Towards a Resource-Aware Programming Architecture for Smart Autonomous Underwater Vehicles[*]

Technical Report DCS-TR-637

Hans Christian Woithe Denitsa Tilkidjieva Ulrich Kremer
{hcwoithe,denitsa,uli}@cs.rutgers.edu

Department of Computer Science
Rutgers University

June 2008

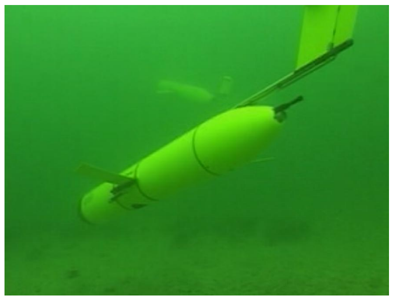
1 Introduction and Motivation

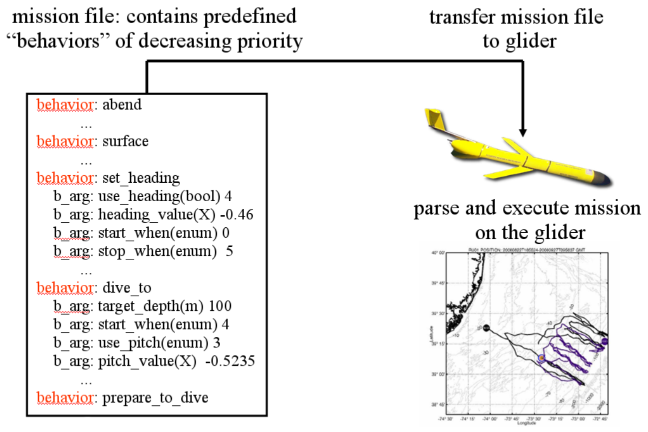
The world's oceans represent over 99% of the earth's ecological space that can support life. This vast space is still widely undiscovered, and the notion that "we know more about the surface of the moon than the bottom of our oceans" still holds. Many global oceanographic phenomena are not yet well understood, in particular their impact on global warming and marine life. In recent years, autonomous underwater vehicles (AUVs) have become an indispensable tool for marine scientists to learn more about our world's oceans and large water bodies. Their use has replaced the tedious process of gathering oceanographic data through sensor probes lowered into the water from surface vessels and operated by scientists. Today's AUVs are able to gather orders of magnitude more data than the traditional approach, operating at a fraction of the overall costs, and allowing deployment under harsh environmental conditions such as those encountered during hurricanes or in polar regions.

The main thrust in the design of autonomous underwater vehicles such as the Slocum glider [12, 13] has been the development of a reliable, effective, and low cost data-acquisition instrument, typically for a small collection of sensors to measure water salinity, pressure, and temperature. The collected sensor data is stored on-board, and a subset of the data can be communicated back to a mission center, allowing scientists to monitor the progress of the glider mission, and potentially change mission parameters, if necessary. At the end of the mission, the glider is retrieved, together with its complete sensor data sets. Although extremely effective as compared to previous oceanographic data-acquisition technologies, state-of-the-art gliders such as the Slocum glider fall short of reaching their full potential. They are (1) very difficult to program by marine scientists, (2) have very limited resources in terms of computation, storage, and communication capabilities, (3) are extremely energy constrained, and (4) rely heavily on a remote mission control center for safe and effective operation. To address these shortcomings, we have designed a new resource-aware programming architecture for the Slocum glider.

The new hardware/software architecture consists of a new programming abstraction and an additional computing system to be deployed within the glider. New compiler and runtime optimizations will balance power/energy needs across a set of sensors. Underwater communication capabilities

[*]This research was supported in part by NSF grant CSR/CSI-0720836. The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.


(a) Two gliders with a downward and upward trajectory (sawtooth-shaped flight path).


mission file: contains predefined "behaviors" of decreasing priority
transfer mission file to glider
parse and execute mission on the glider

```
behavior: abend
        ...
behavior: surface
        ...
behavior: set_heading
   b_arg: use_heading(bool) 4
   b_arg: heading_value(X) -0.46
   b_arg: start_when(enum) 0
   b_arg: stop_when(enum)  5
        ...
behavior: dive_to
   b_arg: target_depth(m) 100
   b_arg: start_when(enum) 4
   b_arg: use_pitch(enum) 3
   b_arg: pitch_value(X)  -0.5235
        ...
behavior: prepare_to_dive
```

(b) Current glider programming model.

Figure 1: Webb Research's Slocum Glider.

will allow gliders to coordinate their sensing tasks across a swarm of gliders, and an overall mission planning tool will give feedback about mission critical parameters such as estimated mission duration given an energy budget, a set of desired sensors, and a flight path as input. The new programming architecture will significantly enhance the capabilities of the gliders, making them "smarter" in terms of power/energy usage and more autonomous by reducing the dependence on a mission control center.

