Trilobite$_G$: A programming architecture for autonomous underwater vehicles

Hans Christian Woithe Ulrich Kremer
Rutgers University
{hcwoithe,uli}@cs.rutgers.edu

Abstract

Programming autonomous systems can be challenging because many programming decisions must be made in real time and under stressful conditions, such as on a battle field, during a short communication window, or during a storm at sea. As such, new programming designs are needed to reflect these specific and extreme challenges.

Trilobite$_G$ is a programming architecture for buoyancy-driven autonomous underwater vehicles (AUVs), called gliders. Gliders are designed to spend weeks to months in the ocean, where they operate fully autonomously while submerged and can only communicate via satellite during their limited time at the surface. Based on the experience gained from a seven year long collaboration with two oceanographic institutes, the Trilobite$_G$ architecture has been developed with the main goal of enabling users to run more effective missions. The Trilobite$_G$ programming environment consists of a domain-specific language called ALGAE, a lower level service layer, and a set of real-time and faster-than-real-time simulators. The system has been used to program novel and robust glider behaviors, as well as to find software problems that otherwise may have remained undetected, with potentially catastrophic results. We believe that Trilobite$_G$ can serve as a blueprint for other autonomous systems as well, and that Trilobite$_G$ will motivate and enable a broader scientific community to work on extreme, real-world problems by using the simulation infrastructure.

Categories and Subject Descriptors D.2.1 [*Requirements/Specifications*]: Languages; D.2.11 [*Software Architectures*]: Domain-specific architectures, Languages; D.3.2 [*Language Classifications*]: Multiparadigm languages, Specialized application languages

1. Introduction

Programming is typically thought of as an activity done by a person in a controlled environment such as an air-conditioned office space, or perhaps in the relaxed environment of a cafe. We typically do not imagine a programmer trying to reprogram or debug a program under extreme conditions, such as on the battle field, exposed to the elements during a disaster response operation, or in a boat that rolls and pitches while deploying a scientific instrument. Programmers in situations like these also experience hardships that may result in additional stressors, like sleep deprivation due to the length of a particular mission. We cannot expect programmers to maintain a high level of performance under these conditions. In addition, the system that needs to be retasked or debugged is itself exposed to an extreme environment and requires attention in real time. Once the retasking or bug fix is done, the system will have to work fully autonomously without the option of any further interactions, at least for some period of time. This means that any mistakes may have severe consequences, including the total loss of the system.

Autonomous underwater vehicles (AUVs), terrestrial robots, aerial drones, space probes, and satellites are examples of autonomous systems that need to be programmed in extreme conditions. These systems have a common set of characteristics independent of their particular application domain that should be reflected in the design and implementation of the programming architecture if it wants to be successful, both in terms of functionality as well as acceptance in the user community. These characteristics include: (a) the need for functionality and resource tradeoffs; (b) teams of users with varying expertise; and (c) real-time decision making. As such, missions have to be designed with resource constraints in mind. Users may also have to collaborate to achieve complex mission goals due to the complexity of the programming and operational tasks. Furthermore, autonomous systems may be deployed in hostile and dangerous environments where decisions are often time and mission critical.

In this paper, we discuss the design, implementation, and preliminary evaluation of Trilobite$_G$[1], a programming architecture for buoyancy-driven autonomous underwater vehicles, called gliders. Gliders are designed to spend weeks to months in the ocean, operating fully autonomously while submerged, and with interactions only possible via satellite communication during their time at the surface. There are three main reasons why we picked the domain of AUVs as our representative example for an extreme programming environment. First, developing and managing AUV deployments have all the characteristics of an extreme programming environment. Therefore solutions in this domain will have direct impact on other extreme domains as well. Second, AUVs are an important application domain. AUVs have revolutionized the way marine scientists and oceanographers collect data and gain knowledge about the world's oceans. However, these autonomous systems are still far from being tools that scientists can use "off the shelf." Trilobite$_G$ will enable oceanographers to utilize their AUVs more effectively. Third, we have substantial knowledge and expertise in the domain of AUVs due to close interactions and collaborations with oceanog-

LCTES '15, June 18–19, 2015, Portland, OR, USA.

http://dx.doi.org/10.1145/2670529.2754971

[1] The trilobites were among the most successful and resilient of all early animals, roaming the oceans for over 270 million years. The 'G' identifies the glider version.

raphers, marine scientists, marine engineers, and AUV pilots, particularly at two major oceanographic research institutes: the Monterey Bay Aquarium Research Institute (MBARI) and the Institute for Marine and Coastal Science at Rutgers (IMCS). The main contributions of this work are:

1. The description of the design and implementation of Trilobite$_G$, a realistic programming architecture for the Slocum Glider AUV. Over 35 oceanographic research groups operate the Slocum Glider world wide, with the U.S. Navy operating 150 gliders alone. The Trilobite$_G$ architecture includes a high-level, domain-specific language ALGAE (AUV Language for Greater Adaptability and Energy Optimization) and a service oriented programming layer at the lower level, together with their compiler, runtime system, and simulators.
2. The evaluation of Trilobite$_G$ through simulation and field deployments off the coast of New Jersey. The experimental results indicate the expressiveness and effectiveness of Trilobite$_G$ and its components.
3. The development of a web interface for the Trilobite$_G$ simulation environment, which will allow oceanographers and other scientists to explore different aspects of glider mission planning and implementation. We hope that this will spark interest in the broader research community in developing new algorithms for glider operations and mission designs for AUVs, as well as other extreme programming domains.

To the best or our knowledge, Trilobite$_G$ is the first programming architecture for autonomous systems where dealing with uncertainty and extreme conditions is the key design principle across the entire programming architecture. It allows performance and energy tradeoffs to be expressed and validated through simulation and on-board measurements. By making the simulation environment public, others will be able to conduct research in this important application domain without the significant capital and operation costs of actual AUVs.

The paper is organized as follows. After a discussion of related work, we will introduce the Slocum Glider which is the main target of Trilobite$_G$. The Trilobite$_G$ programming architecture is discussed in Section 4, followed by a description of its simulation environment. Section 6 presents experimental results. The paper concludes with a summary and future work.

