such services must first be stabilized and trusted before they can be used in real sea trials.

7.5.1 Path Planning

Path planning is an important requirement for an autonomous mobile system, and is necessary to effectively navigate a vehicle during a mission. All current and future information about the area of operation and the vehicle's status are used to formulate

a path. In the case of AUVs, information about the area of operation can be gathered from ocean models and from measurements derived by the AUV itself. Other important vehicle properties, such as its speed and energy consumption, are also important components to consider to effectively plan a course [Rao and Williams, 2009].

The goal of the path planning algorithm used in this assessment is to find a time-optimal path from a start position to a goal position by evading all static and dynamic obstacles in the area of operation, while considering the dynamic behavior of the vehicle and the time-varying ocean current. This path planning algorithm, named the Time Variant Environment (TVE) algorithm [Eichhorn et al., 2010; Eichhorn, 2009], is based on a modified Dijkstra algorithm [Dijkstra, 1959]. A time-variant cost function is included in this algorithm, which will be calculated during the search to determine the travel times (cost values) for the examined edges. This modification allows a time-optimal path to be determined in a time-varying environment. In [Orda and Rom, 1990], this principle was used to find the optimal link combination to send a message via a computer communication network with the shortest transport delay.

The path algorithm uses a geometric graph for the description of the area of operation with all its characteristics. The defined points (vertices) within the operational area are those passable by the vehicle. The passable connections between these points are recorded as edges in the graph. Every edge has a rating (cost, weight) which is the time required for traversing the connection. In the case of an ocean current, the mesh structure of the geometric graph will be a determining factor associated with its special change in gradient. In other words, the defined mesh structure should describe the trend of the ocean current flow in the operation area as specifically as possible. A uniform rectangular grid structure is the easiest way to define such a mesh.

7.5.2 Evaluation

The aim of the presented evaluation results are not to criticize a particular path planning strategy, whether it is human piloted or piloted by a path planning algorithm. Rather, the aim is to provide the capability and technology to evaluate these strategies. In the case of human piloted flights, a graphical interface is provided to pilots that interacts with the software port simulator. For the automated flight, a tool interacts with both the simulator and the path planning program. In both cases, the simulator was modified for the evaluations as described in the following sections.

Simulator Modifications

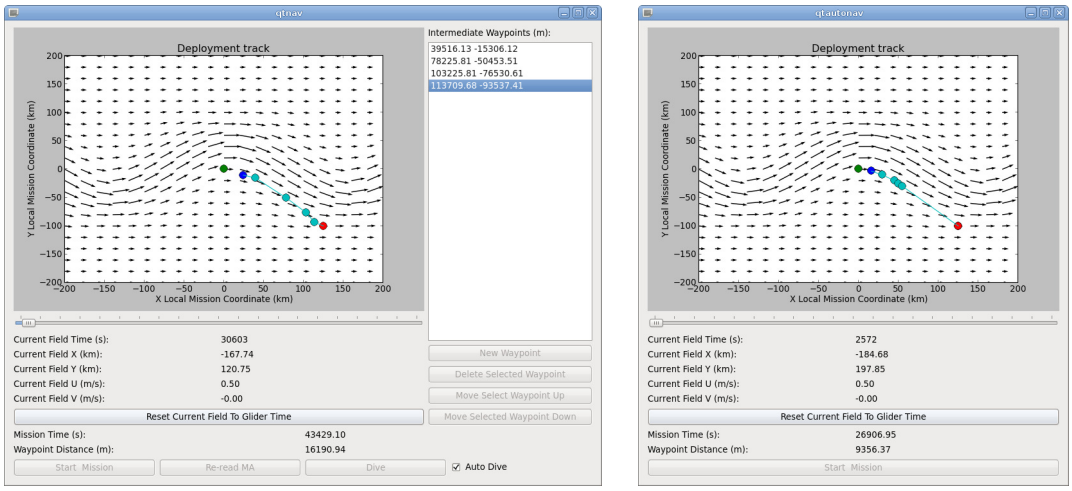
For the evaluation, the new heading algorithm discussed in Section 7.4 was integrated into the simulator. Current correction was not used for the human piloted missions as some of the more experienced pilots preferred to fly without it for the current model used in the evaluation. The algorithm was, however, enabled for the automated flight. Although the path planning tool has been augmented to provided waypoints with the correct current offsets, a more generalized solution was used for the vehicle.

One advantage of running a full software-stack simulator is that specific hardware and software parameters can be easily adjusted to better reflect the flight characteristics of a particular glider. For example, the vehicle's simulation driver determines the speed of the glider through the water using the AUV's pitch and change in depth. The pitch and depth rate are in turn functions of the vehicle's pitch battery and buoyancy pump positions. In a stock simulator, the models that map motor positions to the pitch and depth rate are based on a flight from Buzzards Bay in 2002. Because the simulator is often used to study past and live deployments, the pitch and depth rate models have been modified and are modeled after a two week flight of Rutger's RU06 glider off the coast of New Jersey.

The Slocum Glider model from Buzzards Bay (BB), while valid, is not a generalized model. Each particular glider can fly very differently depending on, for example, how it was ballasted. According to the log files, the RU06 appears to be too positively buoyant; the vehicle spent over 70% of its time in the diving state rather than an even time in the diving and the climbing states.

Using linear regression in the Weka data mining software, [Hall et al., 2009], two models were created for the vehicle. First, a battery and buoyancy pump position to pitch model, and second, a pitch and buoyancy pump position to depth rate model. The predicted pitch and depth rate of the models were compared to the vehicle's log files. The average predicted error for the new pitch model when compared against the vehicle's log files was 3.5° compared to 10.2° of the BB model. Most of the errors in the new model occur during inflections, while in the BB model, the errors lie in the misprediction of the climb angle. The error of the depth rate models was 6 cm/s for the new model compared to 18 cm/s for the BB model. The high error in the BB model was likely due to RU06 being too positively buoyant, and the diving depth rate being much lower than that of the glider in Buzzards Bay. Nonetheless, this does not invalidate the BB flight model as a model to be used for the simulator, but it does showcase the importance of tuning such models to more closely reflect the nuances between particular gliders. These RU06 models are used throughout the evaluation although the BB model could have been used instead.

The stock software allows a fictitious sea current to be specified. However, this current is static until it is explicitly updated by the user. In Section 7.4, this static current specification was used to study the stock glider's heading algorithm. To increase the realism of the experiments, the simulator was modified to dynamically change the currents in an effort to reflect a 3D Gulf Stream current model [Cencini et al., 1999; Alvarez et al., 2004; Eichhorn et al., 2010]. Other model data could have been injected,



(a) (b)

Figure 7.10: (a) The graphical user interface used by a human pilot to help navigate a simulated glider from the green starting waypoint to the red destination waypoint. Intermediate waypoints selected by the pilot are drawn using cyan indicators and lines. The automated tool, (b), uses path planning software to determine intermediate waypoints.

but this model was already integrated as one of the testing environments in the path planning tool.

Piloting Tools

To ease the evaluation on the human test subjects, the simulator in hybrid mode was used. In this mode, simulations of the glider's flight are faster-than-real-time while underwater and real-time at the surface. Typically, pilots interact with the glider using the manufacturer provided Dockserver mission control system. However, a good user experience has yet to be obtained with the Dockserver while the simulator is running in any faster-than-real-time mode. Alternatively, interaction with a simulated glider's terminal is accomplished with the Pexpect [Spurrier, 2012] Python module, commonly used to control and automate programs. Because Pexpect launches the simulator as a subprocess, it is still necessary to slow the simulator down while at the surface. If it is not slowed, the monitoring program will not be able to communicate with the AUV in

time before the vehicle continues onto the next mission segment. The tool is in essence a modified GPILOT tool specifically adapted for this evaluation.

In lieu of the pilots taking over control of the glider at the surface, a graphical user interface has been created to send and receive vehicle events and messages to and from the terminal monitoring program. This GUI tool is shown in Figure 7.10(a). As described, the automated path planning tool also uses the Gulf Stream model in its calculations. Thus, the human pilot is provided with a graphical depiction of the surface currents that will occur within the next 20 days. It is common for pilots to overlay ocean current model data in applications such as Google Earth to assist them in their waypoint generation. Although the presented prediction model is extended to 20 days, thereby creating an atypical advantage compared to a real deployment, it is justifiable considering that the path planning tool knows about the model and the pilots were unfamiliar with the area of operation.

In the interface, pilots can use the slider beneath the current plot to see the surface currents at any particular time. Note that the Gulf Stream model is a 3D model and it applies currents to the simulated glider at all depths. Hovering the mouse over the plot updates the GUI with the water current information at that location and at the time specified by the slider. The vehicle's mission time and distance to the next waypoint is also shown which provides additional feedback to the user about the deployment. If the pilot wishes to plot the surface currents that are being applied to the glider, the reset button can re-adjust the current field slider to the glider's current mission time.

New waypoints can be added and deleted using the tools on the right-hand side of the interface and by selecting current field plot. In Figure 7.10(a), the start location is indicated by the green indicator, the destination by a red indicator, and intermediate waypoints, specified by the user, as cyan indicators and lines. Any modifications to the waypoints list requires the user to explicitly click the "Re-read MA" button to have the

program generate a new mission argument list, send it to the AUV, and have it re-read and update the mission's behaviors.

The mission arguments for the waypoint list sent to the glider are in the glider's local mission coordinates (LMC) system. This is unconventional, and the simulator was extended to support this feature. Typically, only waypoints specified via latitude and longitude are supported in the goto list behavior's mission argument file. This was done purely for convenience so that the Gulf Stream model and the path planning system can both use the same coordinate system.

Users of the tool can only send new waypoint lists when the simulated glider is at the surface. To instruct the glider to continue onto the next dive segment, each lasting four hours, the pilot can click the "Dive" button. If no changes are expected for several dive segments, the "Auto Dive" checkbox will have the GUI tool instruct the glider to continue its mission on behalf of the pilot. If a modification to the mission is required, unchecking this box will allow the pilot to manually re-task the AUV at the next surfacing.

The interface of the automated path planning tool is shown in Figure 7.10(b). The interface presented is very similar to that of the human piloting tool. Although having a GUI is not required, it aids in debugging and development to ensure that the expected waypoints are generated and flown. There is also a "headless" version of the automated path planning system that does not use a GUI. The headless version is helpful in parameter space exploration and has been deployed on a compute cluster where multiple path planning algorithms can execute at the same time.

The automated flight program also uses the monitor program in place of Dockserver to control the glider's terminal. During the simulated mission, when the AUV comes to the surface, the vehicle's current waypoint and mission time are used as input when executing the path planning program. The produced plan is reduced to only a few

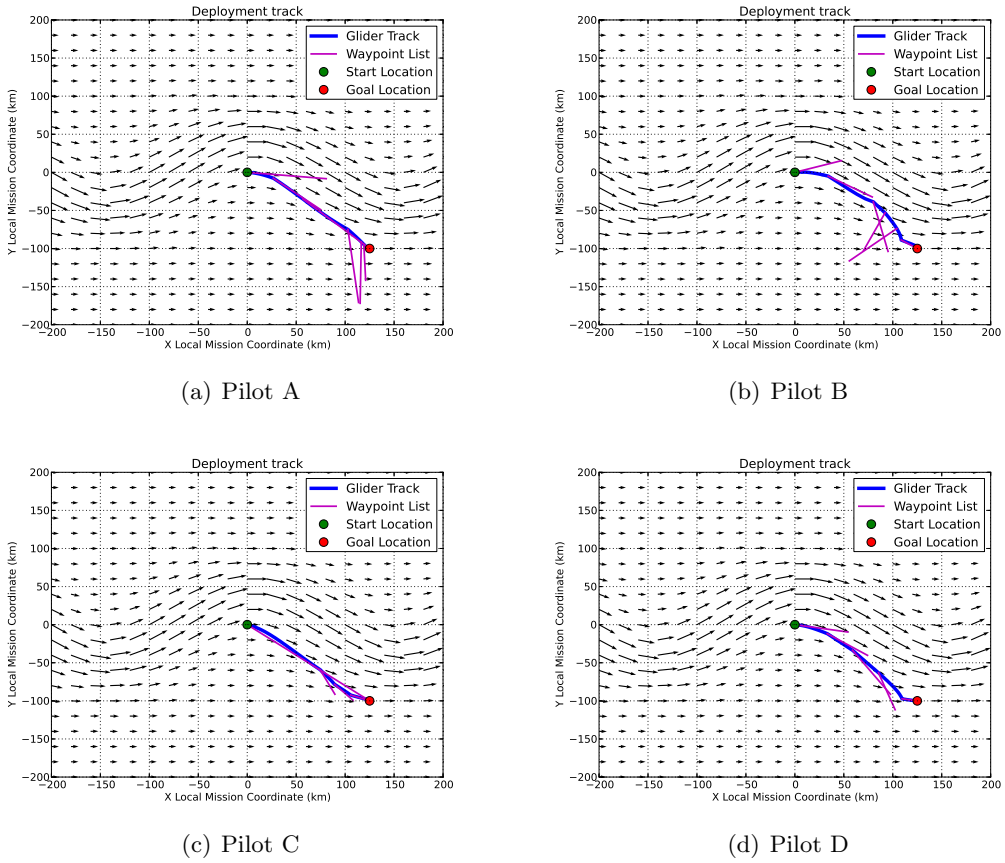
waypoints as the glider's software system is restricted to only a small list of points. Waypoints that are close to one another are also reduced. Despite not executing the exact track generated by the path planning system, this mechanism works within the constraints of the glider. Most of the extensions made to the vehicle's software have so far been made for the evaluation only. Because the aim is to use the same path planning infrastructure to guide a real fleet of gliders, further changes have been kept to a minimum. While the path planning system has the opportunity to re-task the vehicle every time it surfaces, in a real deployment, communicating with the AUV may not always be possible and so the revised plan always ensures that the final destination waypoint is also always included.