2 Slocum: An Autonomous Buoyancy Driven Underwater Vehicle

Our initial goal is to support buoyancy driven autonomous underwater vehicles (AUVs), and particularly the Slocum glider. Propeller-driven AUVs such Remus [7] have different mission profiles, with mission lengths in terms of hours rather than days or weeks. The Slocum glider is a 1.8m long torpedo-shaped, winged AUV developed by Webb Research Corporation [12, 13]. Instead of using a propeller, it achieves forward propulsion by changing its buoyancy, resulting in a sawtooth-shaped gliding trajectory with a forward speed of 20-30cm/s at a dive angle of 26 degrees. Both angle and speed can be adjusted by selecting a particular buoyancy together with a center of gravity. This adjustment is done dynamically using a buoyancy pump and a pitch motor that moves an internal battery pack. Lateral steering is supported by a tail fin rudder. Figure 1(a) shows two Slocum gliders, with the glider in the foreground at a descending, and in the background with an ascending flight path.

The glider has an on-board GPS receiver and typically communicates with a mission control center via an Iridium satellite phone. GPS positioning and satellite communication is only possible when the glider is at the surface. Multiple sensors can be mounted within or on the outside of the vehicle. An acoustic altimeter is installed in the nose section to avoid collisions, in particular collisions with the sea floor. Depending on the gearbox of the buoyancy pump, the glider can operate at depths of up to 30, 100, 200, or 1000 meters. Glider missions may last weeks or months, depending on the selected sensors, the desired flight path, and the frequency of sensor readings. A more detailed description of this class of underwater vehicles can be found in [4].

Current Programming Architecture

The current Slocum glider has only limited on-board processing capability (two Motorola 68338 based single board systems, 1MB flash, 512KB SRAM, 8KB virtual EEPROM). The programming environment allows users, i.e., oceanographic scientists and biologists, to express the actions to be performed by the glider as a priority stack of behaviors (layered control) [3, 15, 1]. Such a priority stack specification is referred to as a "mission", and is written in a mission file. Each individual behavior selects an action that may be overwritten by a behavior with a higher priority. Every 4 seconds, the operating system running on the glider traverses the priority stack of behaviors, from lowest to highest priority, to determine the next action (command) to be performed by the glider. An example mission file is shown in Figure1(b). Behaviors are listed in descending priority in the mission file. In our example, the *abend* behavior has the highest priority. This behavior will be active if the glider needs to resurface due to some software or hardware failure. It is of highest priority since it may prevent the total failure and potential loss of the glider. Other behaviors specify the diving pattern, set of active sensors, and the overall heading of the glider (waypoints).

In non-emergency operation, the current programming model does not allow the glider itself to change its behavior based on the sensor readings that it has collected. To change its behavior, a new set of parameters for the existing mission, or a new mission file has to be downloaded from the remote control center. If the mission change is based on sensor readings, a representative subset of these readings (which may potentially be very large) has to be communicated to the control center via the satellite phone connection. This can be a rather costly operation, both in terms of power/energy and time. Every time the glider surfaces, approximately 20 minutes are spent at the surface to communicate a representative data subset back to mission control via the satellite phone, and to receive new instructions from the control center. The glider typically surfaces every 3-5 hours. Once a new mission has been received, it is "recompiled" and installed in the glider. If only new parameters are received for the existing mission, no recompilation is necessary. This overall programming paradigm for the Slocum glider is illustrated in Figure 1(b). Although a highly successful research platform, the current programming architecture will not be able to satisfy the increasing demands of marine scientists for more sensors, more data storage, a better programming model and tools to support such a model, more flexibility in controlling the gliders, and better mechanisms and tools for resource management.

Limited Programming Model and Tool Support The existing programming model is rather limited and error-prone. Typically, scientists do not write their own behaviors, but build their missions from pre-existing behaviors with parameters set to values that reflect the particular new mission goals. Parameter values are typically interdependent, which is not obvious and requires an in-depth understanding of the software and hardware control systems of the glider. Therefore, scientists typically limit themselves to modifying existing missions that work, and adapt these missions to their new mission goals, resulting in a trial-and-error programming approach. Debugging support, or help with assessing the overall impact of the selected mission characteristics on the expected resource usage, in particular battery life, is very limited. Mission characteristics include the set of active sensors, the flight path profile (glider angle and speed), data logging requirements (frequency, volume, and sampling rate), and communication (satellite phone, underwater acoustic modem).

Need for More Sensors, More Data Storage, and QoR Specifications Marine biologists and oceanographers have an ever increasing demand for new sensors, including optical and chemical sensors. These sensors consume a significant amount of energy, and require a large amount of non-volatile data storage currently not available on the gliders. On the Slocum, much of the acquired data has to be communicated through a serial RS-232 connection which severely restricts the amount of data that can be logged. Not all sensors can be active all the time since this would significantly tax the battery resources. Scientists have to be able to express quality-of-result (QoR) tradeoffs with respect to power and energy constraints. For example, it may be better to gather readings of an energy-expensive sensor in bursts, instead of evenly spreading out the readings over a particular time interval, assuming that the sensor readings cannot be performed all the time. Such information has a semantic content that cannot be derived by any "smart" compiler or operating system, therefore requiring the programmer, in our case the marine scientist, to provide the appropriate specifications. A similar need for QoR tradeoffs are encountered in dynamic networks of mobile devices, such as smart phones. We already have some experiences with such QoR specifications in the context of our NSF funded collaborative Sarana project [6] and SpatialViews project [11, 10].

