Proceedings of the 1996 IEEE
International Conference on Robotics and Automation
Minneapolis, Minnesota - April 1996

Autonomous Underwater Vehicle Control Coordination Using A Tri-Level Hybrid Software Architecture

A. J. Healey[1], D. B. Marco, R. B. McGhee

Autonomous Underwater Vehicle Laboratory
Naval Postgraduate School
Monterey
California, 93943

[1] Point of Contact
(408)-656-3462 (Phone)
(408)-656-2238 (Fax)
healey@me.nps.navy.mil

ABSTRACT

This paper proposes the use of Prolog as a rule based specification language for the coordination of multiple control functions as required to perform missions with autonomous underwater vehicles. Missions being considered include Ocean Survey, Search and Find, Bottom Mapping, Mine Countermeasures, among others. Control of both motion of the vehicle and the logical sequencing of the mission phases including specified forms of error recovery from vehicle as well as mission failures must be accomplished. We first define terms used in this type of control system and show that such systems fall into the class of 'Hybrid' controllers coupling discrete state / time independent and continuous state / continuous time elements. The design of these systems has received little attention, but, the software architecture to implement them is often composed of three levels for ease of segregation and development of functionality.

An implementation of our controller on the NPS Phoenix vehicle uses Prolog as a rule based language to specify and execute the phases of any mission. Embedded in the rule body are functions that interface with the vehicle to gather sensory data and generate signals as required to trigger transitions between control functions, and to initiate commands for the initiation of subsequent control functions. The importance of the use of Prolog is that the same code is used for mission specification as is used for its execution thereby eliminating the question of correctness.

Control of a mission segment using command generation to simultaneously drive the vehicle to a point on space and time (to coincidentally reach a given depth and heading) is described with experimental results. The development of transitioning signals can be problematic and is discussed alongside error recovery techniques using 'guaranteed phase completion'.

INTRODUCTION

The human experience is limited underwater. Even shallow water coastal areas are not well understood. As an aid to the development of coastal environmental understanding, remotely operated vehicles (ROVs) are often used, piloted by a human operator on a surface vessel providing power and signal linkages through an umbilical cable. New technology aimed at eliminating the tether requires acoustic communications to and from the vehicle and a degree of autonomy on the vehicle sufficient to maintain vehicle task control. Building an ever increasing level of automatic capability into an underwater vehicle is of interest to us. In particular, we are concerned with the ease of reconfiguration of control software code as missions become more complex or vehicle capabilities change. To that end, tri-level software architectures are useful for enabling control over the resulting hybrid system which comprises discrete state, as well as continuous time - continuous state elements. The three levels (Strategic, Tactical, and Execution levels in our terminology), separate the control requirements into easily modularized functions encompassing logically intense discrete state transitioning using *asynchronously* generated signals for control of the mission and real time *synchronized* controllers that stabilize the vehicle motion to callable commands.

In our controller architecture, the Strategic level uses Prolog as a rule based mission control specification language. It's inference engine cycles through the predicate rules to manage the discrete event logical aspects of mission related decisions. It transitions states, and generally develops the commands that drive the vehicle through its mission. Error recovery procedures from failures in the mission tasks or the vehicle subsystems are included as transitions to 'error' states that ultimately provide commands to the servo level control for appropriate recovery action.

0-7803-2988-4/96 $4.00 © 1996 IEEE

The Tactical level - at the moment - is a set of "C" language functions that interface with the Prolog predicates and return TRUE / FALSE when called, and which are interfaced to the real time Execution level controller using asynchronous message passing through ethernet sockets.

The Execution level commands the vehicle subsystems to activate control functions that correspond to those commanded. Communication from the Tactical level to the Execution level takes place through a single socket. By the design of this hierarchical control system, the Tactical level runs asynchronously and retains the mission data file and the mission log file in global memory. It sends command scripts to the Execution Level and requests data for the evaluation of state transitions. The architecture is a hybrid between the true hierarchical control of NASREM [1] and the purely reactive schemes 'subsumption' [2, 3]. In this way, control of mission can be retained, while reacting to unanticipated events is also enabled.

In new work with the NPS Phoenix - an autonomous underwater vehicle, we have extended the flight control experiments that were conducted and reported previously [4]. We have now developed the thruster control behavior of the vehicle. Experiments using command generation to drive orthogonal actions have been conducted and illustrate herein both the power of sliding modes for control of transient response and command tracking, and the power of Prolog for the mission coordination.

In order to assist in the subsequent discussion, it helps to define some terms that have long plagued the underwater robotics community. We offer the following:

Let the set of all actuators available to the vehicle be denoted by, A, the set of sensors by S, and the set of continuous time states of the vehicle be X, then:

Definition: A Control Function (CF), is the use of a particular subset, $a_i \subset A$ of vehicle actuators with a particular subset, $s_i \subset S$, of the vehicle sensor suite, to both estimate a corresponding subset of the continuous time continuous states, $x_i \subset X$, of the vehicle and drive a particular set of continuous time error states, e_i to zero. The control error is defined as the difference between a command value and the estimated actual motion. A control function is analogous to the Robot Task in [5] and employs an appropriate control law, 'L', linking the actuator commands, $u_i(t)$ to sensory output, $y(t)$ and commands, $r_i(t)$

$$L(CF_i):\ u_i(t) = f_i(y_i(t), r_i(t)).$$

Definition: A Vehicle Primitive is a linguistic string associated with a particular Control Function.