2. Related Work

Historically, robot manufacturers provided proprietary approaches to programming robots, i.e., one of a kind solutions for programming their particular robots and their specialized applications. More recently, there have been efforts to standardize programming environments for a larger class of robots, with some focused on education. For example, RobotC [5] is an extension of the C programming language that defines interfaces to deal with motors, sensors, and relays as they are used in robotic systems. It is mainly designed as a tool to introduce students to robot programming. The Robot Operating System (ROS) [29] is a set of software libraries and tools for developing software for a wide variety of robotic platforms. Finally, National Instruments LabView Robotics [23] is a set of tools to provide a standardized hardware and software development platform for a wide range of autonomous systems and robots.

The above systems are rather general and mostly provide lower level abstractions for robotic system sensing and actuation. In order to address the particular needs of marine robots, researchers have developed specialized programming systems to support different aspects of AUVs and their operation.

MOOS-IvP is a set of open source tools that provide autonomy for unmanned marine vehicles [4]. 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. 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 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 [8, 9]. 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 an XML-based mission format [10, 11, 20].

The Common Control Language (CCL) has been developed to investigate a standard for communicating between AUVs and surface vehicles as well as human operators [13–16, 24, 27]. To support groups of heterogeneous vehicles, a set of generic or basic behaviors are defined that are common among all AUVs. These generic behaviors are categorized into nine broad classes. The focal point thus far has been on the categories of maneuvering, navigating, communicating, configuring, executing, and monitoring. The behaviors are used as basic building blocks for AUV interaction and for mission files.

Although specialized for AUV operations, the systems described are still rather general. They define low-level and intermediate infrastructures that allow interactions between different sensing and actuation activities, and support the collaboration of AUVs with a wide range of capabilities. One of their main objectives is the standardization and software interoperability across different platforms. The simulation support for these systems often do not exist, or are rather limited.

In a world where energy efficiency and safety is crucial for vehicle operations and survival, these systems address too many aspects (e.g., sensing, communication, and actuation) which makes them too complex and hard to use for most oceanographers. The Trilobite$_G$ design is based on the philosophy that simplicity and energy awareness will lead to a more robust and easy to use programming system. Simulation is also a key component used to build trust into the infrastructure. Simulation allows for a more efficient operation by enabling oceanographers to make better tradeoff decisions while designing a mission and during a mission. In addition, there needs to be a separate support for the domain user (oceanographer) and the services and utilities provider (pilot and vehicle engineer). Both groups have different views of a mission with different expertise that should be supported separately in order to build an overall robust and effective system.

3. Slocum Glider

The Slocum Electric Glider shown in Figure 1 is a buoyancy-driven AUV developed and produced by Teledyne Webb Research [37]. Other buoyancy driven vehicles include the Spray glider, [35] developed by the Scripps Institute of Oceanography at UC San Diego, and University of Washington's Seaglider [18]. Both vehicles are commercially available through Bluefin Robotics and Kongsberg Underwater Technology, respectively. More recent AUV designs

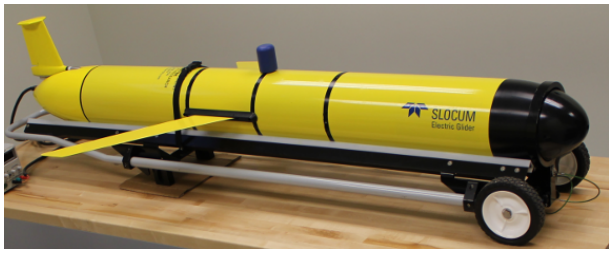


Figure 1. One of our Slocum Gliders equipped with a double payload bay. The glider is 180 cm long and weighs around 70 kg.

have both a buoyancy engine and a propeller, such as MBARI's Tethys system [21] and a new version of the Slocum Glider. These gliders have a foldable auxiliary propeller [6] that allow these hybrid AUVs to operate in glider (cruise) and propeller driven modes.

A stock Slocum Glider contains two 16 MHz computing platforms [30], one for flight control and the other for the collection of scientific data. As part of the ALGAE programming architecture, the capabilities of the AUV have been extended with the integration of a Linux single board computer (AVBot). 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.

The front of the Slocum Glider contains a buoyancy engine which moves a piston to change its displacement of water, allowing for vertical motion in the water column. The pitch of the glider can be adjusted by moving an internal battery pack, thereby changing its center of gravity. The AUV's wings allow it to glide forward through water to produce a saw-toothed flight profile inflecting near the surface and at deeper depths. Using a rudder and GPS, the vehicle is able to navigate and collect data samples using onboard sensors. Satellite and radio communications are used at the surface to transmit data and alter, if necessary, the AUV's mission [33]. Gliders can be equipped with acoustic underwater communication modems, but their practical range is limited (around 5 km).

The Slocum Glider, although slower than a propeller driven vehicle with an approximate speed of 35 cm/sec, has the advantage of having lower overall power requirements. The buoyancy engine is required only during inflection points, which may be as shallow as a few meters below the surface or as deep as 200 m for coastal gliders, and up to 1 km for the deep water version. This produces prolonged flights typically lasting four to six weeks [33] as opposed to hours for propeller-driven vehicles.

Depending on the battery type, sensing, computing, and communication activities, glider missions can last substantially longer. In 2009, a Slocum Glider flew from New Jersey to Spain in 221 days, making it the first robot to cross the Atlantic Ocean [19]. This mission also demonstrated the danger of glider operations. An attempt to cross the Atlantic a year earlier ended when the glider was lost at sea after only few months into the mission.

4. The Trilobite$_G$ Programming Architecture

Trilobite$_G$ is a layered programming architecture for AUVs with separate support for the domain user (oceanographer and pilot), and the services and utilities provider (vehicle engineer). Both groups have different views of a mission with different expertise that should be supported separately in order to build an overall robust and effective system. Since the safety of the vehicle is of paramount concern, and since the real-time decision making capabilities of the oceanographer and pilots may be somewhat compromised, the programming abstractions at the higher level have to be simple and geared towards avoiding unrecoverable mistakes.