Insufficient Dynamic Control The glider should be able to react to dynamic phenomena in the oceans, which is not sufficiently supported in the current Slocum programming architecture. To change its behavior, the glider has to surface, contact the control center, and either download new mission parameters, or a new mission file. This procedure has significant overhead, both in terms of time and battery power. If the phenomena is highly dynamic and has only a small feature size, such as a cross-shelf intrusion event [9, 2, 16], the glider may not be able to find it again.

No Underwater Communication Underwater communication can enable many important glider capabilities, including underwater navigation and swarming behavior. Groups of gliders may support each other by sharing sensor information, allowing such groups to be treated as a single instrument, but with different individual sensor sets. Communication can also be used to support an underwater location service, which is crucial for under-ice exploration where resurfacing and getting a GPS position is not possible. Acoustic communication modems, such as WHOI's Micro-Modem, can be deployed within the Slocum glider [5]. We will get access to at least two glider mounted acoustic modems soon, allowing us to begin experimentation with communication optimizations.

Limited Autonomous Operation The current operation model requires the gliders to surface and connect to the mission control center every three to twelve hours. Substantial amounts of sensor data are uploaded to the control center to allow scientists, or a control program (the *Dock-Server* [13]) to analyze the data and perform corrective actions, if necessary. Much of this data exchange could be saved if the glider had sufficient data processing capabilities to analyze its sensor data on-board. However, global connectivity is still crucial for providing the glider with context information that it cannot derive by itself, such as an approaching bad weather front, or a nearby Gulf Stream eddy that it could use to get a "free ride".

In some cases, a more autonomous behavior is important for the survival of the glider itself. In March of 2008, the Iridium satellite phone [8] connection was disrupted for 5 days, leaving gliders without contact to the control center. The glider went into a holding pattern, flying between two fixed waypoints while trying to establish a satellite connection approximately every 3 hours. Luckily, no glider ran out of battery (in which case it would just drift at the surface), drifted into any shipping lanes, or came close to a shore line, all of which may have resulted in a total loss. A more intelligent, autonomous glider would have been able to better deal with such an emergency situation. Finally, for under-ice arctic exploration, autonomous behavior is vital since communication to the control center is not possible and the glider needs to rely on its own capabilities.

	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

Table 1: Our current single-board computer systems (SBC) that can be installed in the Slocum Glider.

3 New Programming Architecture

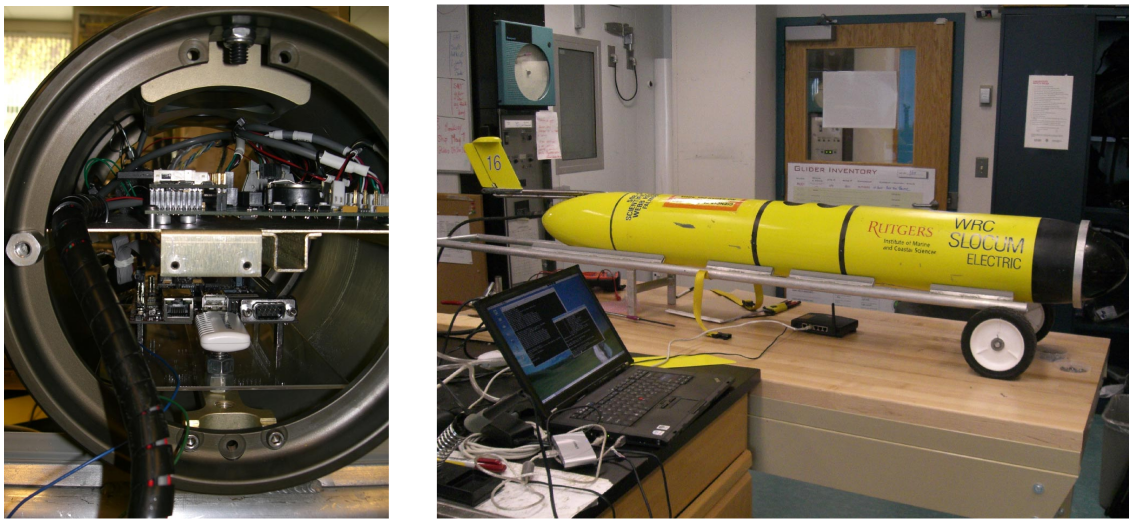
Clearly, in order to enable any significant change to the current programming architecture, new computer hardware is needed as part of the glider infrastructure.