Definition: Orthogonal Control Functions are those that utilize independent subsets of actuators.

Definition: Interacting Control Functions are those that utilize part of the same subset of actuators utilized by other CF's.

Definition: A Sensor Based Reaction, as in a low energy level in the vehicle, vehicle shoaling, or any obstacle detection, is a condition when a sensor value sustains a critical value. If c is the critical value, ε is a small positive bound, and $F(\cdot)$ is a low pass filter, a Sensor Based Reaction occurs if

$$|F(y) - c| < \varepsilon.$$

All Control Functions terminate at a 'Termination'.

Definition: The Termination of a control function occurs either

1) at a specified time, or

2) when a positive definite function $P(\cdot)$ of the control error lies within a prespecified bound. If b, is a positive bound and $F(\cdot)$ is a low pass filter the termination condition is

$$F(P(e)) < b, \text{ or}$$

3) upon a Sensor Based Reaction.

Definition: A Behavior (B), is described as the execution of a particular sequence of CFs each driven to a termination.

Comment: The state of performing a given CF is a "discrete state" of the system. The condition of reaching a termination is linked to the transition of the discrete state of the system from one state to another as defined by the system Behavior (B). A Behavior can be represented for example, by a Petri Net, a finite state diagram, or, linguistically, by a set of Prolog rules.

Definition: A Hybrid Control System in the context of this work is a control system of hardware and software elements that is capable of driving a vehicle through a set of Behaviors.

The definition of a mission plan is now reduced to the specification of a sequence of Behaviors (B) to be conducted during each mission phase with provisions for backtracking and alternative goal satisfaction upon failure

of any desired phase. These include the ordered sequence of control functions and their termination conditions.

The principle of "guaranteed phase completion" is such that all control functions have a termination so that each mission phase, its behaviors and control functions, will terminate - essentially, the mission plan will specify that all mission phases complete - either successfully or by abortion.

VEHICLE CONTROL SYSTEM

The control concepts presented are being evaluated experimentally using the NPS Phoenix vehicle shown in Figure 1. It has been recently outfitted with the tri-level controller, currently implemented in hardware using three networked processors, illustrated in Figure 2. All Execution level software is written in 'C' and runs on a Gespac M68030 processor in a separate card cage inside the boat. Connected in the same card cage is an ethernet card and an array of real time interfacing devices for communications to sensors and actuators indicated in the details of Figure 3. The Execution level control code containing a set of functions in a compiled module called 'exec' is downloaded first and run to activate any mission. It starts the communication s socket on the Gespac side and waits for the higher level controller to start.

Strategic Level

The Strategic level Prolog rules which specify the mission to be conducted are compiled and linked together with the supporting Tactical level 'C' language functions into the single executable process called 'Mission_Control', that is run in a Sun SPARC 4 laptop computer and linked through ethernet and a non-blocking socket to the Gespac processor. Upon starting, it first opens the Sun side of the communications socket, initiating the ethernet link between both Sun and Gespac processors, then sending sequenced control commands to the vehicle. All vehicle control functions, with the exception of the transmission of sonar imaging data, communicate by message passing through that socket. Typical rules are given as an example later.

Tactical Level Software

A second Sun process called the 'Sonar Manager' is opened which runs asynchronously in the Sun and with equal priority to the 'Mission_Control'. This process is linked through a separate socket to the Gespac for the purpose of the reception and handling of sonar imaging data. This process is activated if and when sonar is activated by a Strategic level predicate call. The 'Sonar Manager' captures data that is sent out from the Execution level as soon as it has been acquired, and then processes and passes the data to be displayed on an IRIS Graphics workstation for user visualization purposes. Tactical level software is designed to link with the Prolog rule base, send vehicle primitives to the execution level software and process the numerical computations associated with computing the termination conditions. It necessarily requires the computation of filtered data, and at the present stage of development performs computation asynchronously. Time is not critical as the communication and commands to receive data and activate or terminate control functions are designed to change only as needed, and to not influence the stability of the vehicle motion.

Execution Level Software

The structure of the Execution level software is illustrated by Figure 3 which indicates that it is composed of software at the hardware interface (software drivers) as well as software for vehicle control. After initialization of power systems and sonars, and the basic driver settings, the PIA card pins that control the on/off feature of power supplies, thruster power, screw power, and sonar power, a simple timing loop is entered and reentered at a fixed update rate (in our case 0.1 sec.) during which the following takes place,

1. read the socket 'A' for behavior based mode command flags and control set points,

2. read the sensors,

3. selecting appropriate 'C' code control functions for computing and sending control values to actuators, using multiple 'case of ' checks for distinguishing the commands,

4. writing selected data to memory or sockets 'A' or 'B' as appropriate, and

5. checking time for any time based events and waiting for the next timing interrupt to maintain integrity of the digital control loop,

Specific control laws as built into callable modules of code are easily selected according to the vehicle primitives sent.

