# A State Configured Sensor Based Control Architecture for an Autonomous Underwater Vehicle

Georgios A. Demetriou & Kimon P. Valavanis*

Robotics and Automation Laboratory
Center for Advanced Computer Studies & A-CIM Center
University of Southwestern Louisiana
Lafayette, Louisiana, USA
(E-mail: gad@acim.usl.edu, kimon@cacs.usl.edu)

## Abstract

A new embedded control system hardware and software architecture suitable for an AUV is presented. The proposed scheme is based on the shared memory principle and consists of three components, a supervisory controller, a functional and a hardware/execution component. Off-the-shelf technology has been used to build and implement the described AUV control architecture. Key features of the proposed architecture include: autonomy, learning, recovery, multiple goals in a single mission and use of multiple sensors for data collection. This embedded architecture is modular, reconfigurable, expandable, upgradable and cost-effective.

## 1 Introduction

The reported research has been motivated by the challenge to design and implement a sensor-based hardware and software control architecture for an Autonomous Underwater Vehicle (AUV). The central research objective is that the AUV will operate autonomously in coastal and shallow water sensitive environments like wetlands, shallow water fisheries and polluted environments. Such an AUV is expected to be used for offshore oilfield platform and pipeline inspection and maintenance, wetlands gain/loss detection, shallow water fisheries monitoring, coastal studies, monitoring environmental pollution due to industrial wastes, thus, assisting in preserving the nation's ecosystems.

The AUV's practical use and functionality constraints have been determined based on operational needs specific to the Gulf Coast Region and the more shallow waters of Louisiana. Key features of the proposed architecture include: a*utonomy, learning, recovery, multiple Goals* [1], m*ultiple Sensors* [1], m*odularity/expandability.*

The rest of the paper is organized as follows: Section 2 reviews some of the existing AUV architectures, compares and summarizes their advantages and disadvantages. Section 3 gives a detailed description of the proposed control architecture. In Section 4 the implementation process of the proposed architecture is explained. Section 5 gives a case study example and demonstrates the system's functionality. Finally section 6 concludes the paper.

## 2 Comparison of Existing AUV Architectures

The dynamics of AUVs are typically nonlinear and uncertain [3]. Nonlinearities arise from the hydrodynamic forces as well as cross coupling between vehicle states. Uncertainties arise from changes in the environment that the vehicle interacts with. To meet the demanding control requirements of AUV control four major control architectures have been developed, called hierarchical architecture, heterarchical architecture, subsumption architecture and hybrid architecture.

*Hierarchical Architecture*: In a hierarchical architecture, the various system components are arranged in levels, with higher levels commanding lower levels. The higher levels are responsible for all mission goals and lower levels are responsible for solving smaller problems that are needed to accomplish the mission [2]. The hierarchical architecture is a representation of a top-down approach which is extremely rigid [3]. Each level in the hierarchy receives commands from the level directly above it and sensory information from the level directly below it. The information exchange rate decreases from the bottom to the top of the hierarchy. Non adjacent layers can not communicate directly with each other. This results in long processing times, unreliable sensory information and unknown reactions to unpredictable events [3]. The tri-level control architecture of the NPS Phoenix AUV is an example of a hierarchical architecture. For a more detail description of the NPS Phoenix the reader is referred to [5, 6].

*Heterarchical Architecture*: To overcome the communication overhead of hierarchical control architectures some researchers have proposed heterarchical control architectures. The individual functional modules are treated as cooperating equals with possible direct communication without any

* To whom all correspondence should be addressed

supervisor. The advantage of this architecture is that knowledge and sensory information can be easily integrated into any module. However, the control representation aspects are difficult to address and relatively complicated issues such as chaotic behavior may arise [27].

