The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

A Programming Architecture for Smart Autonomous Underwater Vehicles

Hans Christian Woithe and Ulrich Kremer

Abstract— **Autonomous underwater vehicles (AUVs) are an indispensable tool for marine scientists to study the world's oceans. The Slocum glider is a buoyancy driven AUV designed for missions that can last weeks or even months. Although successful, its hardware and layered control architecture is rather limited and difficult to program. Due to limits in its hardware and software infrastructure, the Slocum glider is not able to change its behavior based on sensor readings while underwater. In this paper, we discuss a new programming architecture for AUVs like the Slocum. We present a new model that allows marine scientists to express AUV missions at a higher level of abstraction, leaving low-level software and hardware details to the compiler and runtime system. The Slocum glider is used as an illustration of how our programming architecture can be implemented within an existing system. The Slocum's new framework consists of an event driven, finite state machine model, a corresponding compiler and runtime system, and a hardware platform that interacts with the glider's existing hardware infrastructure. The new programming architecture is able to implement changes in glider behavior in response to sensor readings while submerged. This crucial capability will enable advanced glider behaviors such as underwater communication and swarming. Experimental results based on simulation and actual glider deployments off the coast of New Jersey show the expressiveness and effectiveness of our prototype implementation.**

I. Introduction and Motivation

In recent years, autonomous underwater vehicles (AUVs) have become an indispensable tool for marine scientists to learn more about our world's oceans. Traditionally, the acquisition of oceanic 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 [15]. Not only are they more cost efficient, but they enable data to be collected in environments that were historically inaccessible or too dangerous. The advent of underwater vehicles has revolutionized oceanographic and marine research by allowing scientists to maintain a constant presence in the world's oceans. One such underwater vehicle is the Slocum electric glider, which is used as our implementation platform and is developed by Teledyne Webb Research [14]. Unlike propeller driven vehicles [8], [9], [17], [16], it belongs to a class of AUVs which achieve forward propulsion by changing its buoyancy [14], [18], [5], [4]. The Slocum accomplishes this by using a pump 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 [7].

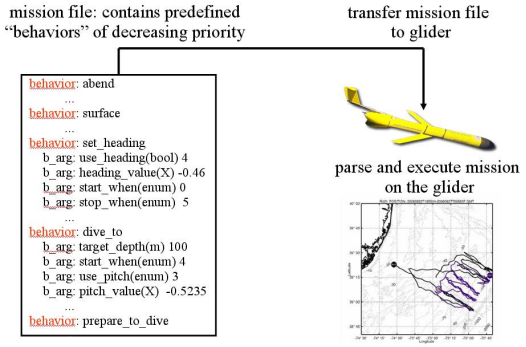


Fig. 1. Current glider programming model

The existing software stack used to program many AUVs, including the Slocum glider, are based on the layered control system [3], [6], [2]. Marine scientists specify the actions they wish the glider to perform through a set of *behaviors* written in mission files as illustrated in Fig. 1. Users and their gliders are thus limited to the actions provided by the set of behaviors the manufacturer supplies. Writing new behaviors for the Slocum glider from scratch is a highly error-prone task and 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 existing missions by modifying 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.

The Slocum glider in its current state is also extremely static in that it cannot react to the dynamic environment it is in. It can only be re-programmed during its periodic surfaces via an expensive satellite modem uplink [10]. Some phenomena are short-lived and so the opportunity to observe them may have been lost by the time a remote operator is able to instruct the glider to re-survey the area of interest. Finally, the number and complexity of sensors that can be installed and supported is limited by the glider's two 16MHz CPUs.

We believe that in order to reach an AUV's full potential, it is necessary to allow marine scientists to express their

This work was partially supported by NSF grants CSR-CSI #0720836 and MRI #0821607

H. Woithe and U. Kremer are with the Department of Computer Science, Rutgers University, 110 Frelinghuysen Rd., Piscataway, NJ 08854, USA {hcwoithe, uli}@cs.rutgers.edu

mission objectives at a level of abstraction that makes sense to them. Thus, we introduce a new domain specific programming language, compiler, and runtime system that enables easier mission creation and maintenance. This paper makes the following main contributions:

1) The discussion of a new programming model that specifies events to make transitions within a finite state machine. States in the machine may be layered, allowing control to be returned to previously active states.
2) The discussion of the implementation of the proposed programming model as part of a new programming architecture for the Slocum glider. This implementation illustrates the challenges of integrating new software/hardware within an existing system. The new hardware includes a Linux-based single board computer (SBC). The new software consists of a compiler, runtime system and two hook interfaces to the existing system.
3) The evaluation of our prototype system through simulations and ocean deployments off the coast of New Jersey. The glider equipped with the new programming architecture was able to track a thermocline, significantly enhancing the capabilities of the glider and illustrating the effectiveness of our new programming architecture.