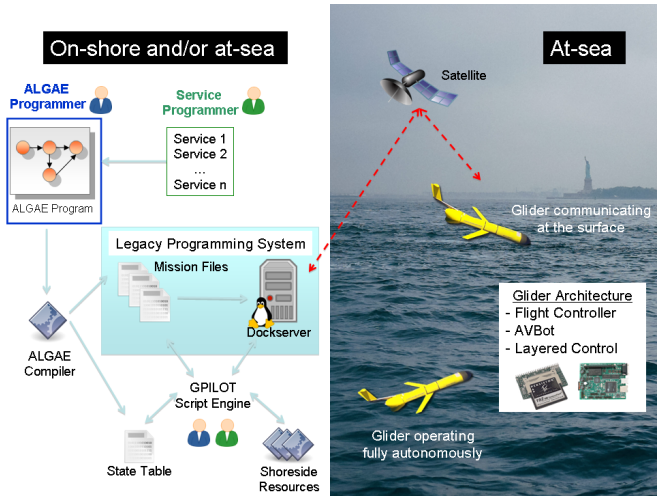


Figure 2. Basic Trilobite$_G$ programming architecture and operational overview. Simulators are not shown.

The programming abstractions in the higher level ALGAE language follow the KISS (keep it simple, stupid) principle[2]. Where possible, the complexity related to vehicle safety is pushed to the lower level where sets of well-tested, predefined behaviors are implemented.

Simulation has an important role in the layered Trilobite$_G$ programming architecture. Pilots can use simulation to make better tradeoff decisions both before and during deployment. Simulation has to take all aspects of autonomous system behavior into consideration, including the health of the system itself, the sensing and actuation activities, and resource consumption (particularly battery energy).

Finally, we have observed specific habits and operational procedures in the oceanographers and pilots we have interacted with. Like airline pilots and astronauts, they follow specific checklists and procedures to ensure a safe and successful mission. In order to make Trilobite$_G$ acceptable and trustworthy within the AUV community, we incorporated as many existing safety features and procedures as possible into the programming architecture.

4.1 Overview

Figure 2 shows the basic Trilobite$_G$ programming infrastructure and the operational framework for the target glider system. Different on-shore and at-sea architecture components have to work together in order to make vehicle operations safe and effective. To make full use of all of Trilobite$_G$'s components, a modified glider must be used. The stock/legacy glider has a flight control computer that determines sensing and actuation activities every four seconds through a layered control behavior stack. Our modified gliders are equipped with energy efficient general purpose programming platforms called AVBot. In addition, they have been retrofitted with the necessary hardware and software to allow glider activities to be monitored and controlled by programs running on AVBot. In other words, the target computing platform consists of a lower level flight controller, primarily used to run basic vehicle specific safety software, and a general purpose system that executes higher level mission specific tasks.

[2] The KISS principle has been attributed to Kelly Johnson, a former lead engineer at the Lockheed Skunk Works. One of Johnson's goal was to enable a team of average mechanics to repair and maintain his aircrafts in the field under combat conditions with only a small set of tools.

In a typical operational scenario, missions are developed on-shore and implemented as mission files (programs) that are dynamically downloaded onto the glider during a deployment. At sea, communication with a glider is only possible through a satellite link while the AUV is at the surface, because radio waves can only penetrate a few meters into the water. Trilobite$_G$ makes use of the Dockserver which allows for communication between the glider and other remote on-shore or at-sea resources. Through the satellite link, sensing data and glider health information can be uploaded to the Dockserver, and new mission files and parameters can be downloaded to the vehicle. Sensing and status data are used by oceanographers and vehicle pilots to monitor and direct the mission in real time. In addition, an automatic pilot (GPILOT) can help monitor and guide missions, often performing regular and sometimes tedious glider and mission control tasks. This frees oceanographers and pilots to concentrate on mission critical decisions. If necessary, GPILOT is capable of running a deployment fully automatically.

In the legacy system, programmers write mission files directly, which can be a rather error prone task. In the Trilobite$_G$ infrastructure, mission files are generated by the ALGAE compiler. ALGAE programs are written by oceanographers using services developed, tested, and implemented by vehicle engineers. The ALGAE compiler detects and reports inconsistencies in ALGAE programs. The compiler generated mission files contain energy optimizations that exploit the architecture on the glider. For example, if all the required data processing and computation activities can be done on the flight controller hardware, the AVBot is disabled, resulting in significant energy savings. Service implementations may use onboard and remote resources, which may be selected on demand if redundant implementations have been specified. For example, a path planning service may be performed on a remote server with greater computational capabilities and additional global information (e.g., ocean models and current predictions [22, 34]). Alternatively, the computation can be performed on board if communication is not possible or not desired. These are tradeoff decisions that the ALGAE programmer can make based on the alternatives that the service programmer has provided.

The overall programming process is divided into high-level (ALGAE) and low-level programming tasks. This architectural design assumes that most changes to a mission and its goals are done by the oceanographers while monitoring a mission from a land or sea based mission control center. Mission definitions and mission changing decisions are made at the higher programming level. ALGAE programs define a mission as a set of states, with each state providing a desired system behavior. State transitions indicate changes in overall system behavior based on a set of user-specified and service-specified events. ALGAE state transitions are only possible while the glider is at the surface. Services are assumed to be thoroughly tested by vehicle engineers before being released for use by the ALGAE application programmers. Services can be rather complex and have access to all computing, sensing, communication, and data processing resources on the vehicle or provided by any on-shore site. Services, such as thermocline tracking (see Figure 6), may change the glider behavior while it is submerged and in fully autonomous mode where no user intervention is possible. As a key design decision in our system, ALGAE users cannot directly write programs that change glider behavior while the AUV is diving. However, such changes can be specified indirectly by using the pre-defined services. This can be compared to a structured programming approach where programmers are prohibited from explicitly using GOTOs [12]. As mentioned, ALGAE programmers can change the glider's behavior only when the vehicle is at the surface, and only through state transitions that can be partially verified by the compiler. State transitions are implemented by GPILOT, which can transparently upload new mission files. Although Figure 2 depicts the overall architecture of Trilobite$_G$, it does not show the significant simulation infrastructures that support the programming and mission design process at the ALGAE and service levels.

4.2 ALGAE: First-tier Application Level

An ALGAE program specifies a finite state machine with n states where each state describes a particular continuous sensing and glider flight behavior. Events can trigger state transitions while the glider is at the surface, i.e., events are evaluated each time the glider resurfaces. For example, a mission may consist of several phases, with each phase mapping to a separate ALGAE state. A glider may be tasked to fly to a particular target area. Once the target area has been reached, additional sensors are activated and the target area is searched following a specified search flight pattern (e.g., a lawn-mower pattern [26]). Once the battery energy has fallen below a safety threshold, the glider aborts the search and flies to a predefined recovery point. For this example mission, the programmer would specify an event that defines the arrival in the target area and an event that defines a low level of remaining battery energy.