*Subsumption Architecture (Layered Control)*: Layers of control are used to let the robot operate at increasing levels of competence [1]. Layers are made up of modules that communicate over low bandwidth channels. Higher level layers can subsume the roles of lower level layers by suppressing their outputs. In the subsumption architecture there is no high-level supervisor (high level of control). Data and control are distributed through out all layers and each layer processes its own information (sensory and commands). The subsumption architecture is decomposed based on the task achieving behaviors of a system. All behaviors are explicitly implemented and then tied together to form a robot control system. Bellingham and Consi [7] modified the subsumption architecture and derived the state configured layered control architecture. The main disadvantage of the subsumption architecture is the difficulty associated with synchronization and timing of the various modules (layers). This difficulty is due to the fact that there is no high-level control. This makes it hard to verify correctness and stability of the system. Another disadvantage is that the complexity of the system increases significantly as the number of behaviors increases. The expandability and robustness that this architecture offers is a great advantage. The real advantage of the layered control architecture is that the behavior network has a relatively low computational overhead. An example of a system that uses the subsumption architecture (in a modified manner), is the Sea Squirt AUV that was developed at MIT [7].

*Hybrid Architecture*: In hybrid control different levels of abstraction are used for system modeling and control purposes [4]. The hybrid architecture is a combination or modification of the other two architectures. The hybrid architecture integrates both low and high level control in a coherent structure [3]. The lower levels of the system have similarities to the subsumption architecture while the desired higher level control (missing from the subsumption approach) has similarities with hierarchical system modeling. An example of a hybrid architecture, is the control architecture of the Ocean Voyager II developed by the Florida Atlantic University [8].

Nine AUV control architectures have been reviewed: *Texas A&M's Autonomous Underwater Vehicle Controller (AUVC)* [9], *Florida Atlantic University's OcearVoyager II* [8], *Massachusetts Institute of Technology's Sea Squirt AUV* [7], *Naval Postgraduate School (NPS) PHOENIX AUV* [5, 6], *Instituto Superior Technico's Marine Utility System (MARIUS) Programme AUV* [10], *Monterey Bay Aquarium Research Institute (MBARI) and Stanford University Ocean Technology's Testbed for Engineering Research (OTTER) AUV* [11], *Massachusetts Institute of Technology Sea Grant's Odyssey AUV* [21], *University of New Hampshire's Experimental Autonomous Vehicle (EAVE) III* [16], *University of Technology at Sydney's ERIC AUV* [16], and *Woods Hole Oceanographic Institution's Autonomous Bentic Explorer (ABE) AUV* [16]. These AUVs have been chosen because the proposed hardware and software sensor based control architecture (presented in the next Section) draws upon the operational and functional principles of the reviewed AUVs. Table 1 compares their control schemes.

## 3 State Configured Sensor Based Embedded Control Architecture for AUVs

The proposed embedded control architecture consists of three components: a supervisory control, a functional control and a hardware/execution component, as shown in Figure 1. The supervisory control component is responsible for the coordination of the overall AUV. It monitors and coordinates the order of module task execution (of the functional component). The functional component is responsible for specific task/operations occurring in a mission. Each module performs a well defined set of tasks. The hardware/execution component consists of the actual electrical, mechanical and sensory components of the AUV and it is directly controlled by the functional component.

The described control architecture is modular and the functionality of each module is determined based on specific tasks performed. All modules share a common communication bus for data storage and retrieval. There is no direct communication between individual modules. Information exchange is accomplished through shared variables. The overall control system architecture functionality is based on state diagrams that define specific AUV operations. A state diagram is responsible for determining the sequence of AUV tasks/operations through the mission various phases. A major advantage of this design scheme is that not all system modules are located at the same level, thus minimizing the delay between sensor readings and data transfer. This delay may be further minimized if the sensor control module (of the functional component) is integrated within the supervisory control component. In such case, the only delay time involved is the time required by the supervisory control component to transfer data from one module to another.

*Supervisory Control Component*

The supervisory control consists of the Master Controller (MC) and the Shared Memory. Figure 2 shows the internal structure of the MC. The MC contains a main/central processor (Intel Pentium), a local memory unit (for execution purposes) and the following four software processes that coordinate the operation of the AUV.

i. *Shared Memory Management (SMM)* process: It takes care of all variable transfers between the modules and the shared memory. These variables are used to transfer information between the modules. Such information is the parameters supplied by the sensor control module (sensor data), the current map file generated by the map generator module, etc.

ii. *Interrupt Handling (IH)* process: It monitors the shared memory for interrupt variables that are sent by the Monitoring and Recovery module if a software or hardware malfunction takes place. If an interrupt occurs, it performs the necessary action (usually stops operation and initiates some kind of recovery function from the Monitoring and Recovery module).

iii. *Monitor External Signals (MES)* process: It monitors the shared memory variables for the signals that the modules send to the MC. These signals include requests for variables, acknowledge signals, and so forth.

iv. *Mission Scheduler (MS)* process (state configured operation): It coordinates the operation of the AUV by transferring control among the modules. The transfer of operation is based on state diagrams. State diagrams represent the mission of the AUV.

