

U.S. copyright law (title 17 of U.S. code) governs the reproduction and redistribution of copyrighted material.

Rutgers  
L

Staff View **Display**

Searching

Resource  
Sharing

My Account | Options | Policies Directory | Comments | Exit

Request Manager

Blank Workform

Printing

Batch

My Requests

Go to page



Admin  
Link



LHR  
Update



Print



Help

SHIPPED - Lender

Edit

GENERAL RECORD INFORMATION

Request Identification:	<b>29518092</b>	Status:	SHIPPED
Request Date:	20070404	Source:	FSILLSTF
U.S.C. Number:	57342733		
Borrower:	<b>NJR</b>	Need Before:	20070504
Receive Date:		Renewal Request:	
Due Date:	<b>N/A</b>	New Due Date:	
Lender:	<b>*AAF</b>		
Request Type:	Copy		

BIBLIOGRAPHIC INFORMATION

Call Number:

Author: Lee, Michael J.; McGhee, Robert; Workshop on Mobile Robots for Subsea Environments 1994 : Monterey, Calif.)

Title: Mobile robots for subsea environments : IARP 2nd Workshop /

Imprint: [Monterey, CA] ; Pacific Grove, Calif. : International Advanced Robotics Programme ; May be obtained from Monterey Bay Aquarium Research Institute, 1994

Article: Mission Configuration using layered control by J. G. Bellingham and J. J. Leonard

Pages: 193-202

Verified: WorldCat Desc: ix, 232 p. : Type: Book

BORROWING INFORMATION

Patron: REGION I

Ship To: INTERLIBRARY LOAN SERVICES/ALEXANDER LIBRARY/RUTGERS UNIVERSITY/169 COLLEGE AVE./NEW BRUNSWICK, NJ 08901-1163

Bill To: same

Ship Via: UPS, COMET (NJ), METRO

Electronic  
Delivery:

Standard Cost: IFM - \$20.00

Copyright  
Copyright: CCL

Email: macalukr@rci.rutgers.edu



IARP

2nd Workshop

International Advanced Robotics Programme

---

MOBILE ROBOTS  
FOR SUBSEA ENVIRONMENTS

# TASK CONFIGURATION WITH LAYERED CONTROL

James G. Bellingham and John J. Leonard

Underwater Vehicles Laboratory  
Massachusetts Institute of Technology Sea Grant College Program  
292 Main Street, Cambridge, MA 02142 USA

## ABSTRACT

A variant of layered control designed for ease of task configuration has been developed. Intended to be used as the middle level of a three level architecture such as state-configured layered control, the approach embodies a number of features designed to facilitate configuration of the vehicle software to achieve specific tasks. Examples include moving the vehicle to a defined location or executing a survey grid. While not intended to provide the entire mission planning level for a vehicle, it can provide that function. Implemented on the Odyssey II autonomous underwater vehicle, it has been used as the mission planner for under-ice vehicle operations in lake Winnepesaukee and the Arctic.

## INTRODUCTION

Layered control is one of a variety of architectures which have been employed for mission level control of mobile robots. Implemented originally on land robots, it was used to solve relatively low-level problems such as coordination of limbs on a walking robot [Brooks, 1986]. In subsequent work, it was used to embed mission level control in mission-dedicated vehicles [Loch, 1989; Connell, 1990]. Extension of the architecture to mission reconfigurable applications highlighted the subtlety of programming layered control [Bellingham et al., 1990] and led to the development of hybrid architectures such as state-configured layered control [Bellingham and Consi, 1991] and SSS [Connell, 1992]. This paper focuses on programming missions within layered control.

Layered control is more of a programming philosophy than a rigid set of guidelines. There are nearly as many implementations of layered control as there are robots which use it, and the nature of the implementation drastically changes the ease of use and flexibility of the software. The central element of the work described here is definition of a clear "interface" for configuring a layered control structure. This involves specifying which behaviors are active, what their priority should be, and setting the values of their arguments. Using this interface, a higher level entity can configure the vehicle for specific tasks. Candidates for the higher level include a human or the state-configuration level of state-configured layered control.

In state-configured layered control, the top level of the vehicle architecture is a state table which is responsible for stepping through the discrete mission phases. Examples of such phases include: transit, grid survey, and homing. To accomplish the goals of a given phase, the state-configuration level arranges the layered control structure by activating the behaviors with the appropriate priorities and argument values. In essence, the state configuration level specifies tasks for the layered control level to achieve. The layered control interface thus provides the medium for such task specification.

Definition of the layered control interface also addresses a second important issue: structuring code to support future augmentation. In principle, layered control is attractive for the incremental way in which a layered control program can be assembled. However, a difficulty encountered at MIT Sea Grant was that relatively common modifications, such as adding a new sensor variable or adding a behavior, propagated changes through the vehicle code. Discrete elements of the code include: I/O, sensor processing, dynamic control, high-level control, and data logging. In addition, data analysis software, mission configuration tools, and vehicle simulators are intimately connected to

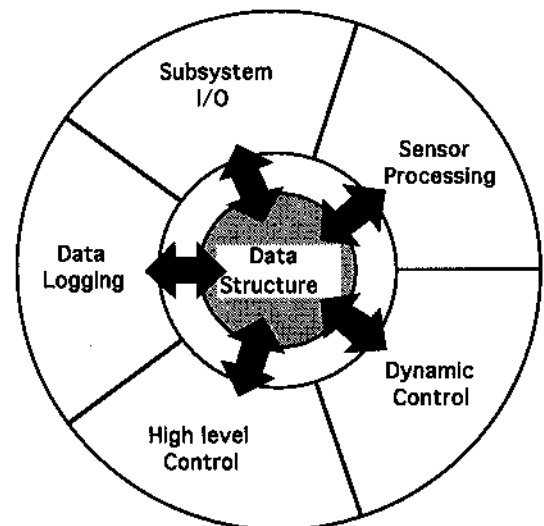


Figure 1: The control software is built around a carefully defined data structure, through which all the vehicle code elements interact.