Results and Discussion

For the human piloted missions, four subjects were asked to traverse a meander in the Gulf Stream model with the aid of the graphical tool shown in Figure 7.10(a). The four pilots had many years of glider flight experience between them, ranging from over ten years to just a few hours. All subjects have been working in oceanography for years as physical or biological oceanographers, or oceanographic technicians.

Before starting the experiment, each subject was given a brief tutorial on the usage of the tool and was allowed to briefly experiment with it. The subjects were encouraged to use the current field slider to gain some insight of the currents in the area of operation. The pilots were also instructed to fly from the starting location to the end point in the minimum amount of time, while prioritizing the safety of the vehicle as if it were a real deployment.

The simulated mission was based off of a previously deployed flight. The glider was instructed to fly at a diving and climbing angle of 26° between 5–95 m. The water depth was set to 200 m so that the vehicle would not have to inflect early to avoid hitting



(a) Pilot A

(b) Pilot B

(c) Pilot C

(d) Pilot D

Figure 7.11: The flight tracks of the simulated mission by human pilots.

the ocean bottom. Finally, the simulated glider is equipped with two backscatter and fluorometer sensors.

Table 7.1 shows the summary of the evaluation. Several subjects performed the experiments more than once. In these cases, the deployment that had the shortest flight time is presented. Subject D performed the experiment once, subjects A and C twice, and subject B three times. Admittedly some subjects expressed they were slightly more aggressive, but within reason, on repeat attempts. This is likely due to their increase in comfort in using the tool and knowing that the AUV is indeed not real.

The human generated flight paths ranged in duration from 3.07 days to 3.34 days,

Pilot	Time (d)	Energy (kJ)
A	3.07	595.26
B	3.34	649.17
C	3.27	636.26
D	3.15	612.32
Auto	2.92	571.47

Table 7.1: Results of human piloted flights and the automatic path planner (Auto).

with overall energy consumption between 595 kJ and 649 kJ. The selected end waypoints were chosen specifically with the knowledge that if no action is taken by the pilot and no intermediate waypoints are provided, the glider would still make it to its destination in a sub-optimal amount of time and energy. The results of this "hands-off" approach was a flight time of 3.38 days with an energy expenditure of 657 kJ. Two of the pilots, B and C, discovered this approach on their first attempts.

For two out of the three subjects who performed multiple evaluations, the pilots were successful in decreasing their flight time on each successive attempt and thereby also reducing the energy dissipated. Subject B, after discovering the "hands-off" approach on their first attempt, did not improve their mission time on their second try. However, the subject quickly turned things around on the third attempt and bested their previous two flights.

The flight tracks of the missions of Table 7.1 are shown in Figure 7.15. As previously mentioned, the green indicator represents the start location of the mission and the red indicator the destination location. The blue lines show the path taken by the vehicle. The magenta line segments represent the assigned waypoints from the current glider location to the next waypoints. Thus, the start of a line segments is the current glider location on the track when the waypoint was assigned, and the end of a line indicates the first of the assigned waypoints. To reduce the number of lines in the figure, only the first assigned waypoint at each re-tasking is shown.

For the human piloted flights, the current correction algorithm was disabled as

suggested by the more experienced pilots. None of the test subjects seemed to have any qualms regarding this constraint and it was observed that compensating for the current seemed natural to pilots. With repeated experiments, it was observed that the subjects were also quickly refining this skill. In one particular case, a pilot quickly adapted and seemed to begin to emulate the navigation characteristics of one the more experienced subjects. This was especially interesting since the evaluations took place independently so that no one subject could learn from another.

In the flight paths show, in Figure 7.15, three of the four pilots tried to use the meander to their advantage as a more favorable current towards the target waypoint. Subject C, who used the "hands-off" approach on their first attempt (not shown), noticed that the vehicle slightly missed the target waypoint and had to backtrack. On their second attempt, Figure 7.11(c), they piloted mostly "hands-off" for much of the flight but tried to prevent the back tracking from east to west by changing course and flying south earlier.

Pilots A and D had similar flight paths and the two best mission times. Both decided to use the meander but were cautious not to get into currents too strong that would be difficult to escape from. It appears that Pilot A, Figure 7.11(a), was able to hug the meander longer than Pilot B in the mission of Figure 7.11(d) and so was able to clench a better time.

Subject B in the attempt shown in Figure 7.11(b) improved on the two previous flights. The subject was initially more aggressive than the other pilots by flying the deepest into the meander. However as they approached the destination it had become clear that if they were not careful they would be swept up. The pilot cautiously executed an escape maneuver, wanting not to put the mission at risk. The AUV was safely able to reach its destination in a respectable time.

The result of a completely automated path planning mission has a flight time of

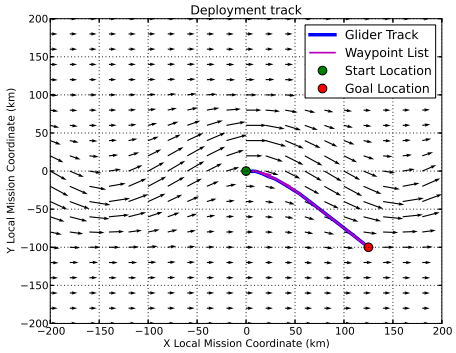


Figure 7.12: The deployment track of an automated path planning system piloting the simulated mission.

2.92 days with an energy consumption of 571 kJ as shown in Table 7.1. The track flown by the vehicle is show in Figure. 7.12. Because the path planning system had an opportunity to refine its path planning at every surfacing, we observe many waypoint list adjustments. Like the human subjects, the AUV was tasked to fly slightly into the meander for the additional speed. The path planning was also successful at navigating the glider out of the strong currents and to its destination.

Despite accomplishing a respectable flight time, further revisions on both the glider and the automated flight system could produce even better times. The result shown required some exploration of the parameter space in the path planning tool and the automated testing program. With further adjustments it may be possible to reduce or eliminate this exploration. For example, the automated testing tool could use the glider's observed average speed and provide it to the path planning system at each surfacing. The speed model in the path planning system itself could also be improved, for instance, by creating a customized speed model as described in Section 7.5.2. Finally, the path planning tool could provide the glider with sea current information that it will experience within the coming flight segment instead of using sea current information from the prior segment.

Although significant progress has been made in bringing an automated path planning system closer to being interpolatable with the Slocum Glider, an extensive amount of further testing must be performed. Other path planners should also be integrated and tested to provided several alternative options. A set of smart and intelligent route services exposed as part of the domain specific language would have a significant impact on the power of the programming framework. This assessment has taken strides towards this goal and compared how such a sample services measures against several human pilots.

7.6 Enabling Computation Intensive Applications

Real time examination of mission data can substantially enhance the overall effectiveness of AUVs in oceanography. However, many AUVs only allow a detailed analysis of data after completion of a mission. The ability to perform on-board analysis of real time data can become computationally intensive, requiring an energy efficient programming infrastructure that can be adapted to battery operated, energy constrained vehicles. A Linux SBC has already been integrated into the Slocum Glider as part of the framework. However, other more powerful multi-core computing platforms, like Intel's SCC (Single-Chip Cloud computer), that are capable of dynamically changing power and energy requirements may also become candidate computing platforms for cyber-physical systems (CPSs) like AUVs.

In this section, ocean modeling and path planning applications for the Slocum Glider are used to illustrate how the energy aware features of the SCC could be used to react to changing energy vs. performance trade-off requirements. Changes to these requirements can be triggered by the computational needs of other applications which are considered more mission critical resulting from the observation of an internal or external event. Clearly, avoiding obstacles has high priority when navigating through a busy shipping

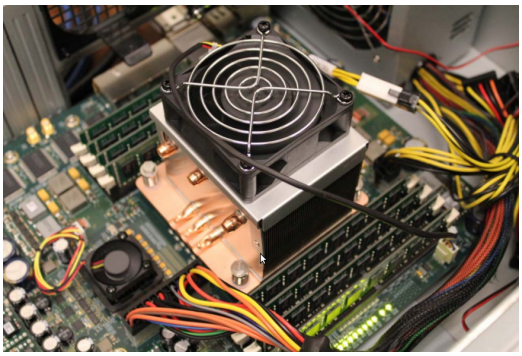


Figure 7.13: Intel's Single-Chip Cloud Computer.

lane. Encountering a physical phenomenon like an algal bloom could also trigger the use of additional sensors and data processing applications. In addition, later phases of a long duration mission may have to deal with reduced battery power and energy budgets, putting more severe constraints on the applications that can be effectively executed.

7.6.1 Single-Chip Cloud Computer

The Intel SCC has been designed to implement a cloud data center in silicon on a single chip [Howard et al., 2011]. The research chip has 48 cores grouped in pairs of two cores (tiles), a 24 router-mesh on-chip network with 256 GB/s bisection bandwidth between tiles, and four integrated DDR3 memory controllers [Gries et al., 2011; Howard et al., 2011]. Each core runs its own OS, thereby acting as an individual compute node. There is hardware support for message passing, but no hardware cache coherency policy is implemented.

The SCC system allows the power and energy management of individual cores and groups of cores, the on-chip network, and memory. Cores can be turned on and off. Frequency and voltage settings are software controlled and can be changed on the fly. This dynamic, fine grain power and energy management feature is the main characteristic of the SCC that should be exploited. The SCC is able to provide a significant

range of energy vs. performance trade-offs, giving AUVs the ability to perform mission critical, computation intensive tasks at the lowest possible energy cost. The SCC consumes between 25 and 125 watts when all cores are active [Mattson et al., 2010]. The speed of the on-chip network and off-chip memory can be adjusted, giving additional opportunities for performance vs. energy trade-offs. The SCC, shown in Figure 7.13, is an experimental platform and not commercially available. As part of an ongoing collaboration with Intel, the SCC is evaluated for deployment within Slocum Gliders.

7.6.2 Applications

Additional computational capabilities can save energy by more effectively managing the use of energy expensive sensors. The SCC is particularly well suited for this task because multiple energy saving algorithm/programs can run simultaneously on the chip, each with their own power and energy characteristics and trade-offs. This section will provide an overview of such applications.

Dead Reckoning - Localization is a critical challenge for underwater operations. Typically, collected sensor data is tagged with spatial and temporal coordinates. AUVs can use GPS localization while at the surface, and dead reckoning (DR) while diving. Unfortunately, DR can result in significant localization errors in the presence of underwater currents. A Doppler Velocity Log can be used to remedy this situation by performing bottom tracking, which allows the vehicle to measure its relative speed, thereby improving DR. However, operating the DVL sensor itself, and processing the acquired data can be energy and computation intensive. Without reliable localization many scientific missions are not feasible, including under-ice deployments where acquiring a GPS position at the surface is not possible. Section 7.3 further describes how significantly a DVL can improve a glider's dead reckoning.

Sensor Triggering - The stock Slocum Glider does not currently support fine-grained

or cross-sensor adaptive sampling. Sensors are typically turned on all the time, or active only on dives or climbs. The effectiveness of some sensors can be improved by making them part of a trigger chain, where low cost sensors activate more costly, but more precise sensors. Adaptive sampling may require significant physical modeling efforts and data processing capabilities. Such use of sensor triggering is described in Section 7.7.

ROMS - The Regional Ocean Modeling System (ROMS), [Shchepetkin and McWilliams, 2005], comprises a traditional ocean forecast model complemented by advanced variational data tools that allow the assimilation of 4-dimensional data, and more importantly, the sensitivity of the forecast to the present and future ocean state and the observational sampling pattern. For example, ROMS can be used to help optimize the path a glider takes between waypoints, or to indicate the regions where new observations would lead to the greatest improvement in forecast precision.

Charting the 3-dimensional and time varying pattern of these anomalies in ocean temperature and salinity represents an attractive testbed for integrating ocean observation and simulation through adaptive sampling and smart control on a single platform. Optimizing the integrated system will necessitate trading off the sampling frequency, the sensors that are active, the distance traversed by the AUV, the ocean model computational effort, and communication, all of which make demands on the available battery power and energy.