We chose an ARM-based (TS-7800) and an x86-based (TS-5500) Linux board as our initial development infrastructure alternatives [14]. Table 1 contains the hardware specifications of both single board computers. The reported power measurements were taken under load using a Tektronix TDS 3014 Oscilloscope. These single board systems have several serial and USB interfaces for sensors and run a commodity operating system that provides for rich driver support as well as an easy development environment. Depending on the sensor, power, logging and processing requirements, it may be advantageous to choose one board over the another. For example, floating point operations are significantly faster on the x86, while the ARM board has more RAM and more serial interfaces. Other single board systems will likely be considered in the future depending on the particular needs of the sensor sets or mission objectives.

Figure 2(a) shows the ARM board within the glider's payload bay, mounted below one of the Motorola-based glider systems, called the "science persistor". The second Motorola board, the "flight controller", is responsible for controlling most of the glider, with the exception of a few sensors which are controlled by the science persistor. The flight controller and the science persistor are connected via a serial RS-232 interface.

One key objective of our overall design is to enable a new programming architecture without significant changes to the underlying, existing hardware/software infrastructure. The main motivation for this decision is two-fold. First, the existing system is well designed to deal with basic safety issues through the layered control system that ensures the physical integrity of the glider. Our new architecture will have control over the glider only when these safety issues are not violated. For example, the underlying system handles hardware failures, water intrusions, and some form of obstacle avoidance, and ensures that the glider can be recovered in any severe failure situation. This safety feature is crucial since each glider costs between $150K - $200K, depending on the sensor configuration. Secondly, the development and testing of the existing glider system took tens of "man years", resulting in a robust commercial product with many customers. Being backward compatible with respect to the existing system will allow current users to gracefully migrate to our new system without loosing the investment into their existing "mission files".

Our initial design that satisfies the above objective is shown in Figure 3. We wrote a "dummy mission" file that defines a new behavior as part of a layered control system. This new behavior ("hookbeh") dynamically generates new subbehaviors triggered by values in the "sensor array" data structure. The sensor array is a key data structure of the existing layered control system and can be considered the data memory of the flight controller. Our new architecture extends the sensor



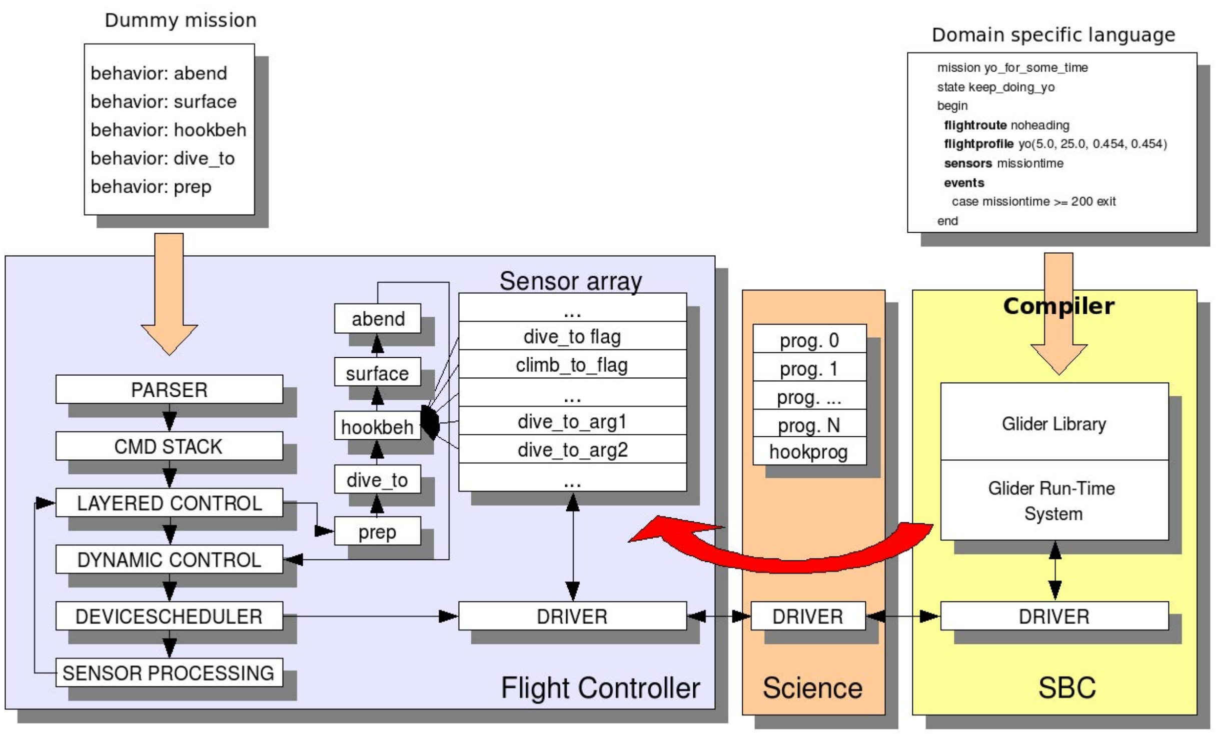
(a) An ARM-based TS-7800 Linux board. The new board is mounted in the lower half of the payload bay.