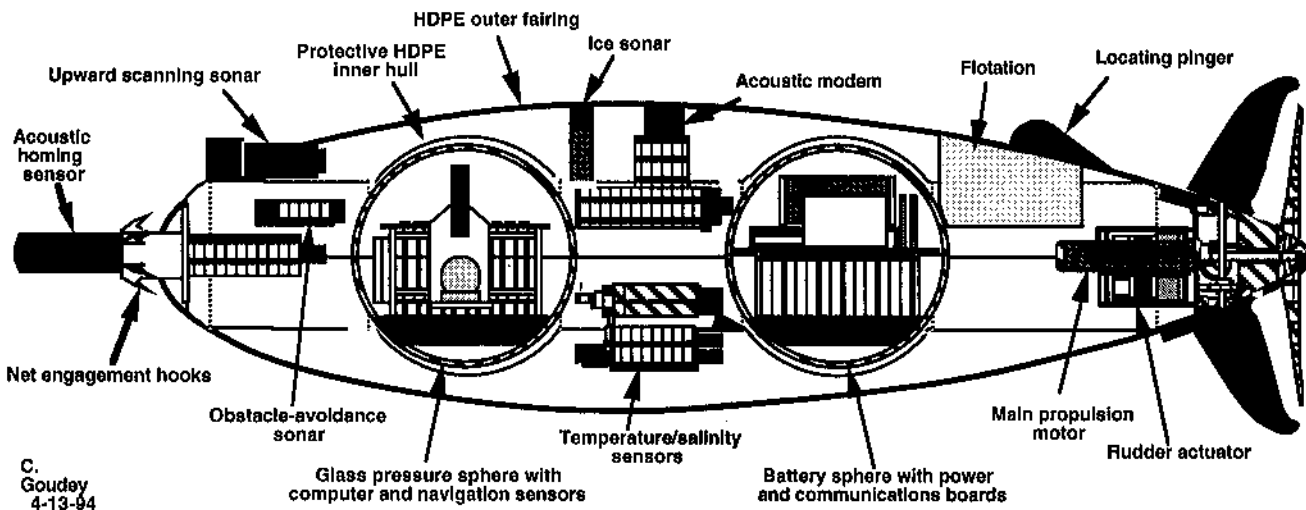


Figure 2: Arctic configuration of Odyssey II. Mission sensors are the scanning sonar at the nose, and the temperature and conductivity sensors located in the center of the vehicle.

the vehicle code. Addition of a new behavior might require changes in nearly all these software elements.

The solution to both the issue of defining a layered control configuration interface, and to designing code for expandability, lay in the creation of a data structure. The structure contains the variables describing the vehicle state: input, output, and adjustable settings. In addition, the structure contains the names and time-stamps for variables. Addition of sensors or behaviors are documented in the data structure, and such changes are propagated to all the code elements on recompilation. Thus the data structure provides several important functions: it provides a single location containing all available information on the vehicle state, it provides a clearing-house for code additions, and it provides a rudimentary but extremely important level of documentation of state variables.

## HARDWARE AND OPERATION

The software described in this paper has been implemented on an autonomous underwater vehicle, and employed in preparation for a number of well defined science survey missions. The vehicle, its navigation systems, and the missions for which it is designed are described below.

### Vehicle Description

Odyssey II is second generation survey class autonomous underwater vehicle, designed as an intelligent mobile instrument platform with deep-water capability [Bellingham et. al, 1992]. Comprised of a low-drag fairing with a single ducted propeller and cruciform control surfaces, Odyssey II is 2.2 meters long, and has a maximum diameter of 0.6 meters (see Figure 1). The fairing is free-flooded, and contains the main pressure housings, which are two glass spheres. In the present configuration, the vehicle has an endurance of eight to

twelve hours, depending on the operating speed and the duty cycle of subsystems such as the acoustic modem.

The primary onboard computer is built around a Motorola 68030 microprocessor [OR Industrial Computers]. In addition to the main computer, a network of small microcontrollers (Motorola M68HC11) is used to distribute "intelligence" to sensors and actuators. The present vehicle state sensor complement includes a three-axis fluxgate magnetometer [EMDS], three-axis accelerometer and angular rate sensors [Systron-Donner], a pressure transducer [Paroscientific], and a water speed sensor. Two ST500 obstacle avoidance sonar and an ST200 altimeter sonar [Tritech] are also integrated into the vehicle. An acoustic modem [Datasonics] has undergone preliminary testing on Odyssey II, and will become a standard device.

For Arctic operations, Odyssey II had no drop weights. However, two drop weights are attached for deep-water operations. The first is used to speed decent of the vehicle to its operating depth, while the second is used to speed ascent in a normal mission, or to force ascent in an emergency. The drop weights can be released either by command from the main vehicle computer, or on the initiative of a second "watchdog" computer which monitors the main vehicle computer and the progress of the mission. The watchdog operates as a backup to the main computer, to ensure vehicle recovery when all else fails.