Path Planning - The task of the path planning algorithm, used in the evaluation, is to find a time-optimal path from a defined start position to a goal position while evading all static as well as dynamic obstacles in the area of operation, with consideration of the dynamic vehicle behavior and the time-varying ocean currents. The path planner is described in further detail in Section 7.5.1. Proper path planning can be crucial if

an AUV must arrive at a target location to observe a short lived phenomenon. It can also save time and energy since it allows the vehicle to navigate through strong ocean current fields. Thus, the energy investment made in computing a path plan may pay for itself if alternative longer paths were taken or the vehicle had to back track its position.

7.6.3 Evaluation

As discussed, the SCC is particularly well suited for parallel applications. For this reason, the ROMS and path planning programs are targeted for investigation to marry a CPS like a glider with a parallel capable infrastructure, like the SCC, that can provide the necessary knobs to trade-off power and energy restrictions with application runtime deadlines.

For the evaluations, custom settings for the SCC are generated. These settings initialize the SCC with different core, network and memory configurations. Because multiple programs are envisioned to be communicating at the same time on the SCC, non-standard mesh network speeds were also studied in the hope that they could provide additional insights on the trade-off space of the SCC.

Both ROMS and the path planning application make use of RCKMPI [Ureña et al., 2011] for message passing which should provide comparable performance to RCCE [Mattson et al., 2010; Ureña et al., 2011]. The MPD process manager, recommended for use with the SCC, is used throughout the experiments. All cores boot a Linux 3.1.4 kernel image. Finally, power measurements were gathered from the SCC infrastructure.

ROMS benchmark

The feasibility of running ROMS on the SCC is evaluated using a sample benchmark provided with ROMS. The benchmark consists of 512x64x30 grid points and 200 time

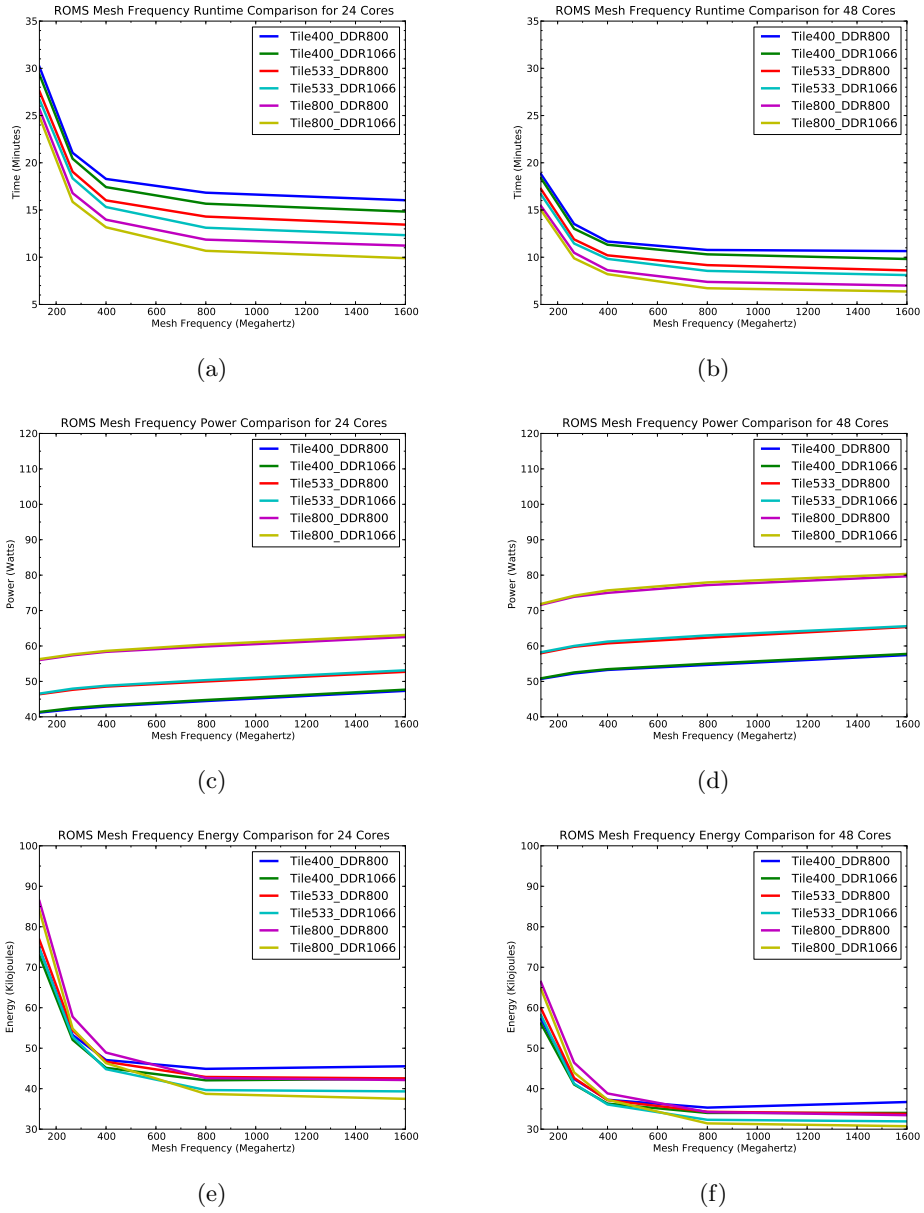


Figure 7.14: ROMS evaluation results for various SCC settings. The execution times for ROMS using 24 (a) cores and 48 (b) cores. The average power, (c) and (d), of the SCC during the execution of program. The energy required to run ROMS for 24 (e) and 48 (f) cores.

step iterations. The main computation is a two-dimensional stencil with nearest-neighbor communication. The grid is divided into tiles, where the total number of tiles must match the number of cores that are part of the computation. The grid's tile dimensions were chosen to maximize the size of grid points calculated per core and to reduce the size of the halo/ghost regions. Larger halo regions require more communication and computation. The tile dimensions used were empirically validated to be optimal for the grid size and the number of cores.

The evaluation of the ROMS benchmark program is shown in Figure 7.14. A diverse set of configurations of CPU, mesh, and memory were tested with both 24 and 48 cores. In most cases, the runtime for 48 cores, Figure 7.14(b), is lower than the 24 cores (Figure 7.14(a)) with the same setting. The fastest configuration with 24 nodes performed nearly identically to the second slowest configuration of 48 nodes, and outperformed the slowest. In scenarios where soft runtime deadlines are acceptable, numerous options and trade-off points are available for ROMS. A global application scheduler can consider these alternatives during the arbitration of the next SCC setting.

The average power consumption during the execution of ROMS is shown in Figure 7.14(c) and Figure 7.14(d) for 24 and 48 nodes, respectively. Throughout the experiments, lower mesh speeds reduced power by several watts. The most pronounced effect on power were high tile frequencies. Battery operated CPSs, like the Slocum Glider, may need to observe power caps during operation, since actuators and other systems can increase the power load on the device. Therefore, it may not always be an option to run the fastest configuration with the highest node count.

Similar to the runtime, it is generally more energy efficient to use 48 cores instead of 24 cores to run the benchmark. The highest setting for the 24 nodes in Figure 7.14(e) is, however, similar to the lowest configuration of the 48 cores seen in Figure 7.14(f). When comparing their respective runtimes, the 48 node setting does outperform the 24.

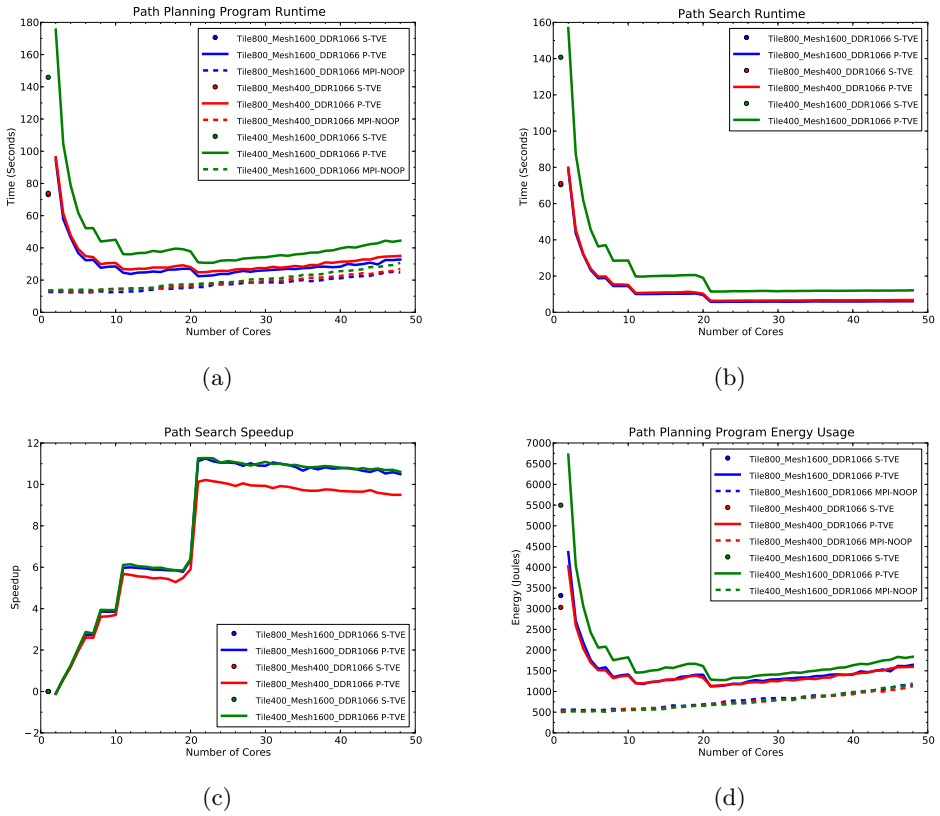


Figure 7.15: Evaluation results for the path planning program for various SCC settings. The runtime of (a) is the time required for the entire program to execute. The search time, (b) is the time required to perform the search for the optimal dive profile. Speedups for each of P-TVE is relative to the S-TVE with the same SCC setting. The energy required for the entire program execution (d) is based on (a).

Across the figures, the crossover points are very similar and are prospective trade-offs opportunities. Because of the dynamic nature of AUVs, mission priorities can change often, emphasizing the importance of a suitable arrangement for runtime, power and energy.

Path Planning

The serial (S-TVE) and parallel (P-TVE) versions of the TVE path planning algorithm have been ported to the SCC. The input parameters to both programs were identical throughout the benchmark tests. Since parameter choice can have an impact on the

amount of parallelism the program is capable of during execution, a set of parameters consistent with previous work [Eichhorn et al., 2012] were chosen.

The opportunity for parallelism that was exploited and implemented in P-TVE was to find the optimal dive profile depths for the vehicle. Because the AUV can experience different currents at various depths, it may be advantageous for the vehicle to glide within a certain depth range for portions of the flight. For each edge in the graph, this dive profile calculation is evaluated for 20 distinct depths ranges.

Results of the path planning programs for the SCC configurations are shown in Figure 7.15. S-TVE results are only available for one core since there is no parallelism involved. P-TVE has a master/slave architecture where the master delegates work to slaves that perform the dive profile task, so at least two cores are required. The MPI-NOOP results measure the overhead of the MPI infrastructure. It is a modified version of P-TVE which initializes MPI and immediately exits.

The program runtime, Figure 7.15(a), and dive profile search time, Figure 7.15(b), decreased as the number of cores increased for P-TVE. There is an initial communication overhead for two cores, when compared to S-TVE, as the master must delegate work to the slave. The step-wise behavior is explained by the number of iterations of work delegation that is performed by the master. For example, with 11 cores, 10 slaves perform work for two work iterations. In the case of 12 cores (11 slaves), the second work delegation will leave one slave idle. Because of the input parameter of 20 distinct depth range calculations, the optimal number of nodes should be 21. This accounts for one master with 20 slaves doing one iteration of work. Additional nodes only provide overhead in P-TVE as indicated by the speedup of the dive profile search in Figure 7.15(c). The speedup for each setting is normalized to the S-TVE search time of the same setting. If the number of profile searches is increased, additional cores could be used with a concomitant increase in benefit. Additional details are available

in [Eichhorn et al., 2012].

To reduce the power and energy of the program, idle slaves are instructed by the master to enter into sleep mode. In sleep mode, a slave performs an asynchronous receive call instead of a blocking receive call. This allows the slave to sleep in between update checks of the asynchronous call. Although this introduces latency for the first receive, it greatly reduces the overall energy used by the slave. This latency is evident in Figure 7.15(c), especially when there are a high number of idle slaves. For example, after 21 nodes, even the idle slaves that will never perform any work experience the latency because they wait for the termination message to be sent by the master.

The evaluation indicates that the path planning program is more reliant on computation than communication as the slowest core speed setting has the longest program and profile search runtimes. Lowering the mesh speed does decrease the speedup of the parallelization because it delays communication between the master and its slaves. However, the effect it has on runtime is not as significant as observed when changing the CPU frequency.