The means of communication among the system modules is the shared memory system of the Supervisory Control component. The goal is to make sure that data integrity and synchronization is maintained at all times. This is done using mutual exclusion by employing semaphores [2]. Each variable is assigned a semaphore. To avoid corruption of the shared memory, thrusted functions are used to guaranty that the only relevant parts of the shared memory are accessed by the various processes. The shared memory system is accessible by all the modules of the system. The MC is responsible for the coordination and management of the shared memory (shared memory management process). The various modules need not know about the function of other modules. The only information a module needs is the data stored in pertinent variables is shared memory.

*The Functional Component*

The functional component is composed of modules that function independently of one another. These modules are:

i. *Sensor Control* module: Controls the activities of the sensors and receives information from the sensors. It makes necessary calculations and generates results that are needed by the rest of the modules of the system and sends the results to the shared memory of the Supervisory Control component.

ii. *Map Generation* module: Generates maps of the 3-D environment based on data obtained by the sensors and on any previously known information of the environment.

iii. *Navigation and Task Execution* module: Responsible for generating collision free paths, trajectories over the paths and avoiding obstacles. It generates global paths (from point A to point B), and local paths (sub-point paths between A and B). It is also responsible for performing necessary mission tasks (i.e. pick object, scan object, etc.). It controls the motors, fins, thrusters, gyros, servos, end effector(s) of the vehicle.

iv. *Object Recognition and Classification* module: Obtains the information generated by the Sensors Control module and classifies the objects detected by the sensors. After classification, the objects are stored in the Knowledge Base (KB) of the AUV. The KB database is located in a physical storage device (solid state disk).

v. *Global Positioning System* module: Calculates the correct location of the vehicle in relation to the world (environment).

vi. *Monitoring and Recovery* module: Studies the overall functionality of the system. It locates malfunctions, either software or hardware, and initiates recovery procedures if necessary. It also initiates recovery procedures when the vehicle runs into situations that normal operation is impossible (avoid obstacles, etc.).

vii. *Acoustic Modem* module: Communicates with external, to vehicle, operators or computers. This allows monitoring of the system by external sources.

*The Hardware/Execution Component*

The hardware/execution component consists of the actual mechanical, electrical, electronic components of the AUV. It is beyond the scope of this paper to discuss the actual mechanics of AUVs. Only the communication procedures (protocols) and message/data passing between the hardware and functional components are considered.

## 3.1 Discussion

Given the configuration of the proposed architecture, the functionality and operability of the vehicle is divided into phases (states). A state diagram, that resides in the control level, transfers operation from phase to phase. Each phase requires the coordination and functionality of specific modules. In state configured control, only the goal oriented modules pertinent to the specific phase of the mission are active. The others remain inactive. In this way, power consumption requirements are minimized. The responsibility for ensuring that the modules are activated at the right time and with the right priority is delegated to the state diagram of the MC. The example in Section 5 clarifies such issues. A similar approach was used by Bellingham and Consi in [7]. The main differences of this design are that the communication between the various modules is

accomplished by using shared memory variables, and that the overall operation of the system is coordinated by the MC.

The shared memory is controlled by the MC. Since access to shared memory variables is controlled using semaphores and automatic logging of variables needs time stamps, every variable is accompanied by a semaphore variable and a time stamp variable. Every module has its own local memory (for execution purposes). A process keeps a local copy of any shared memory variable that it needs to access. Before accessing a shared memory variable, a process must set its semaphore and reset it after the variable data is transferred to the local copy of the variable.

This architecture is a state configured embedded control architecture. Single Board Computers (SBCs) are used for one of the modules. Each SBC has its own on board memory and access to any peripherals that it needs. A real-time operating system (QNX) is responsible for the operation of the whole system. The use of SBCs allows the system to be easily expandable, easily upgradable, modular, reliable and very inexpensive.