A transponder, a radio beacon, and a strobe are used to provide a means of locating the vehicle. All three location aids operate off of power sources independent from each other and the rest of the vehicle electronics to ensure operation even if vehicle batteries run low. The radio beacon and strobe are used for locating the vehicle on the surface. An ultrashort-baseline (USBL) system is used on the surface to track the transponder. Both LXT and

Trackpoint II systems [ORE] have been used to track Odyssey II from the surface.

### Navigation

A commercial USBL system, the LXT [ORE], was integrated into the vehicle to provide a homing capability for under-ice operations. The LXT measures both direction and range to up to two acoustic beacons. Detection of beacons at ranges on the order of two kilometers was obtained in the Arctic. While power consumption of the system is high (40 watts), the USBL need only be turned on for the recovery phase of the mission. While vehicle position can be determined from range and bearing to one beacon, the ability of the vehicle to track two beacons at a time also allows for a range-range solution for the vehicle position (i.e. spherical navigation).

Two different long-baseline navigation systems are available for Odyssey II, both hyperbolic in nature. The first system, originally used on Sea Squirt, employs several beacons: one pinger and two or more transponders [Bellingham et al., 1992]. The pinger triggers the transponders at regular intervals, and the subsequent detections at the vehicle constrain the vehicle position to hyperboloids. The intersection of the hyperboloids combined with knowledge of the vehicle depth allow one to establish vehicle position in a defined operating region. This system operates between 26 and 30 kHz. Attractive features of this system include minimal power consumption on the vehicle, less than 0.1 W, and the potential to support multiple vehicle operations.

The second long-baseline navigation system, also hyperbolic in nature, operates at lower frequencies (8-12 kHz) with beacons synchronized by GPS. This system, first tested in the Arctic, is being developed for use in high multipath environments. The technique utilizes all the detectable arrivals instead of just the first one, as is typical for existing long-baseline navigation systems. The additional information contained in these multipath arrivals offers improved position accuracy and enables tracking of changes in the environment to maintain accuracy over the course of an extended mission [Deffenbaugh, 1994].

### Operational Scenarios

The software is design to provide a framework to support a range of mission capabilities. These missions motivate the basic software architecture, and are reviewed below.

*Arctic Ice Mapping:* In this scenario, the vehicle is deployed to an Arctic ice-camp and held ready for an event [Bellingham et. al., 1993]. To provide navigation over a ten kilometer radius from the camp, a long-baseline network of acoustic communication and navigation beacons is deployed from the ice sheet. When ice activity is detected acoustically, the vehicle is launched through the ice in the general direction of the activity. At the same time, a helicopter is dispatched to instrument the region

undergoing ice activity. The team transported by the helicopter determines the exact area to be surveyed and transmits that information to the camp, which acoustically communicates the survey area location to the vehicle. The vehicle surveys the under-side of the ice with a mapping sonar and then returns to the camp. Recovery of the vehicle is achieved by having the vehicle home on an acoustic beacon at the launch point.

Odyssey II has been employed for operations under ice during 1994, both in Winnepesaukee, and in the Arctic.

*Volcanic Event Response:* In this mission, the vehicle is held ready at a location convenient to the Juan de Fuca Ridge, perhaps Seattle, Washington. When an event, which may or may not be a volcanic eruption, is detected acoustically via the SOSUS network, the vehicle is loaded on a ship of opportunity (or perhaps an aircraft) and transported to the area. The vehicle is launched with a drop weight for rapid descent to the bottom. The initial mission objective is to profile the water column vertically from the sea floor up to about 500 meters over the sea floor to detect and localize the temperature and optical anomaly which would be created by an eruption. This data is transmitted to the surface via an acoustic modem. When an anomaly is found, the scientists on the surface vessel can command the vehicle to take still images of the sea floor in the vicinity of the suspected eruption. Return of the vehicle to the surface for recovery is assisted by release of a second drop weight.

The principle sensors for the volcanic event response are conductivity, temperature, optical backscatter, and a still imaging system. While some data can be transmitted to the surface via the acoustic modem, it is expected that the bulk of the data will be dumped from the vehicle after recovery.

Odyssey II operations on mid-ocean ridges are scheduled for later in 1994, and in summer of 1995.

*Autonomous Ocean Sampling Networks:* In the long term, the most exciting operational scheme is afforded by the Autonomous Ocean Sampling Network (AOSN) concept [Curtin et al., 1993], in which moored buoys provide power and communication nodes to provide a long term, multiple vehicle presence in the ocean. The objective is to provide an economically feasible capability for repeated synoptic characterization of large scale oceanographic phenomena such as meso-scale eddies or ocean fronts. The key to such a system is a small, low-cost autonomous vehicle which can be operated reliably over extended unattended deployments at sea.

## CONTROL SOFTWARE IMPLEMENTATION

As is becoming increasingly popular for control of mobile robots, state configured layered control is a three level structure. At present, the two levels which are implemented are dynamic control and layered control.

Table 1: Commands Accepted by Dynamic Control Level

Command Mode	Type of Control Desired
Vertical mode 0	direct command of rudder position
Vertical mode 1	desired depth
Vertical mode 2	desired pitch
Horizontal mode 0	direct command of elevator position
Horizontal mode 1	desired heading
Thrust mode 0	direct command of thruster amperage
Thrust mode 1	desired speed

Dynamic Control Level

The vehicle dynamic controller resides at the bottom level of the architecture. This level commands the actuators to achieve a desired vehicle state, which is specified by the layered control level. Odyssey II's dynamic controller has been augmented to accept a range of commands.