Vehicle Primitive Development

In previous work [6], waypoint following in a transit phase of a mission was demonstrated in a swimming pool test area where Control Functions included

a) Forward_Speed_Control
   Control vehicle forward speed using stern screws.
b) Fin_Steering
   Control vehicle heading using bow and stern rudders.
c) Fin_Depth_Control

Control depth of the vehicle using bow and stern planes.

d) Waypoint_Following
Follow three dimensional (X,Y,Z) waypoints using fins and stern screws.

e) Bottom_Following
Control the height of the vehicle above the bottom.

These Control Functions were developed with a)-c) running simultaneously, but subsumed by the guidance laws implemented in d); and, with c) subsumed by e). The control laws corresponding to these functions have been implemented based on PD, and Sliding Mode methods as explained in [7].

Control laws for these functions are readily accomplished entirely in the Execution level using digital control algorithms running at 0.1 sec. update rate. Now, however, new, more complex functions are being enabled using active control of thrusters and sonar. These include,

g) Submerge_and_Pitch_Control
Control vehicle depth and pitch angle using vertical thrusters.

h) Heading_Control
Control vehicle heading using lateral thrusters.

i) Longitudinal_Positional_Control
Control longitudinal position of vehicle from a target.

j) Lateral_Positional_Control
Control lateral position of vehicle from a target using lateral thrusters.

k) Center_Sonar(Sonar)
Rotate sonar head 'Sonar' to ahead position.

l) Ping_Sonar(Sonar,Command)
Ping sonar 'Sonar' using 'Command'.

m) Update_Head_Position(Sonar,Command)
Update head position of sonar 'Sonar' based on 'Command'.

n) Read_Sonar(Sonar)
Return sonar range from 'Sonar'.

o) Initiate_Filter_For_Sonar_Range
For smoothed range and range rate (ST1000 sonar only).

p) Reinitialize_Filter
Reset filter for next data gathering / control sequence.

Note: Control functions g) through j) can be implemented using step input commands for their activation or, for more precise control over transient behavior, command generators would be used which specify the desired position, rate, and acceleration of the output as a function of time.

Most of these functions need a given subset of the actuator system to be active under the operation of either an open loop command or a feedback control law. Some of the functions use orthogonal sets of actuators and are thus additive. Some use the same actuators to control different functions and thus control laws may be additive. This means, for example, that vertical thrusters may be used via control laws to control depth as well as pitch, and lateral thrusters to control heading as well as lateral position and side slip speed. In combination with propulsion motors, most functions including Submerge_and_Pitch_Control, and Longitudinal_-Position_Control, as well as Heading_Control, may now be commanded. Heading_Control and Submerge_and_Pitch_Control and virtually any multiple combination of a) to o) above that would not cause a conflict of actuator control or sensor usage, are performed.

Activation of orthogonal behaviors are instituted using message passing that is a way of communicating between Tactical Level 'C' functions and the real time control loop of the Execution Level control. At each pass through the control loop, a read is made from the communications socket and a ladder check for particular 'case of' flags determines which set of sensors and actuators and control laws are to be activated during the computation cycle. The same technique is used to flag the activation of sonars, and filtering actions, and similarly for flags to indicate which data stream is to be written in return.

Reactivity

Reactive behavior in our controller can be handled inside the Execution level control loop through command overrides following a sensor read, as, for instance, a new obstacle detection requiring an emergency surface or obstacle avoidance (flinch) response. At the Tactical level, reactive error recovery can be handled by resetting key parameters associated with control performance evaluations. An example is the resetting of a control gain if a particular function cannot be stabilized. Reactive behavior is also handled at the Strategic level by transitioning to states that command an error recovery procedure such as to surface if, for example, a particular action is not observed to be taken after a pre-specified time out.

While the work of [3] has developed GAPPS rules that are more like our Strategic level rules, but, in the end would also provide mode commands to vehicle servos, our work is developed around a rule based control to sequence mission related tasks [8, 9] according to a mission plan that could (if one prefers to view it this way) represent a hierarchy of state machines with transitioning from one to another as mission phases are completed. The middle level of our tri-level architecture is then used to generate the scripts required to produce in the vehicle the requisite behavioral action. The Tactical Level functions deal with

the interfacing between asynchronous control function commands and the real time computational control requirements of the 'sense - compute - send' cycle within the Execution level vehicle motion control loop.

The behaviors a) through o) are now stably implemented in the NPS Phoenix vehicle through attention to appropriate digital control loops in the Execution level. In principle, once developed to a satisfactory point, the Execution level controller of any vehicle would not require any change as mission requirements change.