We have chosen a finite state programming model since it is simple to conceptualize and matches the current practice of AUV flight operations in both marine science research institutes we have collaborated with. Most of the challenges with respect to sensing, computing, and actuation management are encompassed within an individual state. Each state has to specify *where* the AUV is planning to go, *how* it should fly there, *what* it is supposed to do during the flight, and *when* should a transition to another state occur. Figure 3 illustrates the different specifications using a simple, single state mission as an example.

The `route` statement in Figure 3(a) instructs the vehicle to fly to a particular waypoint, i.e., a particular location. ALGAE supports absolute (GPS) and relative location specifications, for instance, relative to the original deployment location, or a particular flight direction (north, south, ...). The graph on the right side of the figure shows a simulated flight path with the specified waypoint off the coast of New Jersey.

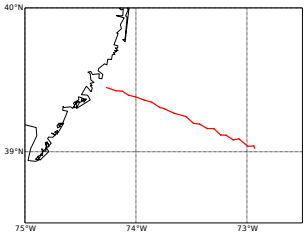
The `profile` statement in Figure 3(b) indicates the desired climb and dive angles, and the depths of the inflection points. In the example, the glider is tasked to fly between 5 m and 50 m at dive and climb angles of 26 degrees. The graph shows the simulated flight path for two dive segments, each lasting approximately one hour (3600 s).

Sensors and their data acquisition characteristics are listed in the `sensor` statement as shown in Figure 3(c). In the example, a Conductivity, Temperature, and Depth (CTD) sensor acquires data continuously while an Environmental Characterization Optics (ECO-Puck) sensor is activated only while the glider passes through a thermocline, which is a layer of water where warm surface water meets cooler deep-sea water. This behavior is shown in the graph for the two simulated flight segments. ECO-Puck activations are shown as red dots on the flight path. In this example, the thermocline is assumed to be at a depth between 20 m and 25 m. The CTD is always on, so its activation is not explicitly shown in the graph. `Thermocline` is a service that has been defined by a second-tier expert programmer and will be discussed in more detail in Section 4.3.

In Figure 3(d), the `surface` command specifies when the glider has to resurface, in seconds. State transitions can only occur when the glider is at the surface, and only to the states listed in the `events` construct. The example mission will surface every hour until the destination waypoint is reached.

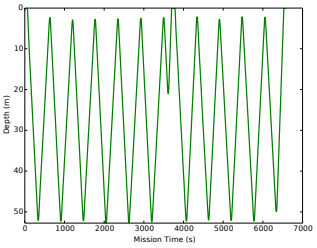
ALGAE programs are simple, yet expressive. They enable oceanographers to reason about their missions, and adapt mission

```
state: state_sensing
begin
  route: gotowaypoint(3927.0, -7415.0)
  profile: yo(5, 0.454, 50, -0.454)
  surface: interval : 3600
  sensors: ctd: interval 4 always,
           bb2flsv4: interval 0 thermocline
  events:
    case interval state_sensing,
    case waypoint state_exit
end
```



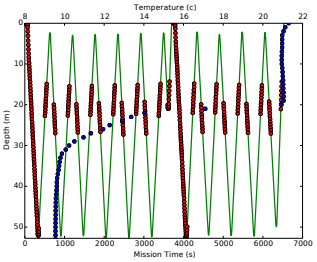
(a) Where to go?

```
state: state_sensing
begin
  route: gotowaypoint(3927.0, -7415.0)
  profile: yo(5, 0.454, 50, -0.454)
  surface: interval : 3600
  sensors: ctd: interval 4 always,
           bb2flsv4: interval 0 thermocline
  events:
    case interval state_sensing,
    case waypoint state_exit
end
```



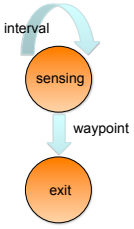
(b) How to go?

```
state: state_sensing
begin
  route: gotowaypoint(3927.0, -7415.0)
  profile: yo(5, 0.454, 50, -0.454)
  surface: interval : 3600
  sensors: ctd: interval 4 always,
           bb2flsv4: interval 0 thermocline
  events:
    case interval state_sensing,
    case waypoint state_exit
end
```



(c) What to do?

```
state: state_sensing
begin
  route: gotowaypoint(3927.0, -7415.0)
  profile: yo(5, 0.454, 50, -0.454)
  surface: interval : 3600
  sensors: ctd: interval 4 always,
           bb2flsv4: interval 0 thermocline
  events:
    case interval state_sensing,
    case waypoint state_exit
end
```



(d) When to change behaviors?

Figure 3. ALGAE programming abstractions illustrated using a single, simple example mission.

behaviors quickly and safely when needed. Modifying the data acquisition characteristic of a sensor, altering the flight pattern, or changing the frequency of surfacing may be accomplished with a single line program modification, often enabled by the use of particular services. Service definitions are discussed in the following section. In addition, the ALGAE programming model allows compile-time detection of inconsistencies in sensing and flight specifications. For instance, if a state requires continuous temperature readings, but the glider is equipped with a flow-through CTD sensor that does not reliably collect readings during inflections, the compiler would warn the operator to use a more reliable pumped CTD sensor instead [2]. Another example is the specification of different services that have conflicting sensor requirements.

4.3 Second-tier Service Level

The second-tier service level is implemented on top of the existing hardware and software infrastructure of the target AUV. In the case of the Slocum Glider, the software infrastructure provided by the manufacturer consists of a layered control system executing on the flight controller as shown in Figure 4. Every four seconds, sensing and actuation activities are selected through these behaviors, where higher priority behaviors can override activities selected by lower priority behaviors. This programming process has been shown to