The energy required for the planning programs are depicted in Figure 7.15(d); it shows opportunities for trade-offs that could be used when choosing an SCC configuration that will run several programs simultaneously on the chip. Although the runtime of the P-TVE is generally longer for the low mesh speed configuration, the power saved by reducing the mesh frequency translates to a comparable energy profile of the highest speed SCC setting. After 21 cores, even the slowest tile setting could be considered, as the energy difference is not substantial. Similar to the runtime results, energy is wasted on idle slave cores. This issue should be addressed as part of future work.

7.6.4 Discussion

The applications described, along with others, could be required to run simultaneously on the SCC. Depending on the current needs of the system the priority of tasks may change periodically, or change based on observations of phenomena in the environment.

Power caps can also restrict the selection of high power SCC settings. A Slocum Glider typically uses alkaline battery packs, so the supply voltage drops as energy is consumed. A glider's fresh alkaline battery pack is rated as 1800 W h, while the SCC's power demands can range from 40 W to 80 W for the applications. As a comparison, the buoyancy engine of the glider operates at 60 W or more during inflections at 200 m depths. The vehicle must maintain a minimum voltage level at all times to operate safely. The use of actuators, like the buoyancy engine, and sensors, such as a DVL, will increase the power needed by the AUV. It may not be possible to run the SCC concurrently with some sensors, while other sensors can be active at the same time as the SCC provided that the chip does not exceed its allotted power.

Having knowledge of the trade-off points for an application is critical when choosing a configuration setting. For example, at some point in a deployment, a vehicle's path may need to be resolved rather quickly. Ideally, the highest tile, mesh and memory speed (Tile800_Mesh1600_DDR1066) should be chosen and P-TVE is run on 21 nodes. However, there could be a loose deadline to perform modeling and thus ROMS must also be considered. If the highest setting exceeds the allotted power, a small sacrifice could be made by lowering the mesh frequency. The impact on the runtime and energy of path planning is minimal. While the impact is greater for ROMS, it may still fall within the soft deadline restrictions.

The ROMS trade-off scenario described could also be made in the case of a power cap. If there is no need for path planning, and the requirements are such that ROMS should have nearly the same runtime and energy profiles as the best setting for 24

nodes, then the program could be run on all 48 cores at half the tile frequency. This allows the program to not only be more runtime and energy efficient but also greatly reduces the required power. The lowering of the frequency, in this case, is what may be needed to bring the power profile below the cap.

Although the focus thus far has been on power cap scenarios, other trade-off points which concentrate on energy and runtime can be made. This is especially true if more applications, like sensor triggering, are involved in the deliberation. Other cyber-physical systems will have their own hardware and software restrictions and priorities. The SCC can provide CPSs a trade-off space in which it can make decisions that involve runtime, power, and energy.

7.7 Adaptive Feature Based Energy Management of Sensors

Increasingly complex sets of sensor payloads that are integrated into today's AUVs place significant demands on a vehicle's energy resources. Effective management of the limited battery energy has become a critical challenge since not all sensors can be active at all times during a mission. As a result, sensors are typically managed manually by remotely switching them on or off, and fixing a data sampling policy *a priori* for each dive segment while the AUV is at the ocean's surface and able to communicate via a satellite link with researchers. The availability of energy efficient computing platforms with significant computing capabilities, like AVBot, enable *in-situ* data processing and decision making, and allow sensors to be activated in flight when the likelihood of collecting scientifically relevant data is above particular thresholds.

In many ocean science applications, energy expensive sensors may be triggered based on observed or predicted features in the surrounding environment. Such features may include a particular depth or lighting condition, the presence or absence of a chemical, or the presence or absence of life forms. Onboard computing systems allow an AUV

to change its flight pattern and sensor readings depending on such features, making feature based or adaptive ocean sampling a reality [Schofield et al., 2007].

An example of adaptive sampling that has gained attention in literature is thermocline tracking [Wang et al., 2009; Woithe and Kremer, 2009; Petillo et al., 2010; Zhang et al., 2012; Cruz and Matos, 2010a]. A thermocline is a layer in the water column where temperatures change drastically with depth, separating the warmer mixed layer from the deep water layer. Figure 4.4(a) of Section 4.4 shows the temperatures of a water column measured using a Sea-Bird CTD profiling sensor lowered from a research vessel off the coast of New Jersey. The thermocline becomes apparent by the sudden change in temperature of the water column starting at a depth of seven meters and ending near twenty meters.

The study of thermoclines has both military and scientific interests. They can influence the propagation of sound, i.e. sonar, which is important in submarine warfare [Haeger, 1995]. Additionally, phytoplankton, which are responsible for much of the oxygen in the atmosphere and are an important link in the ocean's food chains, can reside near or within a thermocline [Gessner, 1948].

Current thermocline tracking algorithms influence the AUV's vertical flight profile in the water column to remain within the boundaries of the thermocline. This enables larger and more fine-grained spatiotemporal data sets to be collected by the vehicle in the target area. However, this approach may be accompanied by a significant increase in energy consumption, especially for buoyancy driven AUVs like the Slocum Glider. They provide an interesting challenge because the buoyancy engine is such an expensive resource and is used sparingly. Propeller driven AUVs constantly run their engines, so the energy impact of performing tracking is not as great. Gliders would also not make significant forward progress if the thermocline is very thin and would require a minimum vertical span to be traversed between inflections to make proper flight feasible.

Additionally, the CTD sensors on most Slocum Gliders are not pumped but instead use the vehicle's momentum to achieve water flow through the sensor. If proper flight and flow are not achieved, inaccurate readings will be recorded [Alvarez and Stoner, November, 2012]. As a result, a glider would not be able to engage in prolonged tracking missions using this mechanism as it is simply too costly and/or too inaccurate.

In this Section describes how feature detection algorithms, like thermocline tracking, can be used to energy manage and trigger other correlated sensors to conserve energy. The following main contributions are presented:

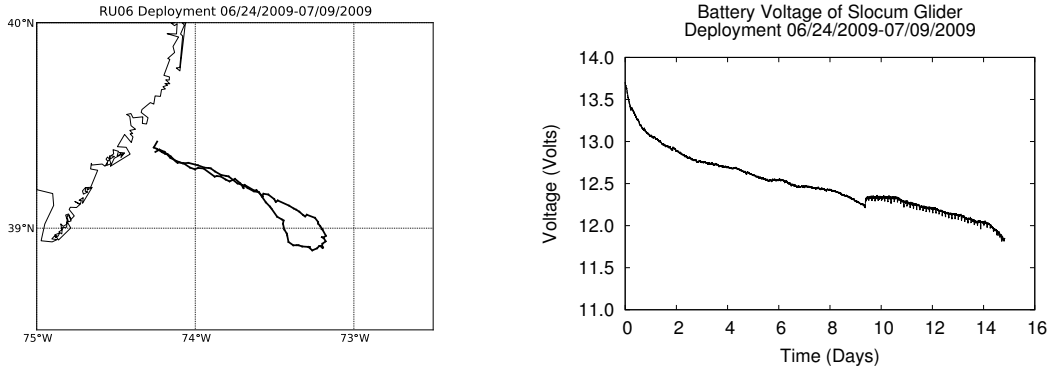
1. Emphasize the importance of sensor energy management with a brief description of a real-world deployment in 2009 that had to sacrifice sensor readings.

2. An overview of previous work in thermocline detection and tracking algorithms and their use in AUVs.

3. An evaluation of how thermocline detection algorithms together with more low cost sensors may be used to manage power-hungry sensors through hardware and software trigger chains. Data from the previous Slocum Glider deployment and sensor energy models built from oscilloscope measurements are used to assess sensor management strategies.

4. The implementation and sea trials of a trigger chain based on a thermocline tracking algorithm with two biological sensors within the Slocum Glider off the coast of New Jersey.

Thermocline tracking was chosen as a generic example to showcase this energy saving strategy because it is applicable to many AUVs since most can be equipped with a CT or CTD sensor [Petillo et al., 2010]. Zhang et al. [2009] also performed sensor triggering by activating gulpers to capture water samples in a thin phytoplankton layer using an algorithm to determine fluorescence peaks. This triggering was motivated by the



(a) Glider deployment off the coast of New Jersey. (b) Battery voltage during the deployment.

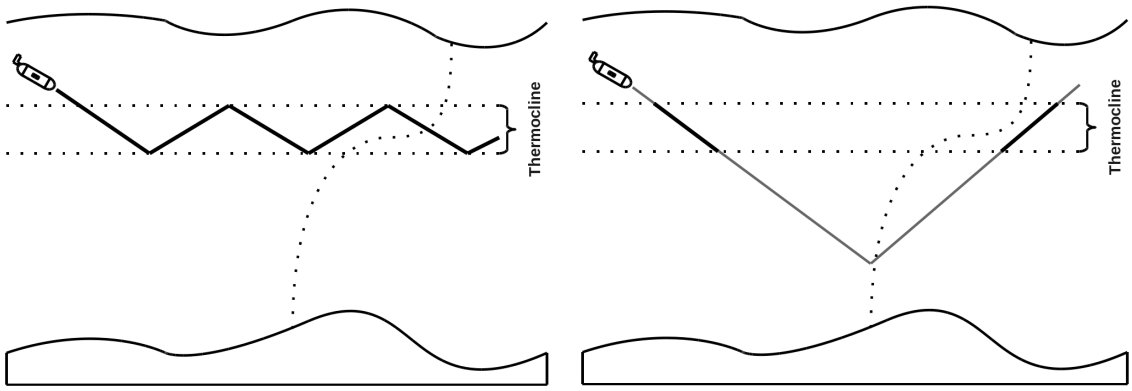
Figure 7.16: (a) A glider deployment off the coast of New Jersey in 2009 and (b) its measured battery voltage. The increase in voltage on day nine corresponds to the shutdown of the ECO-Puck sensors.

sole incentive of gaining the most beneficial data samples out of the limited number of gulpers onboard. Although it is also proposed that algorithms like thermocline tracking should trigger sensors at opportune times, the motivation is instead energy efficiency.

7.7.1 Glider Deployment – Manual Sensor Management

Gliders often operate in unpredictable environments due to shifting underwater currents and weather conditions. Adverse conditions, resulting in unplanned changes to the vehicle's route, may effect the overall energy consumption of a mission. Getting a glider back home safely, i.e., back to shore or to a recovery vessel, is mission critical. Therefore, energy budgets contain mission-specific energy reserves and as a result, only a limited percentage of the overall energy budget is available for the sensor payload. Managing energy is therefore critical for the overall scientific effectiveness of a mission.

Figure 7.16(a) shows a glider deployment off the coast of New Jersey in 2009. A Slocum Glider was equipped with WET Labs' Environmental Characterization Optics (ECO) puck sensors, the BBFL2S and BB3SLO, which measure fluorescence and backscatter. The glider was tasked to fly to the continental shelf and back to shore.



(a) Flight profile tracking. (b) Sensor triggered tracking.

Figure 7.17: Thermocline tracking in which the AUV (a) changes its flight profile to fly and sample within the thermocline or (b) where the AUV activates and samples from its sensors only within the thermocline. Sensor activity is indicated by the darkened profile lines.

During the deployment, a storm caused the vehicle to drift south while it continued to progress towards the shelf. Eventually, with the glider not making enough progress towards the commanded waypoint, it was retasked to fly back to shore. On the ninth day of the deployment it was determined that the glider's overall battery voltage, shown in Figure 7.16(b) and an indicator of the battery's life, was becoming low considering the remaining journey home. Thus, non-critical sensors, including the ECO-Pucks, were turned off manually to conserve energy for the flight (as observed by the increase in voltage in Figure 7.16(b)). Potentially significant data collection had to be stopped in order to allow the safe return of the vehicle. In Section 7.7.4, this deployment will be used to investigate the benefits of feature based, adaptive sampling for sensor energy savings, and what potential impact it could have had for the scientific value of the overall mission.

7.7.2 Thermocline Detection and Tracking

Thermocline tracking algorithms are a prime example of the adaptive sampling possible with AUVs. Recent work in [Wang et al., 2009; Woithe and Kremer, 2009; Petillo et al.,

2010; Zhang et al., 2012; Cruz and Matos, 2010a] perform this tracking by influencing the vehicle's flight profile as illustrated in Figure 7.17(a). This yields an increase in the scientific data gathered by the AUV in this region. These are unlike the proposed approach, shown in Figure 7.17(b), where sensors are energy managed and used only at opportune times in the profile. These same algorithms, with alteration, may also be used to trigger sensors.

A common feature of these algorithms is that they place newly collected depth and temperature data into depth bins. The size of these depth bins may be static or dynamic depending on the algorithm. Readings in each depth bin are averaged to produce a filtered result for the bin. The bin data are then used in the following ways to perform thermocline tracking:

- *Alg1*: In Section 4.3.1, the depth bins are traversed to find consecutive bins that meet both depth and temperature thresholds that are specified *a priori*, for example, a change of 5°C over three meters. These thresholds can be changed when the vehicle establishes communication, where new values may be calculated automatically or set by a flight engineer. Although simple, this algorithm has been deployed to track a thermocline on a Slocum Glider with success. This approach is similar to [Wang et al., 2009] where thresholds for the change in the speed of sound with regard to depth is instead specified *a priori*, for example by a forecasting model.