COORDINATED SUBMERGENCE / ROTATIONAL CONTROL USING COMMAND GENERATION

As part of a joint mission to evaluate control software architectures between US and French research laboratories, it has been decided to evaluate the performance of the NPS Phoenix in a behavior that will submerge the vehicle to a specified depth at a specified rate according to a command function. A similar function will describe the required heading and heading rate so that the attainment of the new final position and heading will occur at a defined final time. The performance of this type of maneuver with land robotic is relatively easy but such performance underwater has not been demonstrated.

In this section we will describe the command generators used, and show the control performance obtained with sliding mode control functions executing simultaneously.

Vehicle Model for Submergence

The vehicle dynamics in submergence using both the bow and stern vertical thrusters can be described by the following differential equation for the continuous time, continuous state evolution:

$$M\ddot{z}(t) + b_z\dot{z}(t)|\dot{z}(t)| = 2Z_{prop}(t) + F_B(t) + \delta f_z(t) \tag{1}$$

where

$$M = m + m_a$$

$$Z_{prop}(t) = \alpha_{vt}v(t)|v(t)|$$

and m_a is the vertical added mass, α_{vt} is a coefficient relating the square of the vertical thruster motor voltage, $v(t)$, to the force developed, $F_B(t)$ is the unmatched buoyancy which varies within some bound but which on any day can be either positive or negative, and is unknown, b_z is the coefficient of square law drag in the z direction, and $\delta f_z(t)$ describes an upper bound on vehicle/model mismatch.

The command generator for the submerge motion is taken from

$$[z_{com}, \dot{z}_{com}, \ddot{z}_{com}] = G(z_0, z_f, a_{max}, T_0, T_f)$$

where a fifth order zero jerk profile has been chosen so that the maximum acceleration and the bandwidth capacity of the vehicle is not overly exceeded and z_0, T_0 is the initial depth and starting time while z_f, T_f is the final desired depth and time at the end of the maneuver. Normalized profiles for position, velocity, and acceleration commands are shown in Figure 4.

The sliding mode control law for submerging is given by

$$v(t) = \sqrt{|x(t)|}sgn(x)$$

where

$$x(t) = \left(\frac{M}{2\alpha_{vt}}(\ddot{z}_{com} + \lambda\dot{\tilde{z}}(t) + \eta tanh(\sigma(t)/\phi)) + \frac{1}{2\alpha_{vt}}(b_z\dot{z}(t)|\dot{z}(t)| - F_B(t) - \delta f_z(t))\right) \tag{2}$$

and the sliding surface is

$$\sigma(t) = \dot{\tilde{z}}(t) + \lambda\tilde{z}(t), \tag{3}$$

and the tracking errors are defined as

$$\tilde{z}(t) = z(t) - z_{com}(t)$$
$$\dot{\tilde{z}}(t) = \dot{z}(t) - \dot{z}_{com}(t). \tag{4}$$

Using integral control the command for voltage is also

$$v(t) = \sqrt{|x(t)|}sgn(x)$$

but

$$x(t) = \left(\frac{M}{2\alpha_{vt}}(\ddot{z}_{com} + \lambda_1\dot{\tilde{z}}(t) + \lambda_2\tilde{z}(t) + \eta tanh(\sigma(t)/\phi)) + \frac{1}{2\alpha_{vt}}(b_z\dot{z}(t)|\dot{z}(t)| - F_B(t) - \delta f_z(t))\right) \tag{5}$$

and

$$\sigma(t) = \dot{\tilde{z}}(t) + \lambda_1 \tilde{z}(t) + \lambda_2 \int \tilde{z}(t)dt. \quad (6)$$

Vehicle Model for Heading

The vehicle dynamics for rotation about the body-fixed z-axis (yaw) using both the bow and stern lateral thrusters can be described by the following differential equation for the continuous time, continuous state evolution:

$$I_z \ddot{\psi}(t) + b_\psi \dot{\psi}(t)|\dot{\psi}(t)| = 2N_{prop}(t) + \delta f_\psi(t) \quad (7)$$

where

$$I_z = I_{zz} + I_{zza}$$

$$N_{prop}(t) = \alpha_{lt} v(t)|v(t)|$$

and I_{zza} is the added inertia about the z-axis, α_{lt} is a coefficient relating the square of the lateral thruster motor voltage, $v(t)$, to the force developed, b_ψ is the coefficient of rotational square law drag, and $\delta f_\psi(t)$ describes an upper bound on vehicle/model mismatch.

The command generator for rotation is taken from

$$[\psi_{com}, \dot{\psi}_{com}, \ddot{\psi}_{com}] = G(\psi_0, \psi_f, a_{max}, T_0, T_f),$$

where ψ_0, T_0 is the initial heading and starting time while ψ_f, T_f is the final desired heading and time at the end of the maneuver and is also fifth order with zero jerk.