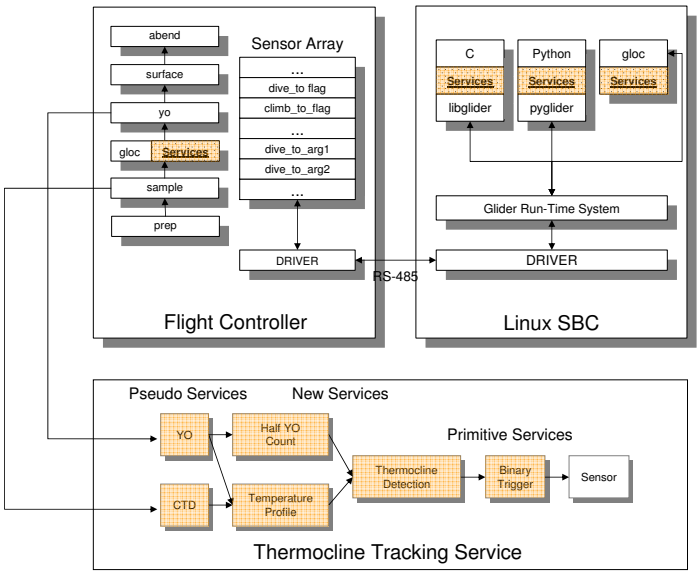


Figure 4. Onboard target system for second tier service programming with a new service definition for thermocline detection. Areas highlighted in orange indicate the location of services within the architecture.

be rather error prone for specifying glider missions, which was a key motivation behind the development of Trilobite$_G$ [40]. However, many-year efforts have gone into the design, implementation and testing of high priority safety features, which ensure the survival and recovery of the glider under basic failure and exception conditions. Such conditions include motor malfunctions, water in the hull, failure of internal communication links, and dangerously low battery. Typically, the detection of such a condition prompts the glider to perform an emergency surfacing (abend), which may include the ejection of a weight in order to increase its buoyancy. Instead of reimplementing the entire control structure with its tested and stable safety behaviors, the second-tier service level of Trilobite$_G$ is part of the layered control system executing at a lower priority than these basic glider safety behaviors. As it turned out, this design decision was crucial to gain the trust of pilots and second-tier expert programmers into Trilobite$_G$ since it kept the existing safety features in place.

An essential component in the Trilobite$_G$ programming architecture is the new service programming model. This service layer provides a development level for advanced vehicle engineers and programmers who want to extend the vehicle's functionality and therefore expand the expressiveness of ALGAE by exposing services as new language constructs. The process of designing, implementing, and testing new services is assumed to not be time critical, i.e., will be done prior to any AUV deployment that will use the new service. At the second tier programming level, 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 Glider, for example, the number of inflections the vehicle has performed is exposed as a system variable. This is a simple example of a counting service. Many such pseudo services are already provided and can be used by new services created by a second-tier expert programmer. The driver on the AVBot, shown in Figure 4, speaks the gliderbus protocol to read and write into the glider's sensor memory. The runtime system exposes the glider's 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 [4]. It is the responsibility of the runtime system and the AVBot driver to interact with the flight controller to request and update any sensor data.

New services can be implemented in a variety of programming languages on the AVBot itself. The current programming architecture supports *C*, *Python*, and *GLOC*, a scripting language that can be executed on the flight controller. Depending on the complexity of a given service in terms of memory and computation, it may be advantageous for the service to be implemented on the flight controller. Flight controller services have direct access with no latency to flight information, while AVBot and science Persistor services may have faster access to acquired sensor data. Finally, these services may also migrate their execution between the computing platforms on the vehicle, if necessary, due to resource constraints, or because it may be beneficial by reducing energy consumption.

Newly created services can interoperate with existing services provided by the vehicle and services within the new programming framework. As mentioned earlier, 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. There are several different thermocline detection algorithms that have been proposed in the literature [7, 31, 38, 40, 43]. As shown in Figure 4, a 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. This is because the sensor is forced on for the first half yo of every mission segment as required by the selected thermocline

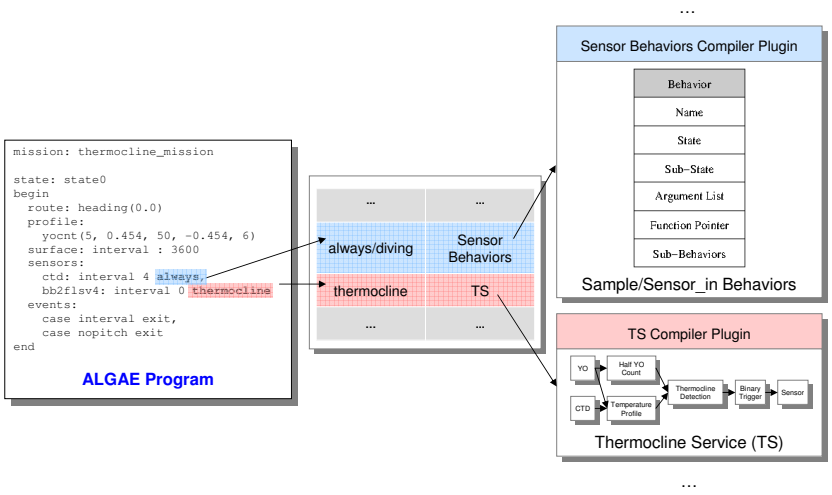


Figure 5. Compiler infrastructure.

detection algorithm. Redundant services can also be specified to implement different quality vs. energy consumption tradeoffs. It will be the responsibility of the higher level programmer or an automatic strategy to select the service with the desired quality vs. energy consumption tradeoffs. The idea of providing alternate mechanisms to produce the same or similar output or functionality is not new and has been applied in different energy-aware contexts [3, 25, 28, 36].

4.4 Compiler

The ALGAE compiler translates ALGAE programs into executable deployments. An executable deployment consists of a set of mission files, one for each state, and a corresponding state transition table. State transitions are executed by GPILOT each time the vehicle surfaces and contacts the Dockserver (see Figure 2).

A service writer extends the compiler through compile-time plugins. The plugins not only specify the name and functionality of the service, but also may contain code that performs consistency checks and optimizations at compile time. Figure 5 depicts, in part, how the ALGAE compiler will generate services for a state specification. In the case of the CTD sensor specification, the compiler will lookup the `always` token and use the sensor translation table to determine that the *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 example, if two sensors both specify `always`, they can be combined under the same vehicle activation flag during execution.

The `thermocline` sensor token in Figure 5, for the bb2flsv4 sensor, would also need to be looked up in the translation table. The thermocline service compiler plugin is then 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, the compiler can generate an error message or warning. Compiler plugins may also be used to select among redundant services that provide different quality vs. energy tradeoffs in their service implementation. The specification of redundant services in Trilobite$_G$ together with possible selection strategies are part of ongoing research.