Earlier dynamic controllers implemented in this laboratory accepted the relatively simple inputs of heading, depth and speed as the commanded vehicle state. These did not always provide the full range of control desired. Consider a "yo-yo" mission in which the objective is to profile the water column between two depths while maintaining a constant heading. The rate of ascent and descent for the yo-yo may be quite shallow if a long transit is desired, or quite steep if high spatial resolution is required. Clearly a dynamic controller which accepts only commanded depth does not provide an easy method for obtaining the desired decent angle.

The dynamic controller implemented in Odyssey II is subdivided into a vertical plane, horizontal plane, and thrust controller which generate commands to the elevator, rudder, and thruster respectively. Each is capable of accepting a variety of commands from the layered control level, summarized in Table 1.

The dynamic control of the vehicle is achieved with the same algorithms as were used for Odyssey, described in Perrier and Bellingham[1992]. Odyssey II is substantially more maneuverable than the original vehicle, primarily because the large stabilizing duct at the stern of Odyssey has been removed. Also, the two vehicle's thrusters are commanded differently: Odyssey by voltage control, Odyssey II by current control. Consequently the control parameters for the two vehicles and some minor calculations differ.

Layered Control

The elementary unit of layered control is the behavior. A behavior receives sensory input and generates commands. Each behavior is responsible for a specific mission objective. For example, the objective of an obstacle avoidance behavior is to prevent the vehicle from hitting objects. A layered control command structure consists of a number of behaviors with different objectives. The command outputs of the behaviors are resolved into the

final command that is sent to the vehicle. Thus, behaviors embody discrete vehicle capabilities. At present, a total of 18 behaviors have been written for Odyssey II, with a little more than half those behaviors employed in the field. A list of those includes:

- depth\_envelope*: ensures that the vehicle does not exceed a maximum depth or climb above a minimum depth, and prevents the vehicle from approaching too close to the bottom.
- arctic\_depth\_envelope*: the same as *depth\_envelope*, except that instead of preventing the vehicle from approaching the bottom, it keeps the vehicle from colliding with the ice canopy.
- detect\_collision*: monitors the output of the accelerometers to detect a jerk (i.e. the time derivative of total acceleration) which indicates a collision.
- mission\_timer*: ensures that the vehicle shuts down after expiration of a set time.
- acquire\_heading*: causes the vehicle to turn to a desired heading.
- modem\_communicate*: loads messages for the modem to send indicating the progression of the mission and detection of any failures.
- setpoint*: commands the vehicle to attain a given heading, depth, and speed for a set length of time.
- setvector*: commands the vehicle to attain a given heading, pitch, and speed for a set length of time.
- waypoint\_2d*: commands the vehicle to attain a given location in space using long-baseline navigation.
- set\_rudder*: causes the vehicle to set its rudder to a given deflection for a period of time.
- survey\_dead\_reckon*: commands the vehicle through a grid survey using dead-reckoning navigation.
- survey\_with\_nav*: commands the vehicle through a grid survey using long-base-line navigation.
- homing*: commands the vehicle to home on an acoustic beacon using the ultrashort-baseline navigation system.
- homing\_directed*: commands the vehicle to home on an acoustic beacon from a particular direction, and to try again if an approach is missed.
- deep\_homing\_directed*: the same as *homing\_directed*, but also causes the vehicle to approach the beacon on a climbing path, to ensure the vehicle stays deep as long as possible.

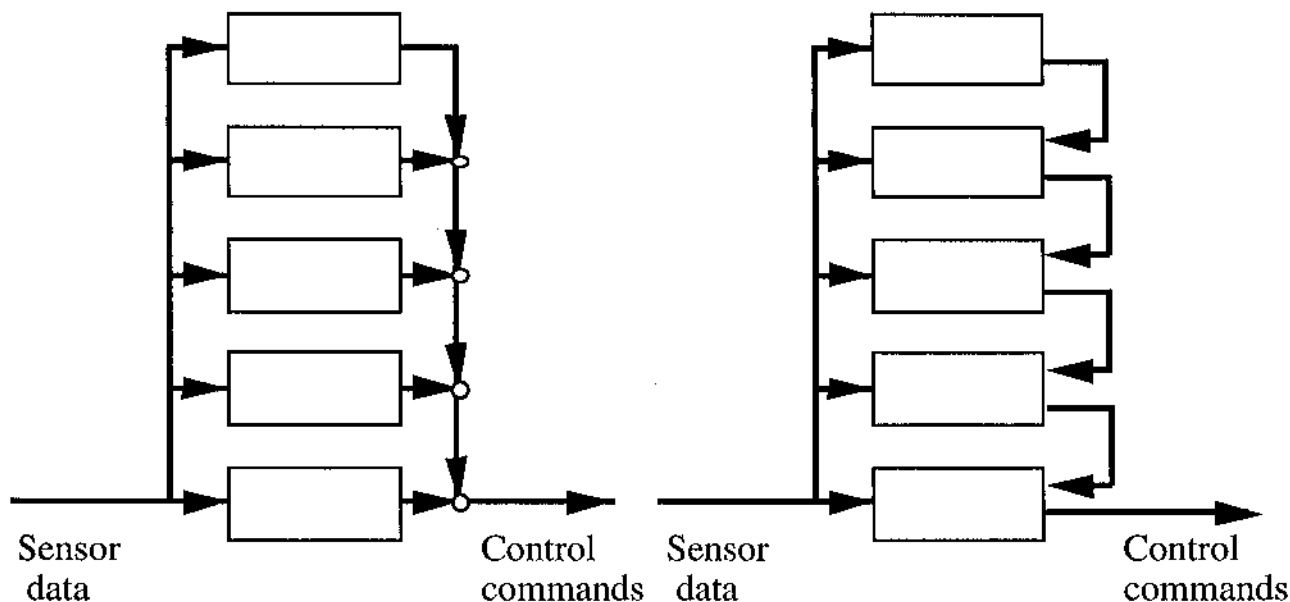


Figure 3: The classical layered control arbitration structure is shown on the left, in comparison with the arbitration structure used for this work, shown on the right.