The sliding mode control law for rotational control is given by

$$v(t) = \sqrt{|x(t)|}sgn(x)$$

where

$$x(t) = \left(\frac{I_z}{2\alpha_{lt}} (\ddot{\psi}_{com} + \lambda \dot{\tilde{\psi}}(t) + \eta tanh(\sigma(t)/\phi)) + \frac{1}{2\alpha_{lt}} (b_\psi \dot{\psi}(t)|\dot{\psi}(t)| - F_B(t) - \delta f_\psi(t)) \right), \quad (8)$$

and the sliding surface is

$$\sigma(t) = \dot{\tilde{\psi}}(t) + \lambda \tilde{\psi}(t), \quad (9)$$

and the tracking errors are defined as

$$\tilde{\psi}(t) = \psi(t) - \psi_{com}(t)$$
$$\dot{\tilde{\psi}}(t) = \dot{\psi}(t) - \dot{\psi}_{com}(t). \quad (10)$$

Transition Criteria

Most control phase transitions of the NPS Phoenix are event based, meaning that a certain set of criteria must be met in order for a transition to occur. A common example of this is when a position set point is sent to the vehicle controllers and reached. A method of determining whether the vehicle has indeed reached this point must be programmed into the control logic. Measuring the position error alone and declaring the maneuver complete when this error is small is not sufficient. This is because the vehicle could be overshooting the commanded position and simply passing through the set point. Therefore, not only must the position error be small but the rate error must also be small. This dual criteria can be expressed mathematically as a positive definite, linear combination of the position error e and the position rate error $\dot{e}$. We use,

$$\sigma_k = w_e|e_k| + w_{\dot{e}}|\dot{e}_k| \quad (11)$$

where w_e and $w_{\dot{e}}$ are positive weights for the position and rate errors respectively. This equation allows a minimum value of σ, denoted σ_0, to be specified defining a threshold for the combination of errors which can be set relatively large when precision control is not required or low for extremely precise positioning. Once σ drops below σ_0, the maneuver is declared complete and a transition to the next control phase may occur.

When noisy sensors are used, the noise prevents σ from settling enough to determine an accurate measurement for the transition, and the use of Equation 11 alone has been found to be unsatisfactory. The signal can be smoothed by filtering σ through a first order digital filter of the form

$$\sigma_{f(k+1)} = e^{-T/\tau}\sigma_{f(k)} + (1 - e^{-T/\tau})\sigma_k \quad (12)$$

where σ_f is the filtered form of σ, τ is the time constant of the filter, and T is the sampling time. The signal for transition, s, is 1 (TRUE) for $\sigma_f < \sigma_{f0}$ or 0 (FALSE) for $\sigma_f > \sigma_{f0}$. Other dynamic error and time based signals are computed similarly.

COMMAND TRACKING PERFORMANCE

The following experiment was performed in the NPS hover tank which measures 6.0 by 6.0 meters square and 1.8 meters deep. During execution all pertinent data was

collected, including depth, depth rate, heading, heading rate, thruster motor speed, etc. The experiment required the vehicle to simultaneously submerge and rotate to a predetermined depth of 1 meter and a heading of 180 degrees. It was specified that the final depth and heading both be reached at 60 seconds from the beginning of the maneuver. This was accomplished using command generators for both control modes with integral control for depth using an anti-reset windup saturation of 0.45 m-sec.

A section of the Prolog code used to control the mission appears below. This code executes until the predefined depth and heading has been attained which is determined by the error criteria presented in the previous section.

```
execute_phase(2) :-
        exec_submerge(X), X==1,
        exec_rotate(X), X==1,
        exec_start_timer(X),
        repeat, phase_completed(2).

phase_completed(2) :- ask_depth_reached(X), X==1,
        ask_heading_reached(X), X==1,
        asserta(complete(2)).

phase_completed(2) :- ask_time_out(X), X==1,
        exec_surface(X), repeat,
        ask_surface_reached(X), X==1,
        asserta(abort(2)).

phase_completed(2) :- ask_sys_problem(X), X==1,
        exec_surface(X), repeat,
        ask_surf_reached(X), X==1,
        asserta(abort(2)).

next_phase(2) :- complete(2), retract(current_phase(2)),
        asserta(current_phase(3)).
next_phase(2) :- abort(2), retract(current_phase(2)),
        asserta(current_phase(mission_abort)).
```

The following tables give the values used in the vehicle control law where the vehicle mass, drag and thruster gains are from [10] and the controller gains were obtained from computer simulation results.