(b) Sealed and decompressed glider in a "bench-top" flight configuration. The payload bay is located in the middle section of the glider.

Figure 2: New single-board Linux system and our bench-top testbed.

array. If a subbehavior flag is set, our "hookbeh" will dynamically insert a corresponding new behavior into the existing layered control stack using parameter values stored in predefined fields in the sensor array. The particular selection of subbehaviors that can be triggered can be thought of as the new instruction set architecture (ISA) of the glider. Once "executed", the subbehaviors are removed. During each execution cycle (approx. 4 seconds), multiple subbehaviors may be created and executed. From now on, we will refer to the new single-board computer as SBC. It has control over the glider by reading from, and writing instruction flags and parameters into the sensor array. The particular selection of possible subbehaviors, i.e., the particular ISA may be application dependent and can be tailored and specialized for given mission characteristics. Since these new instructions together with their parameters have to be communicated from the SBC to the flight controller via a rather limited serial link, we will investigate efficient instruction/ISA encodings, and parameter setting optimization strategies. The existing sensor array is implemented as an array of double precision numbers. Using a double precision entry for each instruction flag is rather expensive due to memory space and necessary communication bandwidth for setting the flag over the serial link. In addition to ISA compression and specialization, we are currently investigating optimizations that preset parameters, or reuse parameters across different invocations of the same subbehavior, i.e., execution of the same instruction. Our new runtime system executing on SBC targets this new ISA.

The proposed new programming architecture allows the execution of existing "old" mission files by running such a mission file instead of our "dummy mission". Another advantage of this approach is that we may define different dummy missions that wrap or package our "hookbeh" with different higher-priority behaviors that may limit or overwrite particular glider actions.



```
mission yo_for_some_time
state keep_doing_yo
begin
  flightroute noheading
  flightprofile yo(5.0, 25.0, 0.454, 0.454)
  sensors missiontime
  events
    case missiontime >= 200 exit
end
```

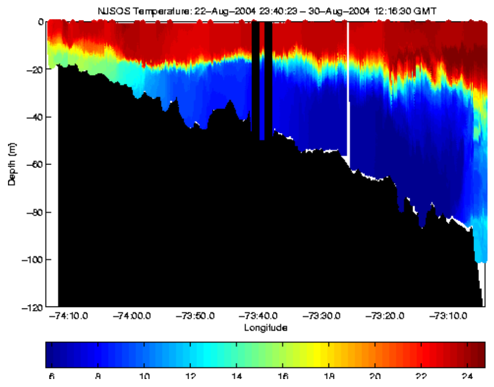
Figure 3: The new programming architecture; from left to right: flight controller with skeleton mission and layered control implementation; science computer; SBC, our new Linux board, with a domain specific programming language, runtime system, and compiler; SBC is connected to the flight controller via a RS-232 link through the science persistor.

3.1 Domain Specific Language

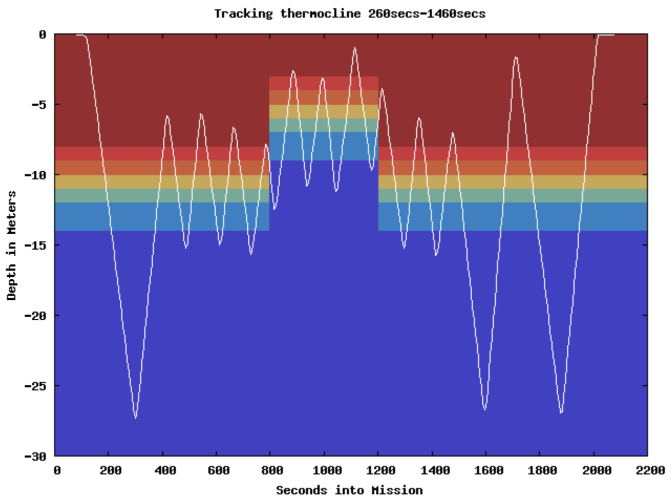
A good domain-specific language design provides programming abstractions that allow marine scientists and oceanographers to express their programs with abstractions that they easily understand and can reason about. At the same time, these abstractions should allow the compiler to catch programming errors as early as possible, and to generate efficient code for our new Slocum ISA. The usability and expressiveness of the language is currently being evaluated in close collaboration with Dr. Glenn's and Dr. Schofield's research groups at the Institute for Coastal and Marine Sciences at Rutgers University.

The initial language proposal combines the notion of a finite state machine, spatial regions, and layered control. The specification of QoR tradeoffs are not yet included. Figure 5 shows a program sketch of a thermocline tracking application. A thermocline is a layer of water where the water temperatures change dramatically. Figure 4(a) shows the thermocline off the coast of New Jersey. The shown data has been collected by a Slocum glider (not equipped with our new infrastructure). A desired tracking behavior is illustrated in Figure 4(b). Instead of flying through the entire water column, the glider should track the thermocline as it changes. This tracking behavior, or any dynamic adaptation of the glider behavior in response to a changing sensor value, is not supported by the current Slocum programming architecture.