*race\_track*: commands the vehicle to alternately home on first one then another acoustic beacon for a number of cycles set by the user.

In the present implementation of layered control, behaviors are capable of a much broader range of actions beyond simply changing the commands to the dynamic controller. Behaviors can call other behaviors, and in fact many of the behaviors listed above accomplish their goals by successively calling "building block" behaviors like setpoint, setvector, and waypoint. An example is *survey\_with\_nav* which calls waypoint to achieve the transit to the survey area, then uses successive calls to waypoint to accomplish the survey. Behaviors can also change arguments of other behaviors. An example of this is *deep\_homing\_directed*, which is designed to bring the vehicle to a recovery point under an ice canopy.

Fundamental to layered control is the arbitration of the outputs of the many behaviors comprising an active behavior structure. For the present work, the separation of the command into horizontal plane, vertical plane, and thrust components extends through the layered control structure. Arbitration for each command type proceeds independently from the rest. Thus a behavior generating a depth mode command will over-ride only the depth mode command of the higher behaviors in the structure, and will leave the other two command modes unaffected.

The classical arbitration technique is to simply have the output from the lowest active behavior in the structure (i.e. the highest priority behavior) control the vehicle. This is unattractive under a variety of circumstances, primarily because it precludes the possibility that commands which satisfy more than just one behavior can be generated.

Consider a case in which an obstacle avoidance algorithm is attempting to avoid a target detected directly in front of the vehicle. From the perspective of obstacle avoidance it does not matter if the vehicle turns right or left, yet it must choose one or the directions without reference to the requirements of other behaviors which may one direction much more attractive than the other.

The variety of commands types accepted by the dynamic control level complicates the arbitration process, since different behaviors may generate different types of commands. Thus a behavior generating a commanded depth and a behavior commanding elevator angle directly may need to be resolved.

A partial solution to both problems identified above is to have behaviors hand off their output to the next behavior in the stack. Thus higher priority behaviors can examine the command generated by the next lower priority behavior, and choose whether to over-ride the command. In addition, the task of deciding whether a command of one mode (e.g. elevator position) should be over-ridden by a command of another mode, can be left to the particular behavior concerned.

### VEHICLE DATA STRUCTURE

An extremely important feature of the present implementation of vehicle control software is the vehicle data structure. The structure contains descriptions and values for sensors and behaviors. It also contains the configuration of the active layered control structure (i.e. the priority and argument values for active behaviors) and the output command structure. The state structure serves a number of important functions: it provides a single global

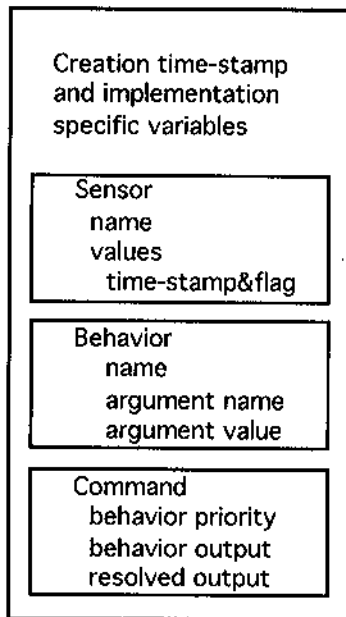


Figure 4: The vehicle data structure and its three primary components.

structure which once accessed provides the entire vehiclestate to a function or process. It also provides the template for mission configuration, data logging, and data analysis.

#### Structure Contents

The data structure has three primary components: the sensor, behavior, and command portions. A type definition used repeatedly in the structure is "sensor," which includes four elements: the value of the variable (a double), a name (a char array), a time-stamp (a long int), and a flag (an int). The name is central to simplifying software upgrades, documenting data logging files and ensuring backwards compatibility of code. The flag variable is presently unused.

The sensor portion of the vehicle state structure is an array of type "sensor." It contains such information as raw readings from vehicle sensors, processed values, coordinates of acoustic beacons, frequencies and other critical parameters of those beacons. At present there are 118 different variables in Odyssey's sensor portion of the vehicle state structure. Examples of sensor names includes:

```
m_present_time(s)
u_debug_var(int)
c_vehicle_stop(err)
c_modem_active(bool)
c_modem_messages_on(arb)
m_roll(rad)
m_pitch(rad)
m_yaw(rad)
m_roll_rate(rad/s)
```

```
m_pitch_rate(rad/s)
m_yaw_rate(rad/s)
```

Note that the sensor names include the units. Thus the vehicle state structure provides both a storage location and some degree of documentation of the nature of the contents.