# 4 Implementation

After reviewing existing technology, it was decided to use the QNX real-time operating system (OS) for software development and the use of single board computers (SBCs) with the STD-32 and CompactPCI standards as the hardware platform for the proposed embedded control architecture.

## 4.1 The QNX Operating System

The QNX is one of the most powerful real-time operating systems (OS) for embedded control systems. QNX is a UNIX-like operating system suitable for applications where real-time multi-tasking performance is an important criterion. QNX Software Systems has updated the initial release of QNX to provide POSIX capabilities and a scalable architecture, two features often required for embedded applications [12, 13, 14].

For embedded applications, the modularity of the OS allows the developer to omit unneeded system processes. With its real-time performance, a reduced-size QNX system becomes comparable to a real-time executive, while delivering the functionality of a POSIX runtime and development environment. In a minimal system, only the micro-kernel, process manager, and system shared library need be present. All other system processes are optional and can be dynamically started and stopped at runtime, or statically bound in at boot time for ROM-based applications. With the addition of the small QNX networking module, an embedded system can become a network-transparent extension of a larger QNX environment for distributed applications, booting either from ROM or from the network [14].

With its micro-kernel, message-passing architecture, QNX can take a network of computers and present them to applications as a "single logical machine," regardless of how many physical computers are joined by the network. Applications developed for this "single logical computer" will run without changes even as the number of computers is scaled to suit the scope of the application. This scalability is possible because QNX encourages applications to be designed as a team of cooperating, communicating processes on a single machine. When run on a QNX network, those processes can be configured to run throughout the network, while QNX provides network-transparent messaging between those processes. The networking allows any process to use any resource on any computer on the network. Disk-less machines can also boot from the network and then use any resource, anywhere. Mission-critical applications are aided by using the network-distributed messaging to implement "hot standby" systems. Multiple redundant network links between network nodes provide protection from network failures as well.

## 4.2 STD-32 SBCs

The STD bus has been the standard bus for industrial control systems since the 1970s. The STD-32 Bus combines a small, industrial strength architecture with the functionality and performance of today's high-end personal computers. This versatile 8-, 16- and 32-bit scalable computer is the right choice for demanding real-time control and data acquisition applications where small system size and cost are important. STD 32 is an open, well designed standard with a wide range of processors, peripherals, industrial I/O, enclosures and complete systems from numerous manufacturers [12, 19].

The STD 32 Bus can run at 32 Mbytes per second for very high-speed data processing applications. Its EISA-like architecture provides more than just a high-performance data path. Other performance characteristics include: Multiprocessing, with centralized arbitration logic to monitor access to the bus, allows the implementation of multiple processors in a single STD 32 system. The 32-bit throughput of the bus is crucial to inter-processor communication in real-time multiprocessing applications; 32-bit addressing and pipelining dramatically improves throughput for block data transfers by reducing bus cycle time and increasing bus bandwidth; High-speed Direct Memory Access (DMA) over the backplane streamlines the operation of data-intensive applications; Slot-specific interrupts expand the number of available system interrupts for servicing systems requests.

A critical component in an STD-32 system is the backplane. The backplane design incorporates several important features including increased backplane signal impedance. A higher backplane signal impedance means "cleaner" signals are sent across the backplane. That is, ringing and reflections are minimized. This is especially

important during signal transitions between the TTL threshold regions of 0.8V and 2.0V. Figure 3 shows a typical STD-32 system. The system shown here is the STD 32 STAR System that is available by Ziatech Corporation. It is a simple, yet extremely powerful approach to the design of real-time control computers. It includes multiple PC-compatible CPU cards in a single card cage (backplane). Each CPU has its own memory and operating system, but shares backplane memory, disks, video and I/O with other CPUs in the system.

### 4.3 CompactPCI SBCs

The newest standard for PCI-based industrial computers is called CompactPCI. It is electrically a superset of desktop PCI with a different physical form factor. CompactPCI utilizes the Eurocard form factor popularized by the VME bus [20]. It is defined for both the 3U (100mm by 160mm) and the 6U (160mm by 233mm) card sizes. CompactPCI has the following features: standard Eurocard dimensions; high density 2mm Pin-and-Socket Connectors IEC approved and Bellcore qualified; vertical card orientation for good cooling; positive card retention; excellent shock and vibration characteristics; metal front panel; user I/O connections on front or rear of module; standard chassis; uses standard PCI silicon manufactured in large volumes; staged power pins for Hot Swap capabilities; eight slots in basic configuration (easily expanded with bridge chips).