Table 1. Parameters for Submergence Control

Parameter	Value	Unit
m	194.88	Kg
m_a	194.88	Kg
b_z	1378.18	Kg/m
α_{vt}	0.018	N/V^2
λ_1	0.400	rad/sec
λ_2	0.040	rad/sec^2
η	0.030	m/sec^2
ϕ	0.061	m/sec

Table 2. Parameters for Heading Control

Parameter	Value	Unit
I_{zz}	53.60	$Kg\text{-}m^2$
I_{zza}	53.60	$Kg\text{-}m^2$
b_ψ	74.86	$Kg\text{-}m^2$
α_{lt}	0.008	$N\text{-}m/V^2$
λ	0.200	rad/sec
η	0.200	rad/sec^2
ϕ	0.200	rad/sec

Figures 5 and 6 show the normalized time responses for depth/depth rate and heading/heading rate respectively. The vehicle was trimmed to be neutrally buoyant on the surface and the depth and heading was set to zero at this point. The depth response tracks very well until the steady state region where an overshoot occurs. This is due to the vehicle becoming 'heavy' at depth from hull compression, although the error is quickly corrected by the integral action of the controller. Since no disturbance was present in rotation, the heading response shows an extremely precise tracking performance with virtually no error.

The depth rate does track the command but is very noisy due discretization noise from the A/D converter associated with the depth cell and the subsequent rate estimation from this signal. Although the signal is far from clean, the tracking performance is not adversely affected. The heading rate measured from the onboard rate gyroscope shows a definite tracking error, and is due to a non-zero bias in the unit.

Figure 7 shows that the depth and heading are simultaneously controlled except for the small depth overshoot at the end of the maneuver.

SOFTWARE ARCHITECTURAL EVALUATION

It is not an easy task to evaluate a given control system architecture. The theoretical design for stability and robustness leads to selection of parameters that are used in the control functions of the Execution level. We are going beyond that now and are interested in the organization of control software. Some software controllers will be successful for fixed purpose tasks, but here, we have a multipurpose flexible control requirement and, because we are talking about control software, we are led to ask the following questions,

1) Does the controller permit easy evaluation of response and change to control parameters to 'tune' the low level servos?

2) Can this be done while testing is ongoing in real time?

3) Can new sensors be added to the vehicle with little change to the control software?

4) What levels of code and how many functions have to be changed for this new sensor to be added?

5) How many rules (code statements) must be changed if the mission is altered to eliminate, or to add, a new phase?

6) How is the control code modified to test just the performance of a particular existing sensor or actuator set?

7) How easy is it to change the data record for a different set of sensors?

8) How easy is it to change the conditions that define the transition signals?

There are perhaps many more questions that should be considered, dependent on the particular control system software used. The evaluation of our controller is ongoing.

CONCLUSION

The conclusion of our work to date has indicated that complex behavior can be readily coordinated through Strategic level rules, that are easily modified. These act as state transitioning mechanisms and the communication through Tactical level software to the Execution level controllers is a simple but convenient way of commanding competent functions of the vehicle. The design of well behaved control laws and functions at the Execution level is essential as a primary part of the design an is effected through careful attention to the digital control loop design. Human interfacing within the controller can take place at any level.

ACKNOWLEDGMENT

The authors wish to recognize the financial support of the National Science Foundation under Grant No.BCS-9306252.

REFERENCES

[1] Albus, J., "System Description and Design Architecture for Multiple Autonomous Undersea Vehicles" National Institute of Standards and Technology, Technical Note 1251, September 1988

[2] Brooks, R. A., "A Robust Layered Control System for a Mobile Robot" *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14-23, 1986.

[3] Bonasso, R.P., Barrat, J. "A Reactive Robot System for Find and Visit Tasks in a Dynamic Ocean Environment", Proceedings of the 8th UUST, University of New Hampshire, Durham, NH. September 27-29, 1993 pp. 69-80

[4] Healey, A.J., Marco, D. B., " Experimental Verification of Mission Planning by Autonomous Mission Execution and Data Visualization using the NPS AUV II." Proceedings of IEEE Oceanic Engineering Society *Symposium on Autonomous Underwater Vehicles, AUV-92* Washington DC., June 2-3, 1992.

[5] Simon, D., Espiau, B., Castillo, E., Kapellos, K., "Computer Aided Design of a Generic Robot Controller Handling Reactivity and Real Time Control Issues", *IEEE Transactions on Control Systems Technology*, Vol. 1, No. 4, Dec. 1993, pp. 213-229.

[6] Healey, A.J., Marco, D. B., "Slow Speed Flight Control of Autonomous Underwater Vehicles: Experimental Results with NPS AUV II" *Proceedings of the 2nd International Offshore and Polar Engineering Conference*, San Francisco, July 14-19 1992.

[7] Healey, A. J., Lienard, D., "Multivariable Sliding Mode Control for Autonomous Diving and Steering of Unmanned Underwater Vehicles", *IEEE Journal of Oceanic Engineering* Vol. 18, No. 3, July 1993 pp. 1-13

[8] Byrnes, R. B."The Rational Behavior Model: A Multi Paradigm, Tri-Level Software Architecture For Control Of Autonomous Vehicles", Ph.D. Dissertation, Naval Postgraduate School, Monterey CA. March 1993

[9] Byrnes, R., Kwak, S. H., McGhee, R. B., Healey, A. J., Nelson, M. L.,"Rational Behavior Model: An Implemented Tri-Level Multilingual Software Architecture for Control of Autonomous Vehicles" Proceedings of the 8th UUST, University of New Hampshire, Durham, NH. September 27-29, 1993 pp. 160-179