- *Alg2*: In [Petillo et al., 2010], a tracking algorithm for use within the MOOS-IvP autonomy system was developed [Benjamin et al., 2010]. Once the depth bins are averaged, the vertical derivatives (the change of temperature over the change of depth) are calculated for each bin. The average of the vertical derivatives is then used to determine the upper and lower bounds of the thermocline. Any depth bin whose vertical derivative is greater than the average derivative is considered to

be part of the thermocline. The algorithm requires an initial profile and periodic resets of the depth bin data to ensure variations of the thermocline are detected. Field experiments using an IVER AUV have successfully tracked a thermocline over a 1.5 h period using this algorithm.

- *Alg3*: Zhang et al. [2010, 2012], performed tracking by using peak-gradient detection. For each ascent and descent leg, the maximum derivative of temperature over the depth bins is used to establish the next target depth for the vehicle to fly. An extension depth is added or subtracted to the target depth so that the algorithm does not get contained in a local maxima and in case the depth of the thermocline changes. This technique was developed on data collected from a Dorado AUV and has since been deployed on the Tethys AUV.

- *Alg4*: In [Cruz and Matos, 2010b,a], a state machine was used to detect and track a thermocline when vertical temperature gradients exceed a set of thresholds. The thresholds may be specified, but are not required since the algorithm will update the thresholds based on hints from the previous dive profile. As the vehicle traverses the water column, the algorithm uses these thresholds to determine the top and bottom edges of the thermocline. The algorithm also records the state transitions that occur to provide feedback of its performance, i.e. how often it detects the top, bottom, or the entire thermocline. This technique has been demonstrated on the MARES AUV to track thermoclines with gradients as low as 0.3°C/m.

Each of the thermocline detection and tracking algorithms have their own characteristics, and it is the responsibility of the oceanographer or marine scientist to select the algorithm that matches best the particular scientific application desired. Each algorithm has its advantages and disadvantages. For example, *Alg1* is very simplistic and

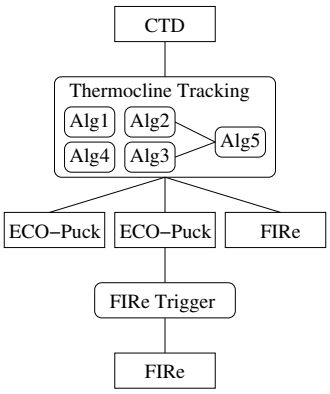


Figure 7.18: Sample trigger chains for thermocline tracking. Any path from the root (CTD) to a leave node demonstrates a possible trigger chain.

only tracks features specified by its temperature and depth parameters. This differs from other algorithms where the parameters may be detected automatically. However, some mission objectives may not require or want this automatic detection and want only those very specific features defined by the parameters for *Alg1* to be recorded. Regardless, the focus of this work is not on the performance, but rather the potential energy savings of the algorithms if they are used to energy manage more advanced sensors. The following sections will describe implementations based on the described thermocline tracking algorithms and show how they may be advantageous when implemented on an AUV such as the Slocum Glider to conserve energy.

7.7.3 Trigger Chains

Instead of the vehicle changing its flight profile, the glider should rather makes use of the aforementioned algorithms to activate and deactivate sensors. The standard CTD sensor equipped on the vehicle could be used to trigger more advanced correlated sensors. In the case of the thermocline, for example, fluorescence and backscatter sensors may be triggered to determine if phytoplankton are present in the tracked area.

Figure 7.18 shows an example trigger chains for the sample application. Each path from the root node (CTD) to a leave node represents a possible trigger chain. A trigger

chain contains hardware (sensors) and software components (services). For instance, one possible trigger chain is (CTD → Alg1 → ECO-Puck) where the thermocline detection algorithm *Alg1* uses the CTD readings to trigger the ECO-Pucks. One maximal trigger chain as shown in the figure is (CTD → *Alg1* → ECO-Puck → FIRe Trigger → FIRe). Here, the ECO-Puck is used as an input to a FIRe trigger program which controls the operation of the FIRe sensor that assess the health of phytoplankton populations.

As discussed, the thermocline detection algorithms have different classification characteristics. Depending on the application, a scientist may want to log as much of the thermocline as possible with the ECO-Pucks. At the same time, it may be sufficient to only get a subset of the readings with a FIRe sensor while the pucks are active. Thus, the FIRe sensor trigger is dependent on the ECO-Puck trigger. This requirement may determine the selection of the feature recognition since some of the algorithms may classify more of the water column as part of the thermocline than others. Not only will this affect the activity of the ECO-Pucks but possibly other child triggers.

The software and sensor trigger chains shown are by no means complete and may involve other sensors as well. For instance, the FIRE sensor may be triggered only if the ECO-Pucks recognize a sufficient concentration of chlorophyll to indicate the presence of phytoplankton. On the other hand, a light sensor may also enable or calibrate the FIRe sensor. It is important to note that different feature recognition algorithms or triggers may execute concurrently to support complex sensing tasks. This is similar to ensemble modeling used in domains such as weather prediction or biomedical research.

The notion of trigger chains is different from energy optimization approaches such as hierarchical power management. Hierarchical power management uses redundancies in system components to select the most energy efficient components to produce the overall result [Sorber et al., 2005]. In the trigger chains, each stage in the chain decides whether to activate the next stage or not, with the last stage activating the target

sensor which is represented by the leave node in the chain. Only this target sensor acquires the desired sensor data, not any of the intermediate triggering nodes.

7.7.4 Evaluation

Thus far, feature detection and tracking algorithms, and how their current implementations may not be well suited for vehicles like the Slocum Glider have been described. To evaluate how effective the proposed sensor management technique may be, the energy used in a previous glider deployment both with and without the proposed sensor triggering mechanism is estimated and compared through simulations. In this section will also showcase the potential benefits gained with this approach. Finally, a flight segment from one of two real deployments where sensor triggering was used is described.

Simulations

All simulations are based on the deployment in Figure 7.16. The log files generated by the vehicle were used to replay and simulate the triggering of sensors throughout the mission. The energy model for the sensors are based on a BB2FLSV5 WET Labs ECO-Puck sensor, comparable to those used in the real deployment. The BB2FLSV4 and BB2FLSV5 backscatter and fluorescence sensors installed in a Slocum Gliders are shown in Figure 7.19 and use approximately 0.61 W each as measured by a Tektronics TDS 3014 oscilloscope.

As discussed, extensive use of sensors can negatively impact vehicle endurance. In order to compare optimizations, a baseline energy estimation must first be established. Based on the logged data files, the estimated baseline energy usage of the ECO-Puck sensors is 890 kJ for the nine days of activity in the mission. To gain additional perspective, using energy models derived from the power measurement infrastructure, the buoyancy engine required an estimated 488 kJ throughout the entire mission. Thus,

Figure 7.19: BB2FLSV4 and BB2FLSV5 sensors installed in a Slocum Glider.

the puck sensor package for only a portion of the sea trial consumed nearly twice the energy required to drive the vehicle for the entire deployment.

Recall that the algorithms described in Section 7.7.2 are traditionally used track thermoclines by altering the AUV's flight profile. Instead, they have been implemented and adapted to trigger sensors only in areas of interest with the aim of conserving energy without a significant sacrifice of data quality. The algorithms observe temperature readings with the glider's CTD sensor and based on its evaluation, trigger the fluorometer and backscatter sensors. Thus, the leftmost trigger chain of Figure 7.18, namely (CTD → Alg1 → ECO-Puck) is showcased. A rightmost trigger chain such as (CTD → Alg1 → FIRe) could use these same algorithms but for a more power-hungry sensor, namely the FIRe sensor (which consumes about 5W) instead of the 0.61 W consumed by the ECO-Pucks. The resulting energy savings for the latter trigger chains would be much more dramatic.

In the implementations, all algorithms use a one meter depth bin size. They collect and calculate the average temperature for the depth bins and use it in the later stages of the algorithm. Additionally, *Alg1* discards any data older than ten minutes and *Alg2* discards data hourly as suggested by the authors. Except for *Alg1*, the ECO-Puck sensors are required to be turned on for the first dive after each surfacing to ensure no

valuable data is lost. *Alg1* is an exception because it does not require an initial dive to bootstrap the algorithm.

In *Alg1*, the averaged temperatures of the bins are used to determine if a threshold parameter of 3°C or more has occurred within four meters. Any portions of the water column that have met the criteria are flagged as thermoclines. These parameters are not changed throughout the simulations, although an engineer or a mission planning system could in a real deployment. Because data is discarded at approximately one hour intervals for *Alg2*, the algorithm will force the sensors on for the immediate next dive profile. This is to ensure no relevant data is lost. The periodic resets allow the algorithm to adjust more quickly to variations of the observed thermocline over time.

Both *Alg3* and *Alg4* in their current implementation do not discard any temperature and depth bin data within a flight segment. The data is however discarded when the vehicle is at the surface. An extension depth of three meters was chosen for *Alg3*, which is larger than the value of two meters suggested by the respective authors. This captures a wider area around the peak gradient since thermoclines near the coast of New Jersey can be quite large during the summer months, as show in Figure 4.4(a).

The initial values of the thermocline, top, and bottom thresholds of *Alg4* are set to values thought appropriate for the mission being simulated. The thresholds quickly converge automatically, as per algorithm specification, after the first dive in the segment. The three thresholds are adjusted at each inflection point by the parameters in [Cruz and Matos, 2010a]. Upon surfacing, these thresholds are again reset to their initial values.

An additional algorithm is also introduced, which will be referred to as *Alg5*, that makes uses of both *Alg2* and *Alg3*. As will be clarified further, *Alg2* has a tendency to be more liberal with its usage of the pucks, while *Alg3* is more conservative. However, depending on how the water column is observed by *Alg3*, the true maximum gradient

Algorithm	Energy (kJ)	Recall	Precision
Baseline	890	0.99	0.14
Alg1	163 (18%)	0.55	0.38
Alg2	587 (66%)	0.89	0.19
Alg3	167 (19%)	0.62	0.42
Alg4	247 (28%)	0.69	0.33
Alg5	327 (37%)	0.77	0.30

Table 7.2: Algorithm Results (First 9 Days)

Algorithm	Energy (kJ)	Recall	Precision
Baseline	890	0.60	0.09
Alg1	285 (32%)	0.52	0.37
Alg2	879 (99%)	0.87	0.24
Alg3	281 (32%)	0.68	0.48
Alg4	435 (49%)	0.71	0.36
Alg5	489 (55%)	0.79	0.38

Table 7.3: Algorithm Results (Whole Mission)

of the column may not be detected until very late into the flight segment. Instead, the algorithm may trigger the sensors in what is considered the local maxima. After several dive and climb legs, the temperatures inside bins should stabilize and the algorithm should only trigger the sensors near the true maximum gradient.

Alg5 considers the two dependent algorithms as software sensors. By default it triggers the ECO-Puck sensors when *Alg2* suggests, but continues to monitor the output of *Alg3*. When *Alg3*'s peak gradient is stable for three consecutive dive/climb profiles, triggering is switched to fire the pucks based on its suggestions instead of *Alg2*. If the peak gradient becomes unstable, triggering is immediately switched back again to *Alg2*. This is shown in Figure 7.18 and demonstrates that trigger chains can apply to software as well as hardware.

The results of the simulations and the baseline energy usage of the sensors for the mission are summarized in Table 7.2 and Table 7.3. Table 7.2 provides a snapshot up

to the point in the actual deployment where the sensors were turned off and no longer used for the remainder of the actual mission. Table 7.3 extends the usage of the sensors for the remainder of the sea trial.

Recall and precision are two metrics commonly used to assess the effectiveness of information collection and retrieval systems [Manning and Schütze, 1999]. Informally, the recall metric describes how much relevant sensing information is missed (false negative) when using a trigger chain as compared to the scenario where sensors are always on. The precision metric captures the case where irrelevant sensing information has been collected (false positives). For each of the presented algorithms, there is a balance between collecting mission relevant data (recall) and mission irrelevant data (precision). These metrics are applied here to classify the thermocline detection and tracking algorithms. For example, an algorithm that never triggers a sensor has a 0% recall, but a precision of 100%. In contrast, an algorithm that always triggers a sensor has a 100% recall, but a very low precision (due to acquiring large amounts of irrelevant data).

Precision and recall of the five algorithms are presented in Table 7.2 and Table 7.3. To define the two terms, a standard that is considered relevant data within the simulations must first be defined. Data is defined as relevant if it is recorded by the ECO-Pucks within three meters of the maximum gradient of the water column. The data used to calculate this gradient is based on averaged temperatures for the depth bins for the whole flight segment. This definition is very similar to how *Alg3* detects its thermocline and thus it is expect to be favored. What should be considered the standard may be application dependent, but this definition was chosen since *Alg4* also makes use of the maximum gradient internally. An algorithm's recall is thus defined as the proportion of the number of logged readings within the relevant area to the number of possible readings within the area, if performed, for example, by an oracle. The recall of the baseline for the first portion of the mission is high because the glider

was instructed to have the sensors on for nearly the entire time. However, the recall drops dramatically when the whole mission is considered because no measurements were recorded for the second portion of the deployment.