The behavior portion of the state structure contains the names of all the behaviors, their arguments, which is of type "sensor." Thus each argument has a name, values, time-stamp, and flag. Time-stamps and flags are not used for behavior arguments. For example the behavior "arctic\_depth\_envelope" has the following arguments:

```
max_depth(m)
min_depth(m)
ice_env_active(bool)
min_ice_separation(m)
depth_cutoff_active(bool)
cutoff_depth(m)
```

A behavior is configured by setting the argument values. However, behaviors are not activated from the behavior portion of the structure: this is reserved for the command portion.

The command portion of the state structure contains both the list of active behaviors as a function of their priority, and their outputs. Thus a behavior is activated by placing its index in the active behavior list. When it is called, it places its output in the command list.

#### Maintaining and Propagating the Structure

A structure editor has been created for making changes to the structure, and ensuring that the appropriate header files are generated once a change has been made. The header files contain the structure definition, as well as a list of constant definitions which are used for accessing the structure.

The constants defined by the structure editor provide a means by which the structure could be interrogated without having to know the array offset of a given element of the structure. To create a constant, the editor takes the name of a variable, strips off the units declaration, and makes the resulting string all caps. The editor generated constant declarations for the sensors listed in the previous section are:

```
#define M_PRESENT_TIME 0
#define U_DEBUG_VAR 1
#define C_VEHICLE_STOP 2
#define C_MODEM_ACTIVE 3
#define C_MODEM_MESSAGES_ON 4
#define M_ROLL 5
#define M_PITCH 6
#define M_YAW 7
#define M_ROLL_RATE 8
#define M_PITCH_RATE 9
#define M_YAW_RATE 10
```

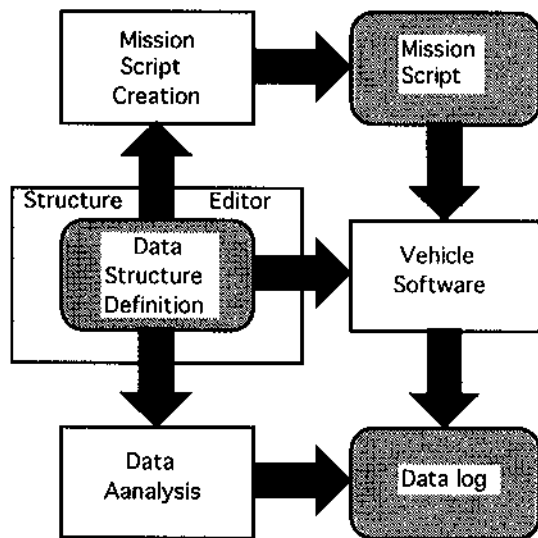


Figure 5: The data structure is central not only to the vehicle software, but to mission script generation and data logging and analysis as well.

Constant declarations for behaviors are generated from behavior names in the same fashion as for sensors.

Access to a given sensor is achieved by reading the sensor array directly, using the constant derived from the sensor name. Updating a sensor is done by a function call, rather than by direct access. The function loads both the new sensor value and the sensor time-stamp.

### Data Logging

Data logging and analysis is vastly simplified by the chosen sensor structure. To log all sensor values of the vehicle, the data logging routine simply checks the sensor time-stamps each control cycle (0.2 seconds). If a time-stamp indicates a change of a given variable, the variable is logged. Thus, only sensors which have changed their values since the last data cycle are recorded, minimizing the use of the storage medium.

In a data file generated during a mission, a different number of sensors may be logged at any particular cycle. The problem is to distinguish sensor values logged at different time intervals. This is solved by always making the first sensor value of a data group have an index number >10000. The float paired with this data index will give the time stamp for the data set in seconds.

A data file generated by Odyssey II has three sections:

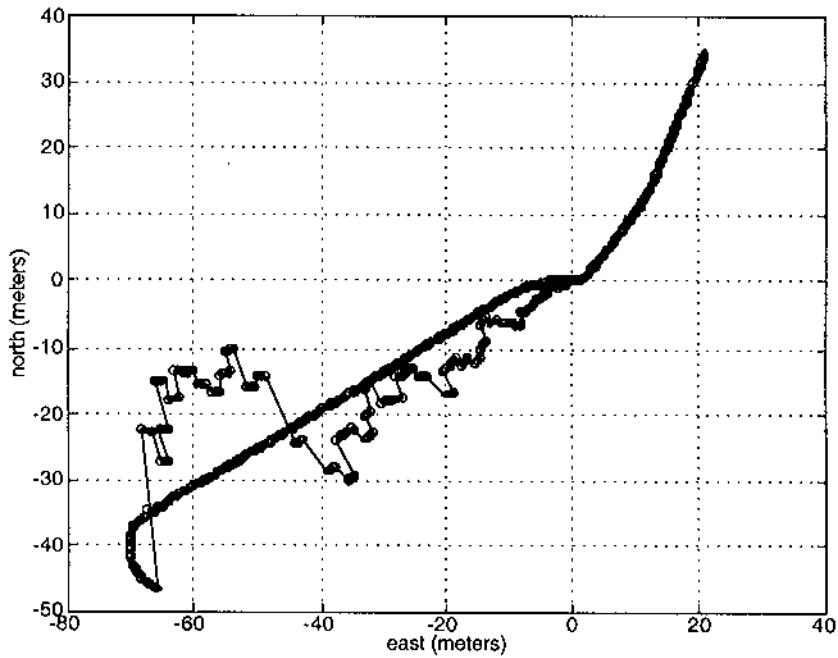
- 1) A series of ASCII lines
  - a) date and time of the mission initialization
  - b) date and time of vehicle data structure creation
- 2) An ASCII section contains pairs of integers and strings followed by '\n'. The pairs are: array index

number of the sensor in question (an integer), and the sensor name). If a line begins with a '#' character, it is a comment line, and everything up to the next '\n' will be ignored.