5. Trilobite$_G$ Simulation Environment

For autonomous systems operating in dangerous environments, simulation is crucial for mission development, planning, real-time troubleshooting, and pilot training. Programming logic errors, misjudgement of resource availabilities and tradeoffs, or wrong reactions to physical conditions, can lead to system failure or even total system loss. Trilobite$_G$ has a crucial need for simulation at different levels of system granularity that captures vehicle behaviors within a virtual, but realistic physical environment.

A typical Slocum simulator as provided by the manufacturer 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 a 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.

The Trilobite$_G$ simulator is a faster-than-real-time, full software stack simulator for the Slocum Glider. The simulator includes detailed vehicular, environmental, and energy models to determine the glider's behavior during mission execution and is capable of running on commodity hardware. Full software stack simulation of the Slocum target system is necessary due to the complex interaction among different drivers for sensors, motors and software components. These complex software and hardware interactions make it difficult to design accurate high-level behavioral models of all system activities.

Since the Trilobite$_G$ simulator 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 we have 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, we have simulated up to 30 mission hours in one minute, 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, we have implemented a hybrid mode that simulates faster-than-real-time while underwater, and real-time while at the surface. 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.

Using the simulator, we were able to help researchers at IMCS pinpoint a problem with a glider that was deployed off the coast of South Africa. The glider was overdue for calling in through its satellite communication. Within 20 min, we recreated the glider's software state and identified a software inconsistency. This inconsistency prevented the glider from resurfacing and forced it to fly a shallow flight path between 5 m and 20 m water depth, causing it to consume significant battery energy in the process. Furthermore, we simulated the glider behavior predicting the next two weeks in a few minutes. We were able to confirm that the glider's flight path would not change unless a system exception occurred which would override the current flight behavior. Indeed, after nearly two weeks, such an exception occurred, allowing the pilots to reestablish contact with the vehicle and implement a bug fix. Unfortunately, the high number of inflections had substantially reduced the battery life, resulting in the need to cut the mission short.

Faster-than-real-time simulation can also be an important tool for path and resource planning [1, 17, 32]. A search space of different mission scenarios can be explored in real-time, supporting mission critical decisions while the glider is at sea. Energy efficient multi-core systems can run thousands of simulations at the same time, either remotely, or on battery-operated on-board computing systems [42].

An important aspect to any AUV deployment is to monitor and estimate a vehicle's energy consumption. In earlier work, we deployed a glider off the coast of New Jersey to measure the power dissipation of the individual components of the AUV during its mission [41]. These measurements were used to build Trilobite$_G$'s energy models and can be used to estimate the energy expended of the vehicle by analyzing its log files. Like the glider, our simulator also generates these log files which can be used by the energy model. We have also integrated a service into the simulator code that executes the energy model during flight and presents it as a glider sensor to facilitate evaluation. These mechanisms are used in our evaluation to provide a sense of the energy dissipation during each stage of a mission.

As part of this paper, we have made the simulator publicly available [3]. Users are able to execute ALGAE programs and can download the log files generated by the Trilobite$_G$ infrastructure. This platform provides a mechanism for users to gain experience on how various tradeoffs in vehicle behavior and sensing can affect missions. Furthermore, it will enable a larger research community to conduct research and perform experiments with AUVs, in particular with the Slocum Glider.

6. Experimental Results and Evaluation

Thermocline detection and tracking is an important application that we have explored in previous work [39]. Programming this application using the legacy programming system is a tedious and error-prone process due to the complexity of the interactions of behaviors in the layered-control stack. The manufacturer does not provide a thermocline detection or tracking behavior.

We have deployed several missions similar in nature to the sample ALGAE program listed in Figure 3. The flight profile of such a deployment is show in Figure 6. Like the sample program, the vehicle is tasked to detect and track a thermocline. When the thermocline has been established using readings from the CTD sensor, ECO-Puck sensors are activated to log data only within the thermocline. The thermocline is represented in Figure 6 by the CTD readings, in blue, indicating a drastic change in temperature within a few meters of depth. The activation of the ECO-Puck sensors, in red, are overlaid on top of the glider's flight profile, in green.

[3] The simulator is available through a web portal at `http://algaesim.cs.rutgers.edu`.

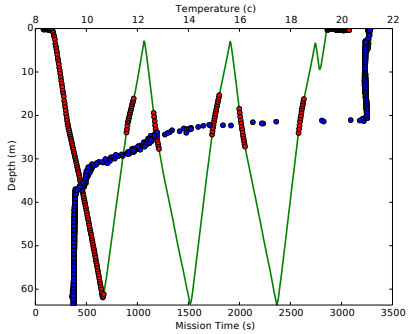


Figure 6. A Slocum Glider tracking a thermocline on September 27, 2012. The vehicle was tasked to activate a sensor only within the thermocline.This behavior cannot easily be expressed with the manufacturer provided programming model.

Sensor Management	Energy	Recall	Precision
Always On	3.7 kJ	1.00	0.08
Thermocline Tracking	1.4 kJ	0.98	0.22

Table 1. Thermocline tracking deployment results.

Unlike the sample program, the mission did not use the go-towaypoint *where* specification, but instead was instructed to zero its fin angle to fly straight ahead. The yo target depth (*how*) was 100 m instead of 50 m, but the glider never reached this target depth because the water depth at the deployment location was approximately 65 m. Finally, the only transition out of the mission was a surface interval of 45 min that required the vehicle to exit.

In the legacy glider programming framework, sensors are typically always activated. Continuous sensor use significantly increases the power required by the vehicle, which ultimately affects its overall flight and sensing endurance. To quantify the impact of continuous sensor usage, we compare the energy utilization of the deployed mission to a nearly identical simulated mission. In the simulated mission the ECO-Puck sensors are always activated, whereas in the deployed mission they are only activated within the thermocline. This simulated mission is equivalent to changing the bb2flsv4 sensor specification from `thermocline` to `always` in Figure 3. The two missions are compared using sensor energy, recall, and precision. As we will show, a simple sensor specification change can lead to dramatic energy savings.

The ECO-Puck sensors equipped on the glider during the deployment have a power dissipation of 1.22 W. For AUVs, making effective use of such an expensive sensor is of extreme importance. Thus, the aim of the thermocline tracking and trigger management approach is to reduce the energy required by the sensors while still capturing relevant sensor readings. The recall metric is defined as the proportion of the number of logged readings within the relevant area (thermocline) to the number of readings possible within the area if given perfect knowledge, for example, by an oracle. The precision metric is defined as the proportion of relevant readings recorded to the total number of readings recorded. Informally, this provides insights into how much relevant sensing information is missed (false negative) when performing sensor management, and how much irrelevant information is collect (false positive).