[10] Torsiello, K., "Acoustic Positioning of the NPS Autonomous Underwater Vehicle (AUV II) During Hover Conditions", Master's Thesis, Naval Postgraduate School, Monterey, CA. March 1994, S/N 0102-LF-014-6603

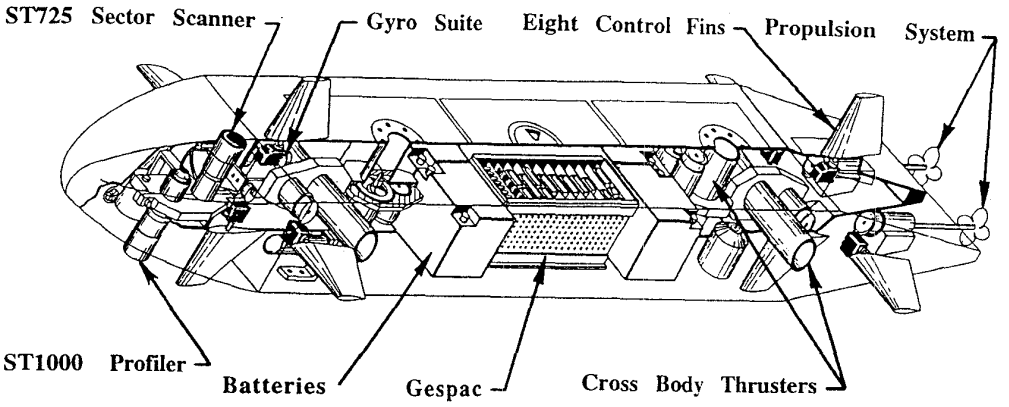

Figure 1. Diagram of the NPS Phoenix Vehicle

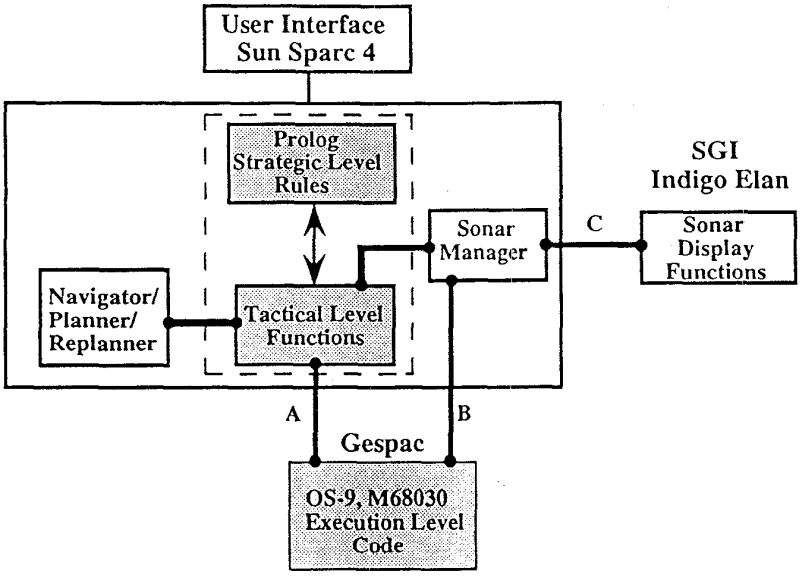

Figure 2. Outline of the Phoenix Networked Controller

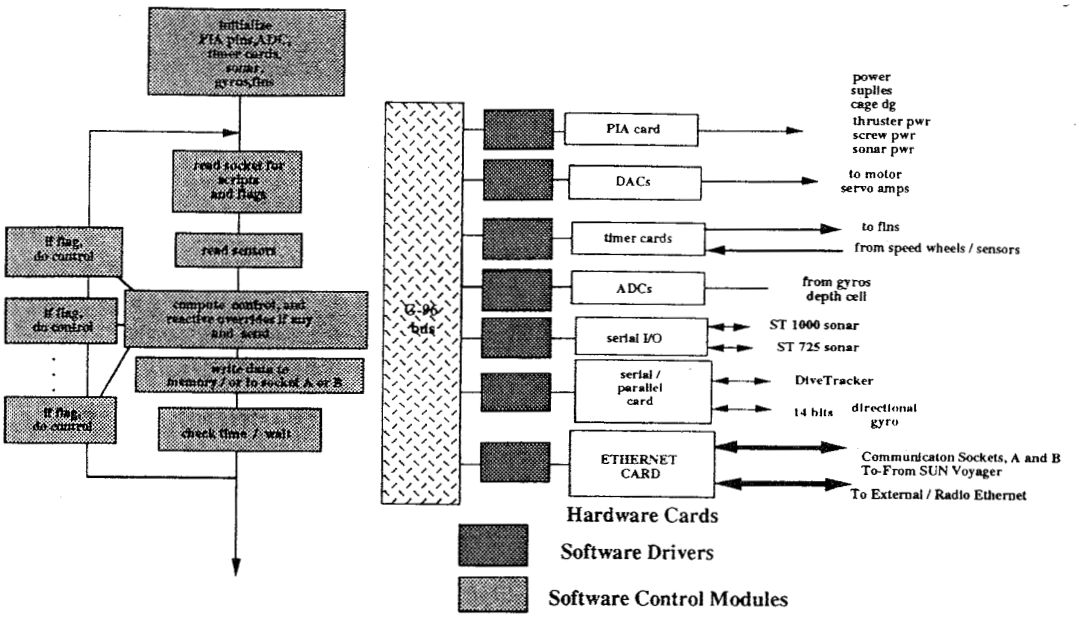