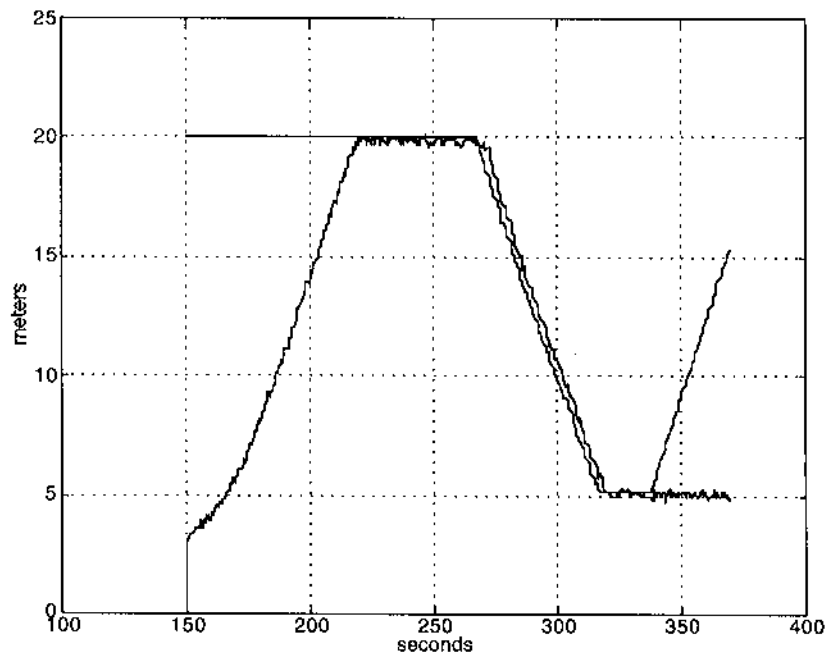
```

state: Arctic setpoint and homing mission March 28, 1994
sensor: u_debug_var(int) 0
sensor: c_modem_active(bool) 1
sensor: c_modem_messages_on(arb) 1
sensor: u_usbl_1north(m) 0
sensor: u_usbl_1east(m) 0
sensor: u_usbl_1depth 4
sensor: uo_control_kpp 1
sensor: uo_control_kdp 1
sensor: uo_control_kip 0
sensor: uo_control_Kdepth 0
sensor: uo_control_pitch_lim(deg) 30
sensor: uo_control_depth_lim(deg) 0.3
sensor: uo_control_kph 1
sensor: uo_control_kdh 1
sensor: uo_control_kih 0
behavior: mission_timer 1
  b_arg: time(s) 220
behavior: beacon_range_cutout 2
  b_arg: beacon_num(#) 1
  b_arg: range_for_good_hit(m) 5
  b_arg: good_hits_required(#) 5
  b_arg: running_hit_total(#) 9
behavior: detect_collision 3
  b_arg: accel_dt_cutoff(m/s^3) 2
  b_arg: collision_flag(bool) 0
behavior: arctic_depth_envelope 4
  b_arg: max_depth(m) 40
  b_arg: min_depth(m) 20
  b_arg: ice_env_active(bool) 0
  b_arg: min_ice_separation(m) 5
  b_arg: depth_cutoff_active(bool) 1
  b_arg: cutoff_depth(m) 45
behavior: deep_homing_directed 5
  b_arg: behavior_flag(#) 2
  b_arg: depth(m) 5.2
  b_arg: speed(m/s) 1.35
  b_arg: approach_heading(rad) 4.15
  b_arg: marker_distance(m) 20
  b_arg: usbl_beacon(#) 1
  b_arg: capture_distance(m) 3
  b_arg: capture_hits(#) 3
  b_arg: tries(#) 10
behavior: setpoint 6
  b_arg: heading(rad) 4.15
  b_arg: depth(m) 5
  b_arg: speed(m/s) 1.35
  b_arg: time(s) 80
  
```

Figure 6: Mission script file for one of the many mission run in the Arctic.



a)



b)

Figure 7: a) Vehicle track for mission script shown in Figure 6. The discontinuous jumps are caused by ultrashort-baseline navigation system updates to the otherwise smooth dead-reckoned track. The vehicle starts at position (0,0) and proceeds to the southwest. On turning, the vehicle acquires the homing beacon, and the successive updates cause the jumps visible in the vehicle track. The portion of the vehicle track extending to the northeast is an artifact of dead-reckoning being continued despite capture of the vehicle in the recovery net. b) Vehicle depth and commanded depth for the mission displayed in a).

- 3) The binary section contains pairs of numbers, one integer and one float (2 bytes and 4 bytes respectively). The first is the index number, and the second is the value of the sensor corresponding to that index number. The data is grouped in sets which have all been logged at the same time. To indicate the beginning of a new set, the first index number is >10000. The float associated with this index number is the time (in seconds since the beginning of the mission) of that data set acquisition.

Data files are self-documenting: the names of the sensors are recorded in the file as are the sensor values. Data analysis programs do not have to have a copy of the data structure to make the results of a data file available to be plotted or manipulated. As a consequence, neither the data logging nor the data analysis are sensitive to changes in the vehicle structure. No code modifications are required to handle new sensors either in data logging or data analysis.