The thermocline tracking program consists of spatial declarations, followed by definitions of behaviors. The spatial declarations represent the convex hull of all points listed in way-point list. At this point, a way-point may be specified by a GPS location. The `avoid` keyword is followed by regions



(a) Actual measured thermocline off the coast of New Jersey by a non-tracking glider.



(b) Simulated tracking of thermocline. The thermocline is is an approximation of the data in the narrow region between the two black lines in the subfigure to the left.

Figure 4: Example: Thermocline tracking application.

that should not be visited by the glider. If the glider happens to be in such a region, normal glider operation will be stopped, and the glider will try to leave the region. If successful, normal glider operation will be resumed. During normal glider operation, the glider is in a particular state. In each state, the glider has a particular flight destination (`flightroute`) and a particular flight profile to get it to the destination (`flightprofile`). In the thermocline example, the glider can be either in a state where it looks for a thermocline (`TargetArea`), is tracking a thermocline (`ThermoTracking`), or needs to contact the control center (`contactControlCenter`). Transitions between these states are specified within events. Events may depend on sensor values. In the example, all sensors are capitalized. The new glider runtime system evaluates events periodically. If an event is recognized, a state transition may happen as specified in the action body of the event, either as a regular transition (`goto`) or as a nested transition (`call`). Subbehaviors form a behavior stack. State transitions from subbehaviors can be initiated by a `return`, which returns the execution to the behavior that invoked the current behavior as a subbehavior. In our example, if the ThermoTracking behavior loses the thermocline, it returns to the behavior `TargetArea` that searches again for the thermocline. The behavior `contactControlCenter` establishes contact with the mission control center. It is typically periodically invoked as specified by the user (example: `yo.surface(180)` in behavior `TargetArea` , or invoked as a subbehavior if an error condition arises.

Current Prototype Implementation

One of the main challenges in the implementation of the new programming architecture was the need for backward compatibility with the existing programming environment. As a result, all desired glider actions are expressed by generating new behaviors in the existing layered control system, and by reading and writing new fields in the sensor array. The layered control system and sensor array both live on the flight controller. We wrote a new behavior ("hookbeh") that is expressed within the old programming system as part of a mission file. Its job is to create and delete new subbehaviors

```
mission ID M01-April2008-TrackThermalBoundary;

regions      regionTarget=(<way-point1>,<way-point2>, <way-point3>, <way-point4>)
             restrictedArea=(<way-point5>,<way-point6>, <way-point7>)
             shippingLane=(<way-point8>,<way-point9>, <way-point10>, <way-point11>)

avoid regions: restrictedArea, shippingLane

mission plan: startWith TargetArea(regionTarget)

state TargetArea(Region regionT) begin
behavior     flightroute − system.searchSweep(regionTarget);
             flightprofile − yo.dive(2, 200, 26, 26); yo.surface(180); // surface every 180 minutes
sensors      CDT = CDT.setParms(0.5, lazy(5)); // every 0.5 secs, lazy log every 5-th reading
             LOC; TIME; SUR;
events
             case SUR.atSurface():
                       if (!sufficientProgress(LOC.getLocation(), TIME.getTime(), regionT) {
                                                               call contactControlCenter( ); }
             case CDT.getTemperatureHistoryVariance(20) > 3 { goto ThermalTracking(3) };
end

state ThermoTracking(int temperatureDifference) begin
behavior     flightroute − system.trackCondition( );
             flightprofile − trackThermalBoundary.track(temperatureDifference); trackThermalBoundary.surface(120);
sensors      CDT = CDT.setParms(0.5, lazy(5)); // every 0.5 secs, lazy log every 5-th reading
             LOC; TIME; SUR;
events
             case SUR.atSurface(): return(); // allows system to update time and location (GPS)
             case CDT.getTemperatureHistoryVariance(20) <= 3 { return() };
end

state contactControlCenter begin
behavior     flightroute − system.getToSurfaceAndStay( );
             flightprofile − {yo.getToSurface()}; yo.floatAtSurface();
sensors PHONE = PHONE.connect();
events
             case PHONE.connected() {
                       char *commBuffer = PHONE.receiveData(...)
                       // receive commands and follow pre-defined communication protocol
                       return();
             case PHONE.timeout() {ABORT_MISSION}
end
```

Figure 5: Program sketch of thermocline tracking.

within the layered control system based on values in the sensor array set by the new SBC. A new driver layer had to be implemented that allows the SBC to read and write sensor values. The SBC is connected to the science computer via a 115200 baud serial connection, which in turn connects to the flight controller via a 9400 baud serial interface. The necessary new driver layer is the main bottle-neck of the current implementation resulting in read/write delays of sensor values of up to 8 seconds.