The precision of an algorithm is defined as the proportion of relevant readings recorded to the total number of readings recorded. The baseline's precision is very low throughout the deployment because the sensors were active for large portions of the water column outside the area of interest. It is nearly impossible to achieve perfect precision since most algorithms require the sensors to be powered for the first dive of a flight segment.

Of all the implementations, *Alg2* achieved the least amount of energy savings (34%), because the algorithm tends to over estimate what is considered to be part of the thermocline. This is because the average of the temperature and depth derivatives of the bins is used to qualify if a given depth bin is part of the thermocline. Then, the depth range that the algorithm considers to be part of the thermocline is between the most shallow and deepest depth bin. Depending on the temperature profile observed by the vehicle, this average can become skewed to include large parts of the water column that are not of interest. For this reason, this algorithm was previously referred to as being liberal with its usage of the sensors. This causes *Alg2* to have the best recall of all the algorithms, but like the baseline, the precision is relatively poor.

Algorithm *Alg3* achieved the second most energy savings (81%) and provides a nice balance of recall and precision, and is therefore preferred over *Alg1* (82%). Since the recall and precision are based off of a standard which closely resembles how this algorithm performs its thermocline tracking and triggering, both the recall and precision are high. In fact, it has the highest precision of all the algorithms in both tables. As previously mentioned, *Alg3* will likely never have perfect precision because it is forced to keep the sensors on for the first dive of each segment. Likewise, it is practically

unfeasible for its recall to be perfect, because it is tracking and triggering sensors as the water column is being observed and does not have a complete picture of the environment.

The hybrid algorithm, *Alg5*, has performance results that lie in between its dependent algorithms. Although it uses more energy than *Alg3*, its recall is much better. In other words, it captures relevant data that *Alg3* misses because it is presumably focused a local maxima at some points throughout the flight. *Alg5* is however, not as precise, but still much better than *Alg2*. While *Alg2* may be over estimating the thermocline region, *Alg5* will eventually switch the firing of sensors as soon as *Alg3*'s peak gradient is stable.

The performance of all the algorithms nevertheless rely on parameters and the evaluation standard they are judged against. For example, *Alg1* captures the water column based only on the depth and temperature thresholds specified. *Alg2* requires periodic resets of the depth bins. Like *Alg1*, this parameter may require fine tuning during deployments. The extension depth parameter of *Alg3* could cause the area around the thermocline to be over or under sampled depending on what is considered relevant scientific data. *Alg4* also contains the concept of an extension depth, and also requires parameters that specify how the thresholds for the thermocline should be adjusted. Thus, the energy savings presented in Table 7.2 and Table 7.3 could change depending on the parameters specified. However, reasonable defaults were chosen for the implementations given the glider flight data.

The energy saved using the triggering strategy can have a significant impact on both the endurance of the vehicle as well as the scientific data samples that can be collected. The glider, unlike propeller driven vehicles, are meant for long term deployments to collect data over long periods of time. Using sensor triggering as described, all implemented algorithms would be capable of extending the sensing of ECO-Puck data to

the whole mission. In the case of *Alg3*, the glider could have also been equipped with the expensive FIRe sensor and triggered it for half the time the algorithm detected the thermocline. Alternatively, the energy saved could have been allocated towards the flight of the vehicle to extend the deployment by approximately 3.5 days or 24%.

Deployment

Section 7.7.4 described how sensor triggering can be used to save significant amounts of energy through simulations of a previously flown mission. This technique has also been developed and deployed in two separate sea trials using the Slocum Glider. The deployments took place out of Belmar, New Jersey, approximately 26 km off the coast and in 60 m of water.

The algorithm implemented for the mission was based on *Alg3*. However, instead of collecting and averaging the temperatures within the depth bins, only the latest temperature per bin was used. The algorithm also calculated the peak gradient after each new CTD sensor reading instead of calculating it only at inflection points as described in [Zhang et al., 2012]. Thus, at times, the peak gradient could shift while flying through the thermocline as new CTD data was collected.

The algorithm was run on the AVBot single board computer as a service. The SBC communicated with the vehicle via a RS-485 serial connection to the glider's flight controller. Previous, the SBC was connected to the vehicle's science computer, however, to decrease the delays in sending control commands, it now communicates directly with the flight controller. This trade-off causes delays for the SBC to receive CTD readings because the science computer must first send the data to the flight controller before it can be sent to the SBC. Ideally, separate data lines should be connected o both vehicle computers. A simplified version of the algorithm could have also been implemented using the GLOC scripting language on a stock glider's flight controller.

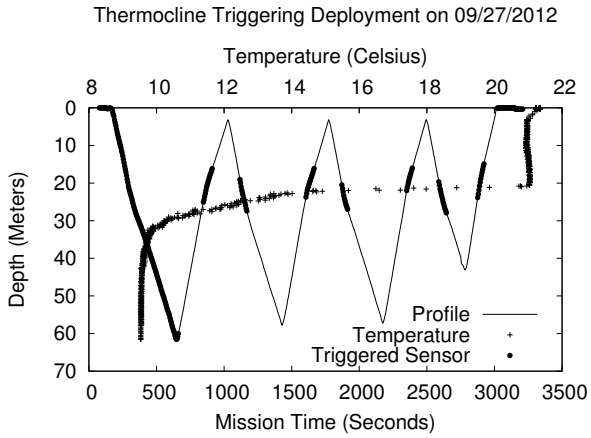


Figure 7.20: A thermocline tracking deployment on September 27th, 2012 off the coast of New Jersey. The solid line depicts the glider's flight profile. The darkened sections of the profile present the portions of the profile where the tracking algorithm triggered the use of sensors.

The flight profile of one segment of the two sea trials is shown in Figure 7.20. The darkened sections of the profile indicate when the algorithm activates the ECO-Puck sensors. The drastic change in temperature through the water column is indicative of a thermocline. Like the simulations, the sensors were forced on for the first dive profile of the segment. Due to implementation concerns, the sensors were also on while at the surface. For the segment shown, an extension depth of five meters was used around the peak gradient, although several segments were also flown with three and four meter extension depths. The profile data shown was obtained from the flight controller. It is apparent that there is a delay between when the SBC receives the temperature and depth data to when the trigger occurs because the center of the darkened lines are not aligned. As previously mentioned, the CTD data must first be sent to the flight controller which in turn sends it to the SBC. When the algorithm wants to trigger the command to fire a sensor, the command must first be sent to the flight controller which forwards the request to the science CPU that activates the ECO-Pucks. However, from the SBC point of view, the sensors were triggered at the correct time and the darkened lines align with the peak gradient. For this reason, several segments were performed

Algorithm	Energy (kJ)	Recall	Precision
Baseline	3.8	1.00	0.09
Alg3 Inspired	1.4 (37%)	0.88	0.23

Table 7.4: Deployed thermocline triggering results

with different extension depths because of the possible data propagation delays.

The results of the *Alg3* inspired algorithm are shown in Table 7.4. The baseline of the table indicates the energy, recall, and precision that would occur if the vehicle flew normally without the sensor management technique. The recall of the deployed algorithm is fairly high, even though the data propagation delays and CTD sampling caused the maximum gradient to shift slightly throughout the flight. The precision is low compared to the simulation results of *Alg3*. Again, this is due to having the sensor on at the surface and during the first dive of the segment. The segment was also short, approximately one hour, while typical segments in missions are three to four hours long. This has a negative affect on the precision because the time at surface and initial dive is proportionately large relative to the rest of the flight. Compared to the baseline, triggering of sensors saved a significant amount of energy. The estimated cost of using the buoyancy engine for the whole flight segment is 1.3 kJ. Therefore, the baseline energy of the sensors is nearly three times that of the buoyancy engine. The implemented algorithm on the other hand, is only 7% more than the energy of the engine. The dramatic conservation of energy observed in both the sea trials and the simulations leads us to believe that the proposed sensor management technique would indeed have a substantial impact, especially on long endurance missions.

7.7.5 Discussion

Dynamic feature tracking, like thermocline tracking, demonstrate a promising step forward in making AUVs more effective scientific and military instruments. A new

sensor management technique was described that makes use of published thermocline tracking algorithms to simulate the triggering of sensors while a vehicle is within the thermocline instead of modifying its flight profile. This sensor management technique has also been successfully used in two real world sea trials and was shown to be capable of saving a significant amount of energy.

The thermocline tracking algorithms and triggers were implemented as services in the new programming framework and executed on AVBot. Simplified versions of these algorithms could have also been implemented for the GLOC scripting engine and executed on the flight or science computing platforms on a stock vehicle. Hierarchical power managements techniques like in [Sorber et al., 2005; Banerjee et al., 2007] could be used in conjunction with AVBot. For example, a specialized GLOC script could execute every control cycle to update its view of the water column so it is consistent with the real world. When the thermocline tracking services is later executed to detect the thermocline, it can use the "cached" view of the water column. This is architecturally similar to the StrongARM computer periodically updated its web cache while acting as a proxy server for a laptop in Turducken [Sorber et al., 2005].

In Section 4.6, the thermocline sensor specification as part of a state made used the exact same thermocline tracking services created in this Section. This showcases that the programming infrastructure is mature and capable of bringing advanced services, such as the thermocline triggering, from software simulation to sea. The presented trigger is also simple to specify in the language and enables the programmer to easily explore energy and sensing trade-offs.

Chapter 8

Conclusion and Future Work

Autonomous underwater vehicles have become a crucial component in studying the world's oceans. The Slocum Electric Glider is a popular AUV used by the scientific community, private sector, and the military to perform long endurance missions. However, like other AUVs, it is based on the layered control subsumption architecture and can be difficult to program. Users of the vehicle typically do not create new missions themselves but simply use the existing set of missions developed and tested by the manufacturer with different arguments.

In this dissertation, I presented a new energy aware programming framework for autonomous underwater vehicles. The framework currently uses the Slocum Electric Glider as its implementation platform, however much of the work is applicable to other types of autonomous systems including autonomous land and air vehicles. The domain specific language and compiler of the framework enables users to easily create new and dynamic missions for the AUV. Embedded in the language are domain specific features that empower the mission writer to pick a satisfiable trade-off point in the trade-off space. For example, the quality of the sensor readings may be sacrificed in order to gain an increase in mission endurance. Furthermore, the system has been designed and tested to meet practical constraints of a mission critical system that operates in extreme environments.

During the development of the framework, a power measurement board was designed and deployed to create energy models for the glider. The measurement board

was equipped on several gliders and captured the power requirements of individual components of the vehicle. The longest sea trial was approximately one month long and was critical in the development of the energy model for the buoyancy engine that provides the propulsion for the AUV.

To enable mission programmers to effectively explore and ultimately choose acceptable trade-offs, the energy models have been incorporated into two glider simulators. The simulation infrastructures are capable of running faster-than-real-time and use environmental information to produce realistic virtual deployments. Authors are quickly provided with feedback on their trade-off decisions after running their missions using the framework.

A number of applications and challenges were explored during the development of the programming framework. These include acoustic communication, swarming and coordination, path planning, high performance computing and adaptive feature-based energy management of sensors. These challenges were researched to gain familiarity with the field and have helped the framework to mature and become a practical system. The feature-based energy management technique, for example, is available in the domain specific language and has been effectively used in sea trials to save energy while still capturing relevant data.

There are many areas of future research. A comprehensive user study outside of the pilots and scientists at Rutgers University could help enhance the framework and help it to be adopted by a broader audience. Such a study may also bring about suggestions for new domain specific features and algorithms that could be added to the framework. Again, the focal point is to allow non-experts to efficiently program AUVs in a way that makes sense to them by incorporating domain specific features. Scientists are the individuals who will consume the collected data so they will know best on what trade-offs they will need to make in order for sea trials to remain useful.

Further work on the simulation front is also needed to enable more refined and accurate mission estimations. Thus far, not all components of the software ported simulator have been made to work on commodity hardware. For example, by porting over more of the glider control system, the simulator would also become more useful in training new glider pilots. Instead of investing in a Shoebox or Pocket simulator, a complete reproduction of the glider piloting environment would be instrumental in gaining the interests of potential pilots as part of a classroom course. Even the programming framework in its current state could be used in such a way.

Potentially, with the integration of a Linux single board computer and the port of the glider's software, an alternative infrastructure to control the glider is in sight. A replacement for the glider's science infrastructure is more immanent and more easily accomplished, since a new flight controller would require more difficult drivers to be programmed. However, such a process could be staged and only portions of the control software could be handled on the SBC initially. For instance, the behavior and layered control execution could occur on the SBC while the generated command data structure could still be filled and executed by the glider's flight controller.

Further research on services on the vehicle itself, on shore, as well as their interactions is also necessary. The compiler must ensure that services that are required by programs will provide the necessary data resolution for each other. Interestingly, this may involve simulations to determine whether a service is capable of providing adequate spatiotemporal resolution that makes another dependent service, that is also providing some sort of trade-off, possible. In the case of thermocline sensor triggering that relies on temperature and depth measurements, for example, simulations in the area of operation may indicate that the water is shallow and that only a subset of the profiles are required to capture an acceptable snapshot of the water column.