CompactPCI is intended as an industrial bus for applications in telecommunication, telephony, real-time machine control, industrial automation, real-time data acquisition and other applications requiring high speed computing, modular and robust packaging design and long term manufacturers support. The CompactPCI bus provides the features and benefit of the PCI bus specifications. Its bus is 32- or 64-bit and its bandwidth is 132 or 264 MB per second [20].

### 4.4 AUV Embedded Control System Implementation

For the design of the sensor based control architecture of the AUV the use of the STD-32 and the CompactPCI bus architectures are chosen. STD-32 offers the use of multiple CPUs (SBCs) within the same system. Each one of the CPUs will function as a module of the overall system. SBCs offer the flexibility, upgradeability, expandability, modularity and reliability that is desired for this system. Using SBCs eliminates a big part of the design process and also the risk of developing a not so reliable control system. The QNX real-time operating system is also chosen as the operating system of the AUV control system. Each module of the system is represented as a CPU in an STD-32 bus configuration. The Ziatech STD 32 STAR System is chosen as the backplane of the control architecture. The STAR System (shown in Figure 3) resembles the control architecture of the AUV shown in Figure 1.

Figure 4 shows the overall system design using the STD-32 STAR System, STD-32 modules and CompactPCI modules. The SSD module is the Solid State Disk that holds the control software of the system and the QNX operating system. It is from this device that the system boots up. The SSD device is also used for storage purposes. When the system starts operation, each one of the modules transfers its execution functions (stored in the SSD) into its local memory and starts execution. The common memory, shown in Figure 4, is accessible to all CPUs for communication. The CPUs operate from local memory for maximum speed.

Currently, the conceptual embedded control architecture design has been completed. An URV, the Phantom S2 ROV (from Deep Ocean Engineering) has been acquired. Its architecture is being modified to convert it to an AUV.

## 5 Example

A simple example, presented in this section, demonstrates and clarifies the functioning of the embedded state configured control architecture.

Consider the simple AUV mission: "*Move from location A to location B*". Figure 5 shows the different phases the AUV has to go through and the modules activated within each phase. The AUV is required to generate a global path from A to B, and local paths between A and B depending on the state of the sensed environment, follow the derived path(s) avoiding obstacles and possibly perform pre-specified tasks at certain points between A and B (for example, stop and take a picture of the surrounding environment). Figure 6 shows the state diagram for this mission and illustrates how control is transferred between the various phases of the mission.

## 6 Conclusion

The described embedded control architecture is simple and cost effective. This architecture is based on individual modules with a master controller (MC) serving as the coordination module for the overall operation of the AUV.

The described design utilizes off-the-shelf components thus minimizing development time. The STD-32 and CompactPCI standards have been tested in many other applications, and they have been widely adopted. Given the hardware platform, the actual control algorithms are implemented at the software level.

Major advantages of the described architecture include:

i. *Learning Capabilities*: The system has a knowledge base (KB) that keeps information that can be used for the current mission or for future missions.

ii. *Recovery Capabilities*: Recovery techniques can be used to bring the system out of unpredictable situations.

iii. *State Configured*: Modification of the mission is easily done by modifying the state diagram. The modules do not need to know about any modifications. All of the mission modifications are done at the control level (MC).

iv. *Expandability/Modularity*: The functionality of the system is easily upgraded or modified. This can be done by removing or adding new modules. Only minor modification to the overall system are necessary in such a case. Only the functionality of the phases and the state diagram need to be changed in order for the system to have a new functionality.

v. *Simple Communication*: The communication of the system is done through the shared memory. The complexity of the subsumption architecture on the overall system design, synchronization and timing, is eliminated. It also eliminates the long processing times, bad sensory and the unknown reactions to unpredictable events that are associated with the hierarchical architecture.