The basic runtime system that executes on the SBC implements a stack-based finite state machine as discussed in Section 3.1. The entire runtime system is written in C. It periodically evaluates pre-conditions of actions stored as function pointers in an event-list data structure. If an event pre-condition is satisfied, the corresponding actions are executed. Such actions typically include calls to the glider runtime library. The glider runtime library provides abstractions for state changes (e.g.: system.goto(), system.call(), system.return(), system.abort()), higher-level glider behavior (e.g.: yo.dive, yo.getToSurface()), and sensor operation (e.g.: CDT.setParms(), CDT.getTemperature()). New sensors, including communication, can easily be integrated into our new programming architecture by extending the glider runtime library.

We have implemented a limited subset of the domain specific language outlined in Figure 5, a basic glider runtime library, and support for complex missions such as a thermocline tracking application. The lex/yacc based compiler translates the domain-specific language program into an abstract syntax tree (AST) representation. In the code generation phase, the compiler generates a specialized version of the runtime system from the AST. The resulting C program is compiled using the standard gcc tool chain. When the generated executable is invoked on the SBC, it takes over the control of the glider.

Experimental Results

For our experimentation, we used a “shoe-box” system and an actual glider with our SBC inside. The shoe-box is a simulation environment that contains the two Motorola based computer systems of the glider, namely the flight controller and science computer. Both are connected via a serial RS232 interface. Navigational motors (buoyancy pump, pitch motor, fin motor) and sensors with their readings are simulated in this environment. The shoe-box system allows us to debug our new programming architecture without using an actual glider. When our SBC is deployed within the glider, it receives its power from the glider’s battery banks. All sensors are active and the navigational motors can be operated. This requires that the entire glider is decompressed to an airpressure of 6 psi. This means that no cables can be used to connect to our SBC from outside the glider. Instead, we use a wireless 802.11 connection to log onto our SBC in the bench-top testing configuration. Some sensor readings still need to be simulated in the bench-top environment, such as depth and water temperature. Other sensors are active, but produce “bogus” data since they are designed to take measurements in the water.

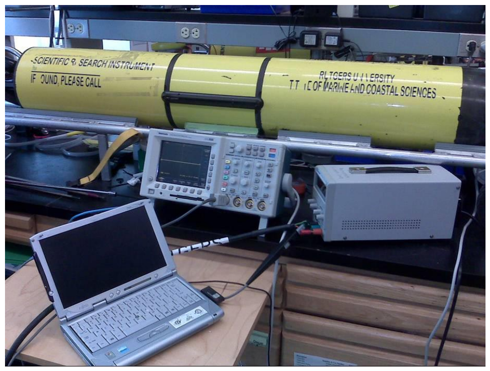
A basic thermocline program was written in our new domain specific language and compiled using our prototype compiler. The resulting executable was installed on the SBC and tested in a bench-top, sealed and decompressed glider as shown in Figure 2(b). For the experiment, our thermocline program was activated after 260 seconds into the mission, and disabled after 1460 seconds. The flight profile during the first 260 seconds, and after 1460 seconds are that of a non-tracking glider. The experimental results are shown in Figure 4(b). Our thermocline program was successfully compiled, and executed on the new SBC allowing the glider to do something that the current programming architecture does not support.

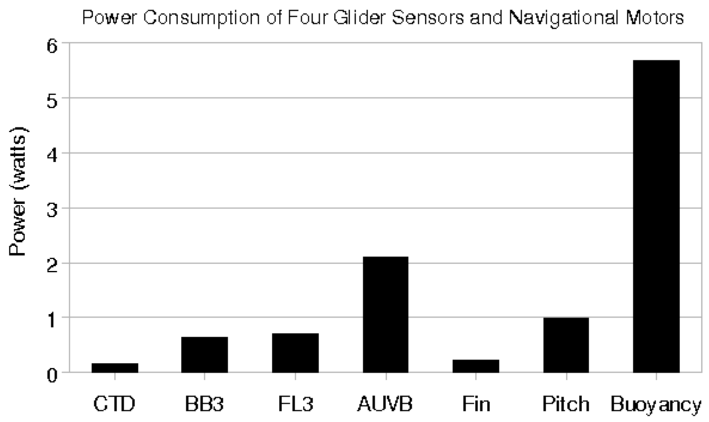
3.2 Resource Awareness

Effective resource management, in particular battery power, is of crucial importance in our glider environment. Figure 6 shows the average power dissipation of several common sensors together with the navigational motors of the glider. The presented numbers were obtained through actual physical measurements using our Tektronix TDS-3014 oscilloscope. Since the glider needed to be sealed and decompressed for the measurements, only whole system measurements were possible. The reported figures are differential power measurements relative to a baseline power level of the system in an idle state. In addition to power issues, the 9400 baud serial connection between the flight controller and science computer is a crucial resource and performance bottleneck. All these resources need to be managed effectively in order to ensure that the glider can reach its full potential as a scientific instrument.