With the addition of the single board computer, more advanced sensor processing

is possible. This work has provided an infrastructure that is capable of hierarchical power measurement techniques. Thus far, this mechanism has not been used to its full potential. As part of future research, investigating more advanced energy management techniques that schedule and plan sensors, actuators, and services is also of interest. Such work could involve using the power measurement infrastructure to refine the energy model for a particular glider while in flight and to perform live task profiling and scheduling

Further refinements to the energy model are also necessary and are part of future work. Because many of the vehicle's services are envisioned to make use of energy and quality of result trade-offs, an accurate energy model is of extreme importance. A new measurement infrastructure is in development that enables plugin sensor boards, allowing not just power measurement, but other sensors such as an accelerometer to be easily integrated into the glider. The new board is also more power efficient and should be capable of being deployed on regular glider flights without becoming a burden.

Finally, identifying what components and lessons of the described system can be used in other cyber-physical systems is of extreme importance and should be investigated further. Other autonomous systems have very similar constraints that include concerns for safety, practicality, resource optimizations and trade-offs, a limited energy budget, and sporadic communication. The framework was designed around these constraints so many of the approaches should be applicable to other systems like autonomous aerial drones, terrestrial vehicles and satellites. Much of the compiler infrastructure that is used to create and build services and check for the consistency of state definitions could also be retrofitted for use on other CPSs. Exploring these other avenues may in fact lead to new lessons that are also applicable to the new programming framework on the Slocum Glider.

Bibliography

A. Aho, M. Lam, R. Sethi, and J. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, second edition, 2007.

A. Alvarez, A. Caiti, and R. Onken. Evolutionary path planning for autonomous underwater vehicles in a variable ocean. *Oceanic Engineering, IEEE Journal of*, 29(2):418–429, April 2004. ISSN 0364-9059. doi: 10.1109/JOE.2004.827837.

A. Alvarez and R. Stoner. Quality assessment of pumped and un-pumped CTD sensors in Slocum gliders. Technical Report CMRE-MR-2012-003, Center for Maritime Research and Experimentation (CMRE), November, 2012.

M. Arima, N. Ichihashi, and Y. Miwa. Modelling and motion simulation of an underwater glider with independently controllable main wings. In *OCEANS 2009 - EUROPE*, pages 1–6. 2009. doi: 10.1109/OCEANSE.2009.5278267.

Woongki Baek and Trishul M. Chilimbi. Green: a framework for supporting energy-conscious programming using controlled approximation. In *Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation*, PLDI '10, pages 198–209. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0019-3. doi: 10.1145/1806596.1806620. URL `http://doi.acm.org/10.1145/1806596.1806620`.

Nilanjan Banerjee, Jacob Sorber, Mark D. Corner, Sami Rollins, and Deepak Ganesan. Triage: balancing energy and quality of service in a microserver. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, pages 152–164. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-614-1. doi: 10.1145/1247660.1247680. URL `http://doi.acm.org/10.1145/1247660.1247680`.

G Beidler, T Bean, K Merrill, M O'Rourke, and D Edwards. From language to code: Implementing auvish. In *15th International Symposium on Unmanned Untethered Submersible Technology (UUST)*. Durham, New Hampshire, August 2007.

J. Bellingham and J. Leonard. Task configuration with layered control. In *IARP Workshop on Mobile Robots for Subsea Environments*. Monterey, CA, May 1994.

Michael R. Benjamin, Henrik Schmidt, Paul M. Newman, and John J. Leonard. Nested autonomy for unmanned marine vehicles with moos-ivp. *Journal of Field Robotics*, 27(6):834–875, 2010.

P. Bhatta and N.E. Leonard. Stabilization and coordination of underwater gliders. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 2, pages 2081–2086 vol.2. 2002. ISSN 0191-2216. doi: 10.1109/CDC.2002.1184836.

R. Bleck, G. Halliwell, A. Wallcraft, S. Carroll, K. Kelly, and K. Rushing. *HYbrid Coordinate Ocean Model (HYCOM) - Users Manual - Details of the numerical code*. HYCOM, version 2.0.01 edition, March 2002. URL `http://hycom.org/attachments/063_hycom_users_manual.pdf`.

R. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

Massimo Cencini, Guglielmo Lacorata, Angelo Vulpiani, and Enrico Zambianchi. Mixing in a meandering jet: A markovian approximation. *Journal of physical oceanography*, 29(10):2578–2594, 1999.

CODAR Corporate Headquarters. CODAR. Mountain View, CA. Http://www.codar.com.

K. Cooper and L. Torczon. *Engineering a Compiler*. Morgan Kaufmann Publishers (Impring of Elsevier Science), San Francisco, CA, 2008.

E.L. Creed, J. Kerfoot, C. Mudgal, and H. Barrier. Transition of slocum electric gliders to a sustained operational system. In *OCEANS '04. MTTS/IEEE TECHNO-OCEAN '04*, volume 2, pages 828–833 Vol.2. 2004. doi: 10.1109/OCEANS.2004.1405565.

N.A. Cruz and A.C. Matos. Adaptive sampling of thermoclines with autonomous underwater vehicles. In *OCEANS 2010*, pages 1 –6. September 2010a. doi: 10.1109/OCEANS.2010.5663903.

N.A. Cruz and A.C. Matos. Reactive auv motion for thermocline tracking. In *OCEANS 2010 IEEE - Sydney*, pages 1 –6. May 2010b. doi: 10.1109/OCEANSSYD.2010.5603883.

D Davis. Automated parsing and conversion of vehicle-specific data into autonomous vehicle control language (avcl) using context-free grammars and xml data binding. In *14th International Symposium on Unmanned Untethered Submersible Technology (UUST)*. Durham, New Hampshire, August 2005.

D. Davis and D. Brutzman. The autonomous unmanned vehicle workbench: Mission planning, mission rehearsal, and mission replay tool for physics-based x3d visualization. In *14th International Symposium on Unmanned Untethered Submersible Technology (UUST)*. Durham, New Hampshire, August 2005.

R. Davis, E. Eriksen, and C. Jones. Autonomous buoyancy-driven underwater gliders. *In: Technology and Applications of Autonomous Underwater Vehicles, G. Griffiths (Ed), Taylor & Francis, London*, pages 37–58, 2002.

P.S. Dias, S.L. Fraga, R.M.F. Gomes, G.M. Goncalves, F.L. Pereira, J. Pinto, and J.B. Sousa. Neptus - a framework to support multiple vehicle operation. In *Oceans 2005 - Europe*, volume 2, pages 963 – 968 Vol. 2. 2005.

P.S. Dias, G.M. Goncalves, R.M.F. Gomes, J.B. Sousa, J. Pinto, and F.L. Pereira. Mission planning and specification in the neptus framework. In *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pages 3220 –3225. 2006. ISSN 1050-4729. doi: 10.1109/ROBOT.2006.1642192.

Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, (1):269–271, 1959.

Christiane N Duarte, Gerald R Martel, Christine Buzzell, Denise Crimmins, Rick Komerska, Sai Mupparapu, Steve Chappell, D Richard Blidberg, and Robert Nitzel. A common control language to support multiple cooperating auvs. In *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology*. 2005.

C.N. Duarte, G.R. Martel, E. Eberbach, and C. Buzzell. Talk amongst yourselves: getting multiple autonomous vehicles to cooperate. In *Autonomous Underwater Vehicles, 2004 IEEE/OES*, pages 96–101. 2004. doi: 10.1109/AUV.2004.1431199.

C.N. Duarte and B.B. Werger. Defining a common control language for multiple autonomous vehicle operation. In *OCEANS 2000 MTS/IEEE Conference and Exhibition*, volume 3, pages 1861–1867 vol.3. 2000. doi: 10.1109/OCEANS.2000.882208.

E. Eberbach, C.N. Duarte, C. Buzzell, and G. Martel. A portable language for control of multiple autonomous vehicles and distributed problem solving. In *Proc. of the 2nd Intern. Conf. on Computational Intelligence, Robotics and Autonomous Systems CIRAS*, volume 3, pages 15–18. 2003.

M. Eichhorn, C.D. Williams, R. Bachmayer, and B. de Young. A mission planning system for the AUV "SLOCUM glider" for the Newfoundland and Labrador shelf. In *OCEANS 2010 IEEE - Sydney*, pages 1–9. 2010. doi: 10.1109/OCEANSSYD.2010.5603919.

Mike Eichhorn. Optimal Path Planning for AUVs in Time-Varying Ocean Flows. In *16th Symposium on Unmanned Untethered Submersible Technology (UUST09)*. Durham NH, USA, August 2009.

Mike Eichhorn, Hans C. Woithe, and Ulrich Kremer. Parallelization of path planning algorithms for AUVs concepts, opportunities, and program-technical implementation. In *Oceans '12 MTS/IEEE Yeosu*. Yeosu, Republic of Korea, May 2012.

Charles C. Eriksen, T. James Osse, Russell D. Light, Timothy Wen, Thomas W. Lehman, Peter L. Sabin, John W. Ballard, and Andrew M. Chiodi. Seaglider: A long-range autonomous underwater vehicle for oceanographic research. In *IEEE Journal of Oceanic Engineering*, volume 26. October 2001.

M. Evans and T. Georges. Coastal ocean dynamics radar (codar): Noaa's surface current mapping system. In *OCEANS '79*, pages 379–384. 1979. doi: 10.1109/OCEANS.1979.1151297.

J.A. Farrell, Shuo Pang, and Wei Li. Chemical plume tracing via an autonomous underwater vehicle. *Oceanic Engineering, IEEE Journal of*, 30(2):428–442, 2005. ISSN 0364-9059. doi: 10.1109/JOE.2004.838066.

E. Fiorelli, N.E. Leonard, P. Bhatta, D. Paley, R. Bachmayer, and D.M. Fratantoni. Multi-AUV control and adaptive sampling in monterey bay. In *Autonomous Underwater Vehicles, 2004 IEEE/OES*, pages 134–147. 2004. doi: 10.1109/AUV.2004.1431204.

Derek A Fong and Nicole L Jones. Evaluation of auv-based adcp measurements. *Limnology and Oceanography: Methods*, 4:58–67, 2006.

L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball. The WHOI micromodem: An acoustic communications and navigation system for multiple platforms. In *IEEE Oceans Conference*. Washington, D.C., 2005.

Erann Gat, R. Peter Bonnasso, Robin Murphy, and Aaai Press. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*, pages 195–210. AAAI Press, 1998.

Fritz Gessner. The vertical distribution of phytoplankton and the thermocline. *Ecology*, 29(3):pp. 386–389, 1948. ISSN 00129658.

M.A. Godin, J.G. Bellingham, B. Kieft, and R. McEwen. Scripting language for state configured layered control of the tethys long range autonomous underwater vehicle. In *OCEANS 2010*, pages 1–7. 2010. doi: 10.1109/OCEANS.2010.5664515.

Google Inc. Google Earth. Http://earth.google.com/.

Joshua G. Graver, Ralf Bachmayer, Naomi Ehrich Leonard, and David M. Fratantoni. Underwater glider model parameter identification. In *Proceedings 13th International Symposium on Unmanned Untethered Submersible Technology*. 2003.

M. Gries, U. Hoffmann, M. Konow, and M. Riepen. SCC: A flexible architecture for many-core platform research. *Computing in Science Engineering*, 13(6):79 –83, November-December 2011. ISSN 1521-9615. doi: 10.1109/MCSE.2011.109.

S.D. Haeger. Vertical representation of ocean temperature profiles with a gradient feature model. In *OCEANS '95. MTS/IEEE. Challenges of Our Changing Global Environment. Conference Proceedings.*, volume 1, pages 579–585 vol.1. October 1995. doi: 10.1109/OCEANS.1995.526820.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009. ISSN 1931-0145. doi: 10.1145/1656274.1656278. URL `http://doi.acm.org/10.1145/1656274.1656278`.

J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers (Impring of Elsevier Science), San Francisco, CA, fourth edition, 2007.

Henry Hoffmann, Stelios Sidiroglou, Michael Carbin, Sasa Misailovic, Anant Agarwal, and Martin Rinard. Dynamic knobs for responsive power-aware computing. In *Proceedings of the sixteenth international conference on Architectural support for*

programming languages and operating systems, ASPLOS XVI, pages 199–212. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0266-1. doi: 10.1145/1950365.1950390. URL `http://doi.acm.org/10.1145/1950365.1950390`.

J. Howard, S. Dighe, S.R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V.K. De, and R. Van Der Wijngaart. A 48-core ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *Solid-State Circuits, IEEE Journal of*, 46(1):173 –183, January 2011. ISSN 0018-9200. doi: 10.1109/JSSC.2010.2079450.

Hydroid, LLC. Remus AUV. Http//www.hydroidnc.com/.