Diagram of the Software / Hardware Interface of the Execution Level of the NPS AUV II

Figure 3. The Structure Of The Execution Level Software

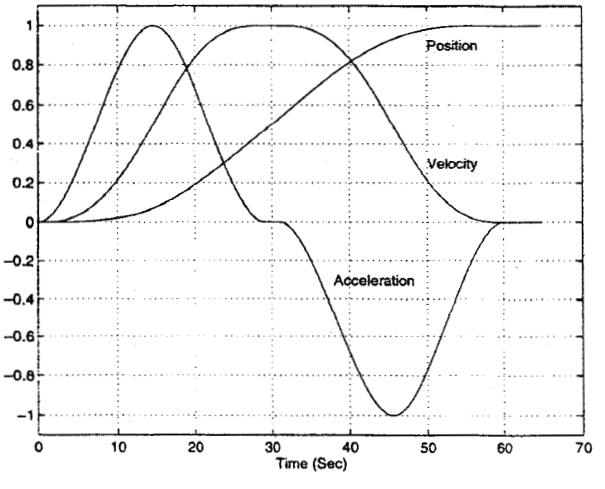

Figure 4. Normalized Command Generator Profiles

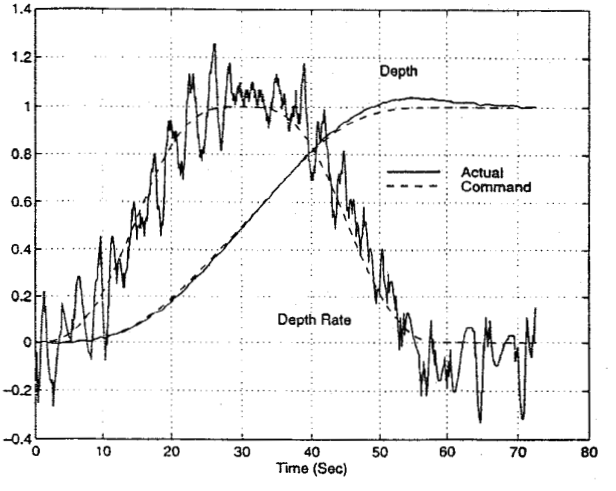

Figure 5. Normalized Depth and Depth Rate Response

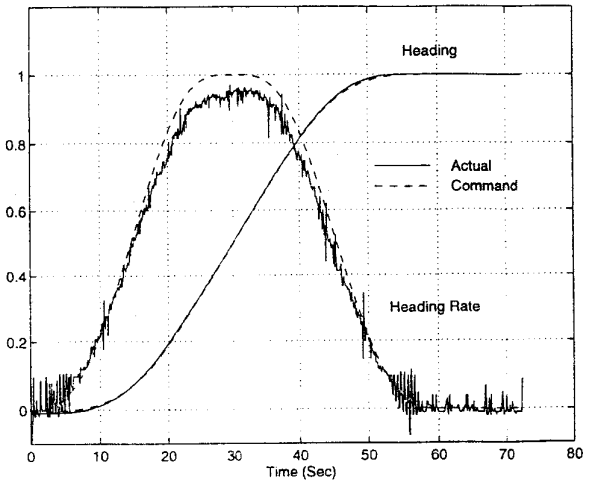

Figure 6. Normalized Heading and Heading Rate Response.

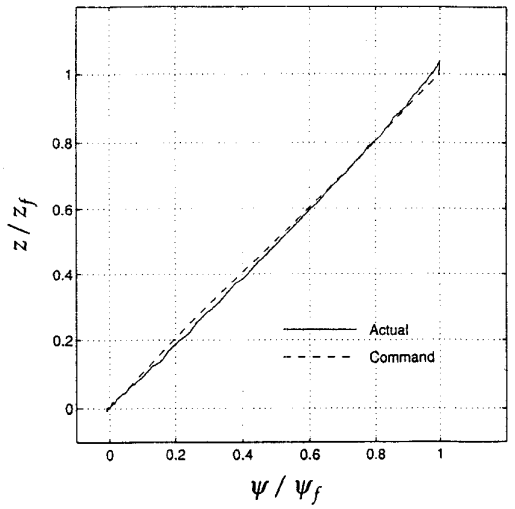

Figure 7. Normalized Depth vs. Heading Response