#### Task Configuration

Programming a task within layered control typically involves three steps: deciding which behaviors are required, assigning them appropriate priorities, and assigning values to the arguments of the behaviors. One attractive feature of our implementation is that these steps can be performed by editing a mission initialization file. Because the vehicle source code does not need to be recompiled, missions can rapidly be configured during field experiments.

Figure 4 shows the mission initialization file for a typical Odyssey II mission performed in the Arctic in March, 1994. The first part of the file specifies initial values for different variables in the vehicle data structure, such as control gains and acoustic beacon locations. The file then gives the priorities of the six behaviors that are active for the mission and supplies argument values to them. The highest priority behavior is mission\_timer, which will terminate the mission after an elapsed time interval. The next line in the mission file sets this time-out to 220 seconds. The next two behaviors, beacon\_range\_cutout and detect\_collision, are designed to shut the vehicle down once it is back in its recovery net. Beacon\_range\_cutout looks for 5 returns with a range less than 5 meters. Detect\_collision looks for a change in acceleration of  $2 \text{ m/sec}^3$ . The arctic\_depth\_envelope behavior sets the operating envelope to be between 20 and 40 meters to assure vehicle safety. The behavior deep\_homing\_directed controls the terminal homing phase of the mission. Argument values specified here include the pinger depth and net orientation (approach heading). And finally, the mission begins with the execution of a setpoint behavior at a heading of 4.15 radians for a duration of 80 seconds.

Figure 5 shows the vehicle trajectory for this mission. The 5 meters depth commanded by the setpoint behavior is overridden by the depth envelope behavior to cause the vehicle to dive to 20 meters depth for the first half of the

mission. When the behavior deep\_homing\_directed becomes active, it modifies the min\_depth argument of the depth envelope behavior to allow the vehicle to climb back up to the depth of the homing beacon in the net. This mission ended in a "bullseye" recovery in the net

## CONCLUSIONS

The implementation of layered control described provided a robust environment for vehicle code development and execution. Adding a behavior to the repertoire has been drastically simplified, and in fact one of the more complex behaviors, deep\_homing\_directed, was written in the Arctic, tested in simulation, and tested on the vehicle in the water in a single day.

The vehicle code as configured can in many respects be considered a kernel with a well defined interface (the data structure). We view this as providing a powerful tool for supporting research on mission level software for autonomous vehicles. State-configured layered control will be implemented, however other approaches will be explored as well, including development of "intelligent" tools to assist human users with task level programming. Such tools should aid a mission programmer with constructing mission files for autonomous missions, but should also have applicability to supervisory control of semi-autonomous vehicles.

## ACKNOWLEDGMENTS

This work was supported by the Office of Naval Research under contract N00014-92-J-1287, the National Science Foundation under contract number BCS-9311151, and the Massachusetts Institute of Technology Sea Grant College Program under contract NA90AA-D-SG424.

## REFERENCES

- Brooks, R., "A Layered Intelligent Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, RA-2, 14-23 1986.
- Bellingham, J. G., Consi, T. R., Beaton, R., and Hall, W., "Keeping Layered Control Simple," *Proceedings AUV '90*, pp. 3-8, 1990.
- Bellingham, J. G. and Consi, T.R., "State Configured Layered Control," *Proceedings of Mobile Robots for SubSea Environments, Int. Advanced Robotics Programme*, Monterey CA, pg 75, 1991.
- Bellingham, J.G., Consi, T.R., Tedrow, U., and Di Massa, D., "Hyperbolic Acoustic Navigation for Underwater Vehicles: Implementation and Demonstration," *Proceedings AUV '92*, 1992.
- Bellingham, J. G., Deffenbaugh, M., Leonard, J. J., Schmidt, H., Catipovic, J., "Arctic Under-Ice Survey operations," *Proceedings 8th Int. Symp. Unmanned Untethered Submersible Technology*, Autonomous Undersea Systems Inst., Portsmouth, NH, Doc. 93-9-01, pp. 50-59, 1993.

- Bellingham J. G., Goudey, C. A., Consi, T. R., Bales, J. W., Atwood, D. K., Leonard, J. J., and Chrysostomidis C., "A Second Generation Survey AUV," *To appear in the Proceedings of AUV '94*, 1994.
- Connell, J. H., Minimalist Mobile Robotics: A Colony-Style Architecture for an Artificial Creature, Academic Press, Cambridge MA, 1990.
- Connell, J. H., "SSS: A Hybrid Architecture Applied to Robot Navigation," *Proc. of the 1992 Proceedings IEEE International Conference on Robotics and Automation*, Nice, France, pp. 2719-2724, 1992.
- Curtin, T. B., Bellingham, J. G., Catipovic, J., Webb, D., "Autonomous Oceanographic Sampling Networks," *Oceanography*, Vol. 6, No. 3, pp. 86-94, 1993.
- Deffenbaugh, M., "A Matched Field Processing Approach to Long Range Acoustic Navigation," Master of Science in Oceanographic Engineering Thesis, Massachusetts Institute of Technology and Woods Hole Oceanographic Institution, February 1994.
- Loch, J. L., "Design of a Reflexive Hierarchical Control System for an Autonomous Underwater Vehicle," Master of Science in Aeronautics and Astronautics, Massachusetts Institute of Technology, June, 1989.
- Perrier, M. and Bellingham, J. G., "Control Software for an Autonomous Survey Vehicle," *Proceedings of Workshop on Underwater Robotics, International Advanced Robotics Programme*, Genova, Italy, 1992.