The new programming architecture implemented on the Slocum glider is flexible and allows the integration of new sensors through standard programming interfaces and corresponding runtime library implementations. Such new sensors may include underwater acoustic communication which can be used to share sensor data among a group of gliders, or to implement tightly coordinated swarming behaviors. Together with the enhanced computing capabilities, our new programming infrastructure enables marine scientists to more effectively use AUVs.

II. Domain Specific Language

The existing software stack is based on Brook's layered control system. The writing and modification of mission files is a non-trivial task. To properly program the AUV, a user must have detailed knowledge of the concepts of a layered control system as well as how each behavior changes the glider's state. This requires intimate understanding of how a behavior has been implemented and how they will function at any time during the mission. In reality, most users do not wish to be burdened with low level implementation details. A good domain specific language should provide the user with programming abstractions that allow them to easily express their goals. Without an intuitive mechanism to specify the objectives to be attained, the effectiveness of any tool is dramatically reduced. Additionally, the current framework does not allow the AUV to dynamically adapt to its changing environment.

We have constructed a high level programming language and compiler that allows users to easily specify their desired

```
mission sampleMission

state s1
begin
 flightroute nav(3910.5, -7349.8058)
 flightprofile yo(5.0, 10, .454, .454)
 sensors missiontime, depth, pitch
 events
  case missiontime >= 2000 goto s3
  case missiontime >= 500 &&
       missiontime < 1500 call s2
end

state s2
begin
 flightroute noheading
 flightprofile thermotrackyo(5, 20, .454,
                             .454, 10, 10)
 sensors missiontime, depth, pitch, temp
 events
  case missiontime >= 1500 return
end

state s3
begin
 flightroute nav(3910.5, -7349.5058)
 flightprofile yo(5, 20, .454, .454)
 sensors missiontime, depth, pitch
 events
  case missiontime >= 2500 exit
end
```

Fig. 2. An example program written in our domain specific language.

intentions, even if these intentions include reacting to observed phenomena. Our design was guided by requirement specifications of the Slocum glider engineers at Rutgers University. Their most basic view of a mission involved that of states and state transitions. A glider is instructed to perform an action and should only transition to perform another action based on events. There are situations were it is desirable to return to a previous state in order to resume the mission objective that was present before the phenomena had been observed and corresponding actions were taken.

Fig. 2 contains an example program written in the current prototype programming language and compiler system which resembles a stack based finite state machine The user specifies a sequence of states that they wish the glider to be in during a deployment. Each state contains a `flightroute` which describes how the vehicle should be navigated. This could be as simple as having no heading or flying directly to a waypoint or a more complex sweep of an area specified by a convex hull. The `flightprofile` details how the AUV should glide through the water. Examples here would include a simple *yo* action (a series of climbs and dives) or more elaborate movements such as the tracking of a thermocline as described in the experimental section. Sensors to be logged by the system are defined in a comma separated list using the `sensors` keyword.

Transitions between glider states occur based on user defined `events`. To meet the requirements of our user group, different transition mechanisms were developed. The `call` and `return` transitions provide a call stack of states. Thus, it is possible to transition to states temporarily and return to the calling state much like a sequence of function calls. In contrast, the `goto` transition destroys the current

call stack and is used when there is no need to return to previous states in the order specified by the call stack.

The program in Fig. 2 begins in state *s1*. The state instructs the glider to begin navigating to the waypoints indicated. The *yo* `flightprofile` creates a climbing and diving sequence between 5 and 10 meters at an angle of .454 radians (26 degrees). When the time sensor indicates the current mission time to be greater than or equal to 500 seconds, the glider will transition to the state *s2*. After 1000 seconds of tracking a thermocline (where the two additional `flightprofile` parameters indicate a temperature and depth threshold) the `return` statement in *s2* will cause control to return back to *s1*. After 2000 seconds into the mission a transition to *s3* will occur which differs only in its target depth. Since we know *s3* to be our final state, as indicated by the `exit` transition, it will never be necessary to return to *s1* and thus a `goto` is used to transition to *s3* from *s1*.

Our infrastructure is such that `flightprofiles` and `flightroutes` are easily added to the programming language and compiler. The addition of variables to our language has also allowed more complex actions to be constructed based on lower level primitives so modification to the compiler are not strictly necessary. The `thermotrackyo` (described later), for example, can be implemented using several states and variables, where the variables can change the state's `flightprofile` parameters based on recently collected depth/temperature data points.