It may not be possible to run all sensors all the time, or to use a flight path that requires the buoyancy pump and pitch motor to be activated frequently, in particular at high depths. The power consumption of the buoyancy pump may vary by up to a factor of six for the same amount of water displacement depending on the water pressure at different depths. Techniques are needed that will balance the overall energy requirements across multiple sensors, selecting a flight path and sensor reading frequency that optimizes the expected benefit of the sensor readings. This will require input from the user via QoR language specifications.

In order to save communication costs and costs due to frequent surfacing, we will investigate the benefit of data set analysis techniques that will only require contact with the control center if an unexpected situation is encountered. Typically, data sets of the order of 20K to 100K bytes are communicated to the control center after a diving phase lasting between three and twelve hours.


(a) Power Measurement Setup.



(b) Physical Power Measurements

Figure 6: Measured average power of a set of glider sensors and glider navigation equipment. From left to right: CDT - conductivity, temperature, depth; BB3 - bio-optical sensor to measure water clarity; FL3 - flurometer to test for chlorophyll-a, uranine, or rhodamine; AUVB - Fluorometer (laser); Fin - fin motor; Pitch - pitch motor; Buoyancy - buoyancy pump.

By allowing the on-board processor to make the decision what to do next instead of the remote control center, the energy required for communication and for surfacing can be significantly reduced. We physically measured the average power needed to upload different file sizes from the glider to the contol center via the Iridium satellite phone link. For all file sizes, the average power was approximately 6 W with a transmittion rate of 180 bytes/second. The transmission times for files of size 20K and 100K were a little less than two minutes and around ten minutes, respectively.

Since the energy consumption of the buoyancy pump and some sensors (e.g.: the laser in the AUVB fluorometer needs to be kept at a fixed temperature) depend on environmental conditions such as water depth and water temperature, an accurate power/energy management strategy will need on-line power measurements. Both of our current SBC alternatives have an analog to digital capture function by way of an on-board AVR microcontroller. The voltage measurement resolution on the TS-5500 is 12 bit, and 10 bit on the TS-7800. On the TS-7800, this translates to a voltage measurement resolution of 3 mV. We are currently in the process of instrumenting the power supply of the buoyancy pump with a 0.1 Ohm current sensing resistor. In addition to the voltage drop at this resistor, we will measure the overall supply voltage of the battery. An actual glider deployment with our power measurement infrastructure is scheduled within the next month.

4 Future Work

We have currently scheduled boat and crew time for one glider at the end of July to test our thermocline application and to perform on-board power measurements of the buoyancy pump. The glides will be deployed in 30m of water approximately 30km off the shore of New Jersey. This will be the first real test of our new programming architecture. Other goals in the near future are the extension of the current compiler to handle more of our proposed language features and to extend the glider library to support acoustic underwater communication and some basic form of swarming behavior.

Acknowledgements

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.

References

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

[2] P.E. Biscaye, C. N. Flagg, and P. G. Falkowski. The shelf edge exchange processes experiment, SEEP-II: An introduction to hypotheses, results and conclusions. *Deep-Sea Research Part II*, 41:231–252, 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] L. Freitag, M. Grund, S. Singh, J. Partan, P. Koski, and K. Ball. The WHOI micro-modem: An acoustic communications and navigation system for multiple platforms. In *IEEE Oceans Conference*, Washington, D.C., 2005.

[6] P. Hari, K. Ko, E. Koukoumidis, U. Kremer, M. Martonosi, D. Ottoni, L-S. Peh, and P. Zhang. SARANA: Language, compiler, and runtime system support for spatially-aware and resource-aware mobile computing. *Philosophical Transactions of the Royal Society A*, 2008. to appear.

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

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

[9] M. S. Lozier. and G. Gawarkiewicz. Cross-frontal exchange in the middle atlantic bight as evidenced by surface drifters. *Journal of Physical Oceanography*, 31:2498–2510, 2001.

[10] Y. Ni. *Programming Ad-Hoc Networks.* PhD thesis, Department of Computer Science, Rutgers University. Technical Report DCS-TR-594, January 2006.

[11] Y. Ni, U. Kremer, A. Stere, and L. Iftode. Programming ad-hoc networks of mobile and resource-constrained devices. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'05)*, Chicago, IL, June 2005.

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

[13] 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.

[14] Technology Systems. http://www.embeddedarm.com. Foutain Hills, AZ.

[15] K. Valavanis, D. Gracanin, M. Matijasevic, R. Kolluru, and G. Demetriou. Control architectures for autonomous underwater vehicles. *IEEE Control Systems Magazine*, 7(6):48–64, December 1997.

[16] P. G. Verity, J. E. Bauer, C. N. Flagg, D. J. DeMaster, and D. J. Repeta. The ocean margins program: an interdisciplinary study of carbon sources, transformations, and sinks in a temperate continental margin system. *Deep-Sea Research Part II: Topical Studies in Oceanograph*, 49:4273–4295, 2002.