Rick J Komerska and Steven G Chappell. Auv common control language (ccl)–a proposed standard language and framework for auv monitoring & control layer 1–ccl vocabulary and message set specification. 2007a.

Rick J Komerska and Steven G Chappell. Auv common control language (ccl)–a proposed standard language and framework for auv monitoring & layer 2-ccl support library. 2007b.

C. Kunz, C. Murphy, R. Camilli, H. Singh, J. Bailey, R. Eustice, M. Jakuba, K. Nakamura, C. Roman, T. Sato, R.A. Sohn, and C. Willis. Deep sea underwater robotic exploration in the ice-covered arctic ocean with auvs. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3654–3660. 2008a. doi: 10.1109/IROS.2008.4651097.

Clayton Kunz, Chris Murphy, Richard Camilli, Hanumant Singh, John Bailey, Ryan Eustice, Michael Jakuba, Ko ichi Nakamura, Chris Roman, Taichi Sato, Robert A. Sohn, and Claire Willis. Deep sea underwater robotic exploration in the ice-covered artic ocean with auvs. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*. September 2008b.

Andreas Lachenmann, Pedro José Marrón, Daniel Minder, and Kurt Rothermel. Meeting lifetime goals with energy levels. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 131–144. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-763-6. doi: 10.1145/1322263.1322277. URL `http://doi.acm.org/10.1145/1322263.1322277`.

John J. Leonard, Andrew A. Bennett, Christopher M. Smith, Hans Jacob, and S. Feder. Autonomous underwater vehicle navigation. In *MIT Marine Robotics Laboratory Technical Memorandum*. 1998.

N. Mahmoudian, J. Geisbert, and C. Woolsey. Approximate analytical turning conditions for underwater gliders: Implications for motion control and path planning. *Oceanic Engineering, IEEE Journal of*, 35(1):131–143, 2010. ISSN 0364-9059. doi: 10.1109/JOE.2009.2039655.

C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999. ISBN 9780262133609. URL `http://books.google.com/books?id=YiFDxbEX3SUC`.

T.G. Mattson, R.F. Van der Wijngaart, M. Riepen, T. Lehnig, P. Brett, W. Haas, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe. The 48-core SCC processor: the programmer's view. In *High Performance Computing, Networking, Storage and Analysis (SC), 2010 International Conference for*, pages 1 –11. November 2010. doi: 10.1109/SC.2010.53.

S.S. Mupparapu, S.G. Chappell, R.J. Komerska, D.R. Blidberg, R. Nitzel, C. Benton, D.O. Popa, and A.C. Sanderson. Autonomous systems monitoring and control (asmac) - an auv fleet controller. In *Autonomous Underwater Vehicles, 2004 IEEE/OES*, pages 119–126. 2004. doi: 10.1109/AUV.2004.1431202.

National Oceanic and Atmospheric Administration. National Geophysical Data Center. Http://www.ngdc.noaa.gov/.

Brian D. Noble, M. Satyanarayanan, Dushyanth Narayanan, James Eric Tilton, Jason Flinn, and Kevin R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, SOSP '97, pages 276–287. ACM, New York, NY, USA, 1997. ISBN 0-89791-916-5. doi: 10.1145/268998.266708. URL `http://doi.acm.org.proxy.libraries.rutgers.edu/10.1145/268998.266708`.

Ariel Orda and Raphael Rom. Shortest-Path and Minimum-Delay Algorithms in Networks with Time-Dependent Edge-Length. *Journal of the Association for Computing Machinery*, 37(3):607–625, 1990.

F. Paley, D.A.and Zhang and N.E. Leonard. Coordinated control of an underwater glider fleet in an adaptave ocean sampling field experiment in monterey bay. In *Control Systems Technology, IEEE Transactions on*, pages 735–744. 2008. doi: 10.1109/TCST.2007.912238.

Persistor Instruments Inc. CF1 computer system. Marstons Mills, MA. Http://www.persistor.com.

S. Petillo, A. Balasuriya, and H. Schmidt. Autonomous adaptive environmental assessment and feature tracking via autonomous underwater vehicles. In *OCEANS 2010 IEEE - Sydney*, pages 1 –9. May 2010. doi: 10.1109/OCEANSSYD.2010.5603513.

A. Rajala, M. O'Rourke, and D.B. Edwards. Auvish: An application-based language for cooperating auvs. In *OCEANS 2006*, pages 1–6. 2006. doi: 10.1109/OCEANS.2006.307128.

Dushyant Rao and Stefan B Williams. Large-scale path planning for underwater gliders in ocean currents. In *Australasian Conference on Robotics and Automation (ACRA)*. Sydney, Australia, December 2009.

R. Rew and G. Davis. Netcdf: an interface for scientific data access. *Computer Graphics and Applications, IEEE*, 10(4):76–82, 1990. ISSN 0272-1716. doi: 10.1109/38.56302.

Rutgers University. The scarlet knight's trans-atlantic challenge. Http://rucool.marine.rutgers.edu/atlantic/.

Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. Enerj: approximate data types for safe and general low-power computation. In *Proceedings of the 32nd ACM SIGPLAN conference on Programming language design and implementation*, PLDI '11, pages 164–174. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0663-8. doi: 10.1145/1993498.1993518. URL `http://doi.acm.org/10.1145/1993498.1993518`.

Philip J. Schneider and David H. Eberly. *Geometric Tools for 3D Graphics*. Morgan Kaufmann Publishers, San Francisco, 2003. ISBN 1-55860-594-0.

T. Schneider and H. Schmidt. The dynamic compact control language: A compact marshalling scheme for acoustic communications. In *OCEANS 2010 IEEE - Sydney*, pages 1–10. 2010. doi: 10.1109/OCEANSSYD.2010.5603520.

O. Schofield, L. Creed, J. Graver, C. Haldeman, J. Kerfoot, H. Roarty, C. Jonees, D. Webb, and S. Glenn. Slocum gliders: Robust and ready. *Journal of Field Robotics, Wiley Periodicals, Inc.*, 24(6):473–485, 2007.

Bryan Schulz, Robert Hughes, Edward Matson, Ryan Moody, and Brett Hobson. The theseus autonomous underwater vehicle. a canadian success story. In *Proceedings of MTS/IEEE OCEANS Ő7*, volume 2. October 1997.

Bryan Schulz, Robert Hughes, Edward Matson, Ryan Moody, and Brett Hobson. The development of a free-swimming uuv for mine neutralization. In *Proceedings of MTS/IEEE OCEANS 2005*, volume 2. September 2005.

A.F. Shchepetkin and J.C. McWilliams. The regional oceanic modeling system (ROMS): a split-explicit, free-surface, topography-following-coordinate oceanic model. volume 9, pages 347–404. 2005.

Jeff Sherman, Russ E. Davis, W.B. Owens, and J. Valdes. The autonomous underwater glider šspray:. In *IEEE Journal of Oceanic Engineering*, volume 26. October 2001.

Stelios Sidiroglou-Douskos, Sasa Misailovic, Henry Hoffmann, and Martin Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, ESEC/FSE '11, pages 124–134. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0443-6. doi: 10.1145/2025113.2025133. URL `http://doi.acm.org/10.1145/2025113.2025133`.

S. Singh, M. Grund, B. Bingham, R. Eustice, H. Singh, and L. Freitag. Underwater acoustic navigation with the WHOI micro-modem. In *OCEANS 2006*, pages 1–4. September 2006. doi: 10.1109/OCEANS.2006.306853.

J. Sorber, N. Banerjee, M. D. Corner, and S. Rollins. Turducken: Hierarchical power management for mobile devices. In *International Conference on Mobile Systems, Applications, and Services (Mobisys), Seattle, WA, June 2005*, pages 261–274. 2005.

Jacob Sorber, Alexander Kostadinov, Matthew Garber, Matthew Brennan, Mark D. Corner, and Emery D. Berger. Eon: a language and runtime system for perpetual systems. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 161–174. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-763-6. doi: 10.1145/1322263.1322279. URL `http://doi.acm.org/10.1145/1322263.1322279`.

Noah Spurrier. Pexpect. http://www.noah.org/wiki/pexpect, 2012.

G Stante, M Nahon, and CD Williams. Simulation of an underwater glider. 2007.

R.P. Stokey. A compact control language for autonomous underwater vehicles. April 2005. URL `http://acomms.whoi.edu/publications/`.

R.P. Stokey, L.E. Freitag, and M.D. Grund. A compact control language for auv acoustic communication. In *Oceans 2005 - Europe*, volume 2, pages 1133–1137 Vol. 2. 2005. doi: 10.1109/OCEANSE.2005.1513217.

Technologic Systems. Technologic systems. Foutain Hills, AZ. Http://www.embeddedarm.com.

Teledyne RD Instruments. DVL Explorer. Poway, CA. Http//www.rdinstruments.com/explorer.aspx.

Teledyne Webb Research. Slocum glider, http//www.webbresearch.com/slocum.htm. Falmouth, MA.

Isaías A. Comprés Ureña, Michael Riepen, and Michael Konow. Rckmpi - lightweight MPI implementation for Intel's single-chip cloud computer (SCC). In *Proceedings of the 18th European MPI Users' Group conference on Recent advances in the message passing interface*, EuroMPI'11, pages 208–217. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-24448-3.

Ding Wang, Pierre F.J. Lermusiaux, Patrick J. Haley, Donald Eickstedt, Wayne G. Leslie, and Henrik Schmidt. Acoustically focused adaptive sampling and onboard routing for marine rapid environmental assessment. *Journal of Marine Systems*, 78(Supplement 1):S393 – S407, 2009. ISSN 0924-7963. doi: DOI: 10.1016/j.jmarsys.2009.01.037. Coastal Processes: Challenges for Monitoring and Prediction.

L. Whitcomb, D. Yoerger, and H. Singh. Advances in doppler-based navigation of underwater robotic vehicles. In *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*, volume 1, pages 399–406 vol.1. 1999. ISSN 1050-4729. doi: 10.1109/ROBOT.1999.770011.

Hans C. Woithe, William Brozas, Christian Wills, Bharath Pichai, Ulrich Kremer, Mike Eichhorn, and Michael Riepen. Enabling computation intensive applications in battery-operated cyber-physical systems. In *MARC Symposium*, pages 34–39. July 2012.

Hans C. Woithe, Ilya Chigirev, David Aragon, Murium Iqbal, Yuriy Shames, Scott Glenn, Oscar Schofield, Ivan Seskar, and Ulrich Kremer. Slocum glider energy measurement and simulation infrastructure. In *OCEANS 2010 IEEE - Sydney*, pages 1–8. May 2010. doi: 10.1109/OCEANSSYD.2010.5603909.

H.C. Woithe, D. Boehm, and U. Kremer. Improving slocum glider dead reckoning using a doppler velocity log. In *OCEANS 2011*, pages 1–5. 2011.

H.C. Woithe, Mike Eichhorn, Oscar Schofield, , and U. Kremer. Assessing automated and human path planning for the slocum glider. In *18th International Symposium on Unmanned Untethered Submersible Technology (UUST)*. Portsmouth, New Hampshire, October 2013.

H.C. Woithe and U. Kremer. A programming architecture for smart autonomous underwater vehicles. pages 4433–4438. October 2009. doi: 10.1109/IROS.2009.5354098.

H.C. Woithe and U. Kremer. An interactive slocum glider flight simulator. In *OCEANS 2010*, pages 1–10. 2010. doi: 10.1109/OCEANS.2010.5664299.

H.C. Woithe and U. Kremer. A lightweight scripting engine for the slocum glider. In *OCEANS 2011*, pages 1–7. 2011a.

H.C. Woithe and U. Kremer. Using slocum gliders for coordinated spatial sampling. In *OCEANS, 2011 IEEE - Spain*, pages 1–8. 2011b. doi: 10.1109/Oceans-Spain.2011.6003468.

Woods Hole Oceanographic Institution. Whoi acoustic communications: Micro-modem. Woods Hole, MA. Http://acomms.whoi.edu/umodem/.

Yanwu Zhang, J.G. Bellingham, M. Godin, J.P. Ryan, R.S. McEwen, B. Kieft, B. Hobson, and T. Hoover. Thermocline tracking based on peak-gradient detection by an autonomous underwater vehicle. In *OCEANS 2010*, pages 1 –4. September 2010. doi: 10.1109/OCEANS.2010.5664545.

Yanwu Zhang, J.G. Bellingham, M.A. Godin, and J.P. Ryan. Using an autonomous underwater vehicle to track the thermocline based on peak-gradient detection. *Oceanic Engineering, IEEE Journal of*, 37(3), July 2012. ISSN 0364-9059. doi: 10.1109/JOE.2012.2192340.

Yanwu Zhang, R.S. McEwen, J.P. Ryan, and J.G. Bellingham. An adaptive triggering method for capturing peak samples in a thin phytoplankton layer by an autonomous underwater vehicle. In *OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*. October 2009.

X. Zheng. Layered control of a practical auv. In *Autonomous Underwater Vehicle Technology, 1992. AUV '92., Proceedings of the 1992 Symposium on*, pages 142–147. 1992. doi: 10.1109/AUV.1992.225181.