Although quite simple, the provided language can be extremely flexible. Currently, we are in the process of stabilizing the compiler infrastructure so that we can perform a user study of the proposed language with day-to-day users of Slocum gliders. We would like to use the study to determine if our programming framework is flexible and intuitive enough to meet their requirements.

III. Implementation

In order to enable any significant change to the current architecture, such as reacting to sensor readings, cooperative control of gliders [11] and the support of complex sensors, new computer hardware is needed as part of the glider infrastructure. We chose an ARM-based (TS-7800) and a x86-based (TS-5500) single board computer (SBC) running Linux as our initial development platform. Table I contains the hardware specifications of both SBCs [19]. The reported power measurements were taken under load using a Tektronix 3014 Oscilloscope. Other SBCs will likely be considered in the future; depending on mission requirements it may be advantageous to choose one board over another.

To execute missions written in our programming language, we must gain control of the current software system. One key objective of our overall design is to remain as non-intrusive as possible. The existing system is well designed to deal with safety issues that ensure the physical integrity of the glider and we wanted to take full advantage of this. The glider is also a successful commercial product with many customers, thus we want to remain backward compatible with the existing system to allow users to gracefully migrate to our infrastructure.

TABLE I
SBCs that have been installed in the Slocum Glider.

	TS-5500	TS-7800
CPU:	x86 at 133MHz	ARM9 at 300MHz or 500MHz
Memory:	64MB	128MB
Storage:	1 CF	1 FullSD, 1 MicroSD, 2 SATA
USB:	2	2
Serial Ports:	3	Up to 10
Analog to Digital :	8 12-bit channels	5 10-bit channels
Active Power:	2.7W	3.42W (300MHz), 4.14W (500MHz)
OS:	Linux 2.4	Linux 2.6

The design that satisfies the objectives described above is shown in Fig. 4. The Slocum AUV's current processors, the flight controller and the science computer, communicate via a 9600 baud serial connection. The flight controller is responsible for safely piloting the vehicle according to the mission specifications. Readings from installed sensors are collected by the science computer and are directed to the flight controller for logging. The readings along with other glider state information are stored in the flight controller's *sensor array*. This sensor array can be considered the flight controller's data memory and is periodically written to persistent storage (typically every 2-4 seconds).

The SBC which hosts our runtime system can converse with the science computer through its own 115200 serial connection. Using the existing infrastructure, which permits the reading and writing of the flight controller's data memory, we have developed a driver layer (*hookprog*) on the science computer so that it may act as a proxy for the SBC. Thus,

```
double min, max, currTemp;
eventList el, retEl;
event iter;
event reachMinDepth = { LEQ, M_DEPTH, &min }
event reachMaxDepth = ...
event tempChange = ...
addEvent(el, reachMinDepth);
addEvent(el, reachMaxDepth);
addEvent(el, tempChange);
min = 5; max = 20;
while(1) {
  retEl = gliderRun(el);
  while(iter = nextEvent(retEl, iter) {
    if(iter == reachMinDepth)
      dive();
    if(iter == reachMaxDepth)
      climb();
    if(iter == tempChange)
      trackThermo(&min, &max, 10, 10);
  }
}
```

Fig. 3. Compiler-generated skeleton C program that is executed by our runtime system.

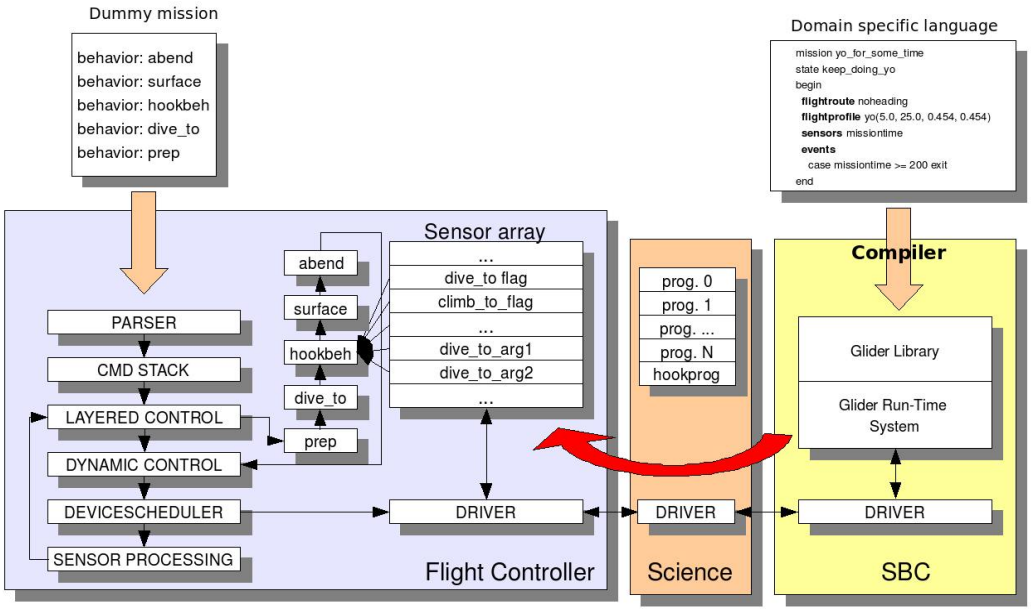


Fig. 4. The new programming architecture; from left to right: flight controller with skeleton mission and layered control implementation, science computer, and our SBC with a domain specific programming language, runtime system and compiler.

with the layer in place, our computing system can also read and write data to the data memory.

Another software hook introduced into the existing system is the *hookbeh* behavior. Based on the instructions sent by the SBC through the sensor array, the behavior dynamically generates and executes a sequence of sub-behaviors that perform the glider actions on our behalf. This allows our software infrastructure to command the glider while remaining backward compatible. If a user decides not to run our software, they can simply neglect to include the behavior in their layered control mission file, thus removing the hook.

Finally, the compiler for our new domain specific programming language is implemented using the open-source tools Flex [1] and Bison [12]. The compiler generates C code for our glider runtime and library system. Fig. 3 shows the skeleton C code generated by our compiler for state *s2* of Fig. 2. Each `event` the state will react to is declared and associated with a predefined or custom function and its required data. All events are then added to an event list which is passed to the event system `gliderRun`. The generated C code is then compiled using GCC [13], targeted for our particular single board computer installed in the glider.

The runtime system is the software stack which conducts the necessary communication with the science computer and presents an event-driven programming interface similar to our programming language, albeit at much lower level (as shown in Fig. 3). Along with the glider library, which provides commonly used routines such as performing a sequence of dives and climbs, it is easy for programmers experienced in C to create a mission directly using the runtime. Users could also implement new routines that allow access to new sensors, data processing services, or decision making strategies. It is at this level that the additional processing power supplied by our single board computers can be harnessed.

IV. Experiments

Most of the development for our programming infrastructure was made using a "shoe-box" simulator. This simulator contains a subset of components used in the production glider. To verify our system's functionality we used the shoe-box simulator, a glider on a bench-top, as well as short deployments in the Atlantic. In the bench-top configuration the glider performs as it would at sea, by moving motors and turning sensors on and off, but sensor values such as depth and temperature are simulated. In all configurations, our SBC commands the glider software infrastructure by using one of the science computer's serial interfaces to communicate with our software hooks.

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

To detect and track such a thermocline, we have developed a simple algorithm that observes when a threshold has been met in recent depth/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, we were able to successfully track a thermocline using a benched glider in simulation mode. A domain specific program would use a state similar to *s2* of Fig. 2 to accomplish this task. Fig. 5 shows a glider's depth profile of such an experiment, where the thermocline is represented by the gradient. The warm surface water is

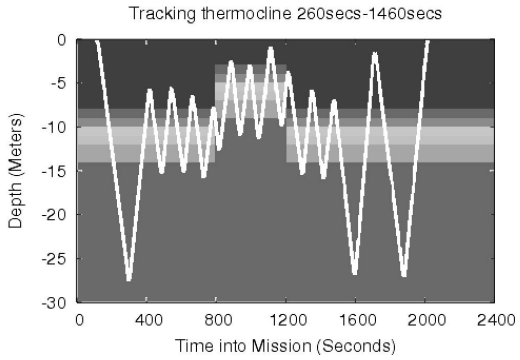


Fig. 5. On bench thermocline tracking simulation.

assumed to be 24 degrees Celsius and the cool deep water 7 degrees Celsius. 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. We used our infrastructure to pilot the tracking of the thermocline from 260 seconds to 1460 seconds into the mission; after this it would return to finish its layered control mission. The profile depicted in Fig. 5 confirms that, although simple, the tracking algorithm performs well.

As part of the experimental evaluation of our system we were able to perform two deployments in the Atlantic Ocean approximately 30 kilometers 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 our 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 our language to change its target depth from 15 to 25 meters 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 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 our simulations did not reveal the error in our algorithm, we considered it a prosperous deployment.

The focal point of our 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. Fig. 6(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 meters where dramatic temperature change can be observed. A mission nearly identical to the one in the first deployment was carried out. The vehicle's depth profile in Fig. 6(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 seconds 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.