The results of our sensor management comparison of the deployments are shown in Table 1. Applying the thermocline tracking approach saves more than half the energy while still capturing most of the data within the area of interest. In addition, the precision sig-

nificantly increases by almost a factor of three. The precision could, in fact, be improved further if the algorithm was not forced to turn on the ECO-Puck sensors at the surface and for the first dive. The decision to force the sensors on is justified by the fact that the thermocline tracking algorithm learns about the temperature profile as it flies. To ensure that no relevant readings are missed, the sensors are kept on for the first profile and at the surface until the glider has some notion of where the thermocline resides in the water column. Considering that most dive segments in long term deployments are much longer than the sea-trial we presented, the impact of this policy on the precision decreases as the dive segments become longer.

In summary, specifying a thermocline triggering strategy in the existing legacy programming system would have been prohibitively difficult for an oceanographer, pilot, or even a vehicle engineer. The presented example demonstrates the simplicity by which Trilobite$_G$ enables users to specify and implement complex services and is representative of the potential the framework has for the oceanographic community.

7. Conclusion and Future Work

Programming autonomous systems, such as buoyancy-driven AUVs, is an extreme challenge due their particular operational characteristics. AUVs are deployed in hostile environments, operate fully autonomously for the bulk of their missions, and need to adapt to changes in physical environments and available resources, particularly battery energy. This often requires making trade-off decisions in real-time while limited communication with the AUV is possible. These factors put an extreme burden on programmers who are often themselves working in stressful environments while piloting a fleet of vehicles.

This paper introduces Trilobite$_G$, a two-tier programming architecture that consists of a domain-specific language layer for oceanographers and marine scientists (ALGAE), a lower level service definition layer for vehicle engineers, and a set of simulators to support both programming layers. The design is driven by the need to provide a safe and practical programming framework that both oceanographers and vehicle engineers can trust. This means that existing glider safety features and systems are integrated within the new architecture. Correspondingly, the language design reflects the operational constraints of AUV deployments where vehicle communication is only possible during their time at the surface. Other autonomous systems, such as satellites and robots, have similar constraints and can benefit from such a programming approach.

The domain-specific language is based on a simple state-based programming model. Again, state transitions are only possible while the AUV is at the surface and reachable via satellite communication. Each state specifies a flight route, a flight profile, a sensor set with possible trade-off specifications, the length of each flight segment during which the glider is submerged and operating fully autonomously, and events that will lead to the transition to other states. The service layer provides more complex system behaviors which can dynamically change the glider state while it is submerged and in fully autonomous mode. The safe implementation of services require an in-depth knowledge of the system characteristics that only vehicle engineers have. Services are used at the domain-specific language level to provide more complex system behaviors, such as adaptive sensing based on thermocline detection.

The prototype Trilobite$_G$ programming architecture has been implemented and consists of the ALGAE compiler, an on-shore automatic pilot and mission monitor, and an on-board runtime environment. Trilobite$_G$ has been extensively tested through simulations and multiple deployments in the Atlantic Ocean off the coast of New Jersey. The programming architecture is able to express overall system behavior such as thermocline tracking and other behaviors that are not supported by the legacy programming environment. This was accomplished without losing any existing safety features of the glider, which was one of the major concerns of the oceanographers, pilots, and vehicle engineers we worked with.

The faster-than-real-time simulator proved to be very useful for path, resource, and tradeoffs planning, and for real-time troubleshooting. We believe that the ALGAE programming model in conjunction with the simulation environment is a safe, robust, and practical architecture that can support programming under extreme conditions. This is due to its simplicity at the higher level where real-time decisions have to be made, and its expressiveness at the lower service level where services are thoroughly tested before being released for use at the higher level. The design of the Trilobite$_G$ architecture was based on a seven year collaboration with oceanographers and AUV engineers at two major marine science research institutes (IMCS and MBARI), and the experiences we gained with our own gliders. As part of this paper, we have made our Trilobite$_G$ system and simulator available through a web interface, allowing a larger research and educational community to target these important systems.

There are many promising directions for future research, including compiler optimizations that map computations to different system components in order to save energy, strategies that transparently map services to on-shore or on-board resources, and programming abstractions that allow groups of AUVs to be treated as a single scientific instrument. The latter goal has become of increasing importance since advanced sensor payloads have become too large to be supported by only a single vehicle. Coordination among several autonomous AUVs will benefit from acoustic communication. We are planning to include different communication services in our Trilobite$_G$ system that will enable coordinated swarming behaviors.

Acknowledgements

This work has been partially funded by NSF grants CSR-CSI #0720836 and MRI #0821607. Any opinions, findings, and conclusions or recommendations expressed in material related to this project do not necessarily reflect the views of the National Science Foundation. We would like to thank the oceanographers, pilots, and glider engineers we have collaborated with for their invaluable input to the Trilobite$_G$ system. In particular, we would like to thank David Aragon, Tina Haskins, Chip Haldeman, and Oscar Schofield at IMCS.

References

[1] A. Alvarez, A. Cait, 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.

[2] A. Alvarez, R. Stoner, and A. Maguer. Performance of pumped and un-pumped CTDs in an underwater glider. In *OCEANS 2013 IEEE - San Diego*, pages 1–5, 2013.

[3] W. Baek and T. 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, 2010.

[4] M. R. Benjamin, H. Schmidt, P. M. Newman, and J. J. Leonard. Nested autonomy for unmanned marine vehicles with MOOS-IvP. *Journal of Field Robotics*, 27(6):834–875, 2010.

[5] R. I. Carnegie Mellon Robotics Academy. RobotC. http//www.robotc.net.

[6] B. Claus, R. Bachmayer, and C. D. Williams. Development of an auxiliary propulsion module for an autonomous underwater glider. In *Proceedings of the Institution of Mechanical Engineers, Part M: Journal of Engineering for the Maritime Environment*, November 2010.

[7] N. Cruz and A. Matos. Adaptive sampling of thermoclines with autonomous underwater vehicles. In *MTS/IEEE OCEANS 2010 - Seattle, WA*, September 2010.

[8] 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.

[9] 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.