## 7 References


1. Brooks, R. A.: A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, Vol. RA-2, 14-23, 1986.
2. Ganesan, K., Smith, S. M., White, K., Flanigan, T.: A Pragmatic Software Architecture for UUVs, *1996 IEEE Symposium on Autonomous Underwater Vehicle Technology,* 209-215, June 2-6, 1996, Monterey, California.
3. Coste-Maniere, E, Wang, H. H. and Peuch, A.: Control Architectures: What's Going On?, *Proceedings of the International Program Development in Undersea Robotics & Intelligent Control (URIC)* - A Joint U.S./Portugal Workshop, 54-60, March 2-3, 1995.
4. Antsaklis, P. J.: Intelligent Control for High Autonomy in Unmanned Underwater Vehicles, *Proceedings of the International Program Development in Undersea Robotics & Intelligent Control (URIC)*, A Joint U.S./Portugal Workshop, 25-32, March 2-3, 1995.
5. Healey, A. J., Marco, D. B., McGhee, R. B., Brutzman, B. P., Cristi, R. and Papoulias, F. A.: Coordinating the Hovering Behaviors of the NPS AUV II using Onboard Sonar Servoing, *International Advanced Robotics Programme, The 2nd Workshop on: Mobile Robots for Subsea Environments,* 53-62, May 3-6, 1994, Monterey, California, USA.
6. Healey, A. J., Marco, D. B., McGhee, R. B., Brutzman, B. P. and Cristi, R.: Evaluation of a Tri-Level Hybrid Control System for the NPS PHOENIX Autonomous Underwater Vehicle, *Proceedings of the International Program Development in Undersea Robotics & Intelligent Control (URIC)*, A Joint U.S./Portugal Workshop, 78-90, March 2-3, 1995.
7. Bellingham, J. G. and Consi, T. R.: State Configured Layered Control, *Proceeding of Mobile Robots for Subsea Environments*, Int. Advanced Robotics Programme, 75-80, 1991, Monterey, California.
8. Smith, S.: An Approach to Intelligent Distributed Control for Autonomous Underwater Vehicles, *Proceesings of the 1994 Symposium on Autonomous Underwater Vehicle Technology*, July 19 and 20, 1994, Cambridge, Massachusetts, USA.
9. Barnett and McClaran, S.: Architecture of the Texas A&M Autonomous Underwater Vehicle Controller, *IEEE 1996 Symposium on Autonomous Underwater Vehicle Technology*, 231-237, June 2-6, 1996, Monterey, California.
10. Pascoal, A., Silvester, C., Oliveira, P., Fryxell, D. and Silva, V.: Undersea Robotics Research at IST: The AUV MARIUS Programme, *Proceedings of the International Program Development in Undersea Robotics & Intelligent Control (URIC),* A Joint U.S./Portugal Workshop, 111-118, March 2-3, 1995, Lisbon, Portugal.
11. Rock, S. M., Wang, H. H. and Lee, M. J.: Task Oriented Precision Control of the MBARI/Stanford OTTER AUV, *Proceedings of the International Program Development in Undersea Robotics & Intelligent Control (URIC)*, A Joint U.S./Portugal Workshop, 131-138, March 2-3, 1995, Lisboa, Portugal.
12. STD32 - QNX, *http://www.ziatech.com*, 11/22/1996
13. QNX 4.21 from QNX Software Systems, *http://www.vir.com/vir/QNX-info1.html*, 11/24/1996.
14. QNX 4.21 Specifications Overview, *http://www.vir.com/vir/QNX-info3.html*, 11/24/1996.
15. Lehrbaum, R.: Designing with PC/104 - A Tutorial, *AMPRO, Computer Incorporated.*
16. Istituto Automazione Navale, CNR, Control Architectures Database, *http://www.ian.ge.cnr.it/robotics/archit/dbarchit.htm,* 01/16/97.
17. Antsaklis, P. J., Passino, K. M.: An Introduction to Intelligent and Autonomous Control, *Kluwer Academic Publishers*, 1993, Norwell, Massachusetts, USA.
18. Yun, J.: Underwater Robotic Vehicles: Design and Control, *TSI Press*, 1995, Albuquerque, New Mexico, USA.
19. Why STD 32, *http://www.std32.com/Whystd.html*, 01/30/97.
20. CompactPCI, Welcome to the Future of Industrial Computing, *http://www.gespac.com/html/cpci.html*, 02/17/97.
21. Bellingham, J. G., Bales, J. W., Atwood, D. K., Chrissostomidis