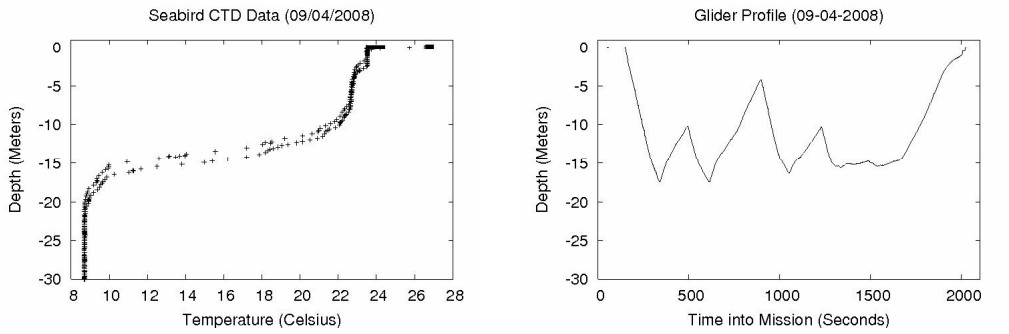
It is interesting to note that the vehicle leveled out at 15 meters towards the end of the mission. Although our infrastructure was still in control for a small portion of this time, it piloted the vehicle to climb. For most of the hovering period however the glider's layered control system was in control 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, we have shown that our 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 a thermocline tracking algorithm was not the focus of our experiments. We are happy to leave this task to a knowledgeable marine scientist.

V. Conclusion and Future Work

The current programming model on the Slocum gliders is not well suited to react to its environment nor is it trivial to program for. We have described an initial version of our programming language based on needs conveyed to us by engineers which work with the AUV on a daily basis. We have also presented how we are able to pilot the vehicle using software hooks that were embedded into the existing system. Finally, we have demonstrated the power and flexibility of our system through simulations and deployments in the Atlantic Ocean.

In the near future, we will extend the current compiler to handle additional language features to support on-board power management. To this end, we are in the process of adding power measurement capabilities to the glider. The insight we will gather from the energy consumption of the glider and its sensors during simulation and actual deployments will give us an idea of possible tradeoffs between battery consumption, quality of sensor readings and glider flight profiles. Such tradeoffs are necessary for missions where not all sensors can be active all the time. It is the programmers responsibility to specify the desired tradeoffs through new language constructs. Based on these specifications, our compiler and runtime system will find the most effective operation plan that satisfies the desired tradeoffs.

Additional work needs to be done at the hardware level as well. For an AUV such as the Slocum glider, designed for



(a) Thermocline as detected by Sea-Bird CTD

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

Fig. 6. Mission to track a thermocline in the Atlantic Ocean.

endurance missions lasting weeks or months, our two SBCs are too power hungry. We are planning to develop and deploy an adaptive system that can adjust itself to the particular computation needs of a given flight profile or sensor set.

A user study with Slocum engineers and undergraduate students is in the planning stage. Their feedback will help to determine if the current programming model is sufficiently flexible and intuitive for creating missions. A port of the language to a propeller driven AUV is also underway, giving us the opportunity to evaluate our new programming architecture on two different AUV platforms with different hardware and application characteristics.

VI. Acknowledgments

We would like to thank Scott Glenn, Oscar Schofield and David Aragon from the Institute of Coastal and Marine Sciences at Rutgers for their support and help. In particular, we appreciate David's advice and patience while teaching us how to use and operate the Slocum glider. We would also like to thank Chris Mansley and John McCabe for reviewing this paper.

References

[1] Flex: The fast lexical analyzer. http://flex.sourceforge.net/.

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

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] LLC. Hydroid. Remus auv, http//www.hydroidnc.com/products.html. Pocasset, MA.

[9] 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 2008.

[10] Iridium Satellite LLC. Iridium satellite phones, http://www.iridium.com. Bethesda, MD.

[11] Derek A. Paley, Fuming Zhang, and Naomi Ehrich Leonard. Cooperative control for ocean sampling: The glider coordinated control system. *IEEE Transactions on Control Systems Technology*, 16:735–744, 2008.

[12] GNU Project. Bison - gnu parser generator. http://www.gnu.org/software/bison/.

[13] GNU Project. Gcc, the gnu compiler collection. http://gcc.gnu.org/.

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

[15] 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.

[16] 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 ‘97*, volume 2, October 1997.

[17] 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.

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

[19] Technologic Systems. http://www.embeddedarm.com. Foutain Hills, AZ.