[10] P. Dias, S. Fraga, R. Gomes, G. Goncalves, F. Pereira, J. Pinto, and J. Sousa. Neptus - a framework to support multiple vehicle operation. In *MTS/IEEE Oceans 2005 - Europe*, volume 2, pages 963 – 968 Vol. 2, 20-23 2005.

[11] P. Dias, G. Goncalves, R. Gomes, J. Sousa, J. Pinto, and F. 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, 15-19 2006.

[12] E. Dijkstra. Go To statement considered harmful. *Communications of the ACM*, 11(3):147 – 148, March 1968.

[13] C. Duarte and 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. .

[14] C. Duarte, G. 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.

[15] C. N. Duarte, G. R. Martel, C. Buzzell, D. Crimmins, R. Komerska, S. Mupparapu, S. Chappell, D. R. Blidberg, and R. Nitzel. A common control language to support multiple cooperating AUVs. In *Proceedings of the 14th International Symposium on Unmanned Untethered Submersible Technology*, 2005.

[16] E. Eberbach, C. 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.

[17] M. Eichhorn. Optimal Path Planning for AUVs in Time-Varying Ocean Flows. In *16th Symposium on Unmanned Untethered Submersible Technology (UUST09)*, Durham NH, USA, August 23-26 2009.

[18] C. C. Eriksen, T. J. Osse, R. D. Light, T. Wen, T. W. Lehman, P. L. Sabin, J. W. Ballard, and A. M. Chiodi. Seaglider: A long-range autonomous underwater vehicle for oceanographic research. In *IEEE Journal of Oceanic Engineering*, volume 26, October 2001.

[19] S. Glenn, O. Schofield, J. Kohut, J. McDonnell, R. L. D. Seidel, D. Aragon, T. Haskins, E. Handel, C. Haldeman, I. Heifetz, J. Kerfoot, E. Lemus, S. Lictenwalner, L. Ojanen, J. Roarty, F. Carvalho, A. Lopez, A. Martin, C. Jones, D. Webb, J. Miller, M. Lewis, S. McLean, A. Martins, C. Barrera, A. Ramos, and E. Fanjul. The trans-atlantic Slocum glider expeditions: A catalyst for undergraduate participation in ocean science and technology. *Marine Technology Society Journal*, 45(1):52–67, January/February 2011.

[20] M. Godin, J. 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.

[21] B. W. Hobson, J. G. Bellingham, B. Kieft, R. McEwen, M. Godin, and Y. Zhang. Tethys-class long range AUVs - extending the endurance of propeller-driven cruising AUVs from days to weeks. In *Proceedings of IEEE-OES AUV Symposium*. Southampton, U.K., September 2012.

[22] HYCOM consortium. Hycom. http://www.hycom.org/.

[23] N. Instruments. Labview Robotics. http//www.ni.com/robotics.

[24] R. J. Komerska and S. 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. 2007.

[25] A. Lachenmann, P. J. Marrón, D. Minder, and K. Rothermel. Meeting lifetime goals with energy levels. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 131–144, 2007.

[26] J. Mare. Path following algorithm for minimally specified lawnmower type AUV missions. In *OCEANS 2010 IEEE - Sydney*, 2010.

[27] S. Mupparapu, S. Chappell, R. Komerska, D. Blidberg, R. Nitzel, C. Benton, D. Popa, and A. Sanderson. Autonomous systems monitoring and control (asmac) - an auv fleet controller. In *Autonomous Underwater Vehicles, 2004 IEEE/OES*, pages 119–126, 2004.

[28] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the sixteenth ACM symposium on Operating systems principles*, SOSP '97, pages 276–287, 1997.

[29] Open Source Robotics Foundation. Robot Operating System (ROS). http//www.ros.org.

[30] Persistor Instruments Inc. Cf1 computer system. Marstons Mills, MA. http://www.persistor.com.

[31] S. Petillo, A. Balasuriya, and H. Schmidt. Autonomous adaptive environmental assessment and feature tracking via autonomous underwater vehicles. In *IEEE OCEANS 2010 - Sydney, Australia*, May 2010.

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

[33] O. Schofield, J. Kohut, D. Aragon, L. Creed, J. Graver, C. Haldeman, J. Kerfoot, H. Roarty, C. Jones, D. Webb, and S. Glenn. Slocum gliders: Robust and ready. *Journal of Field Robotics*, 24(6):473–485, 2007.

[34] A. Shchepetkin and J. McWilliams. The regional oceanic modeling system (roms): a split-explicit, free-surface, topography-following-coordinate oceanic model. In *Ocean Modelling*, volume 9, pages 347–404, 2005.

[35] J. Sherman, R. E. Davis, W. Owens, and J. Valdes. The autonomous underwater glider Spray. In *IEEE Journal of Oceanic Engineering*, volume 26, October 2001.

[36] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M. D. Corner, and E. 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, 2007.

[37] Teledyne Webb Research. Slocum Glider. Falmouth, MA, 2013. http//www.webbresearch.com/slocum.htm.

[38] D. Wang, P. F. Lermusiaux, P. J. Haley, D. Eickstedt, W. G. Leslie, and H. Schmidt. Acoustically focused adaptive sampling and on-board routing for marine rapid environmental assessment. *Journal of Marine Systems*, 78(Supplement 1):S393 – S407, 2009.

[39] H. Woithe and U. Kremer. Feature based adaptive energy management of sensors on autonomous underwater vehicles. *Ocean Engineering*, 97(3):21 – 29, March 2015.

[40] H. Woithe and U. Kremer. A programming architecture for smart autonomous underwater vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2009 - St. Louis, MO*, October 2009.

[41] H. Woithe, I. Chigirev, D. Aragon, M. Iqbal, Y. Shames, S. Glenn, O. Schofield, I. Seskar, and U. Kremer. Slocum glider energy measurement and simulation infrastructure. In *OCEANS 2010 IEEE - Sydney*, 2010.

[42] H. C. Woithe, W. Brozas, C. Wills, B. Pichai, U. Kremer, M. Eichhorn, and M. Riepen. Enabling computation intensive applications in battery-operated cyber-physical systems. In *MARC Symposium*, pages 34–39, July 2012.

[43] Y. Zhang, J. Bellingham, M. Godin, and J. Ryan. Using an autonomous underwater vehicle to track the thermocline based on peak-gradient detection. *IEEE Journal of Oceanic Engineering*, 37(3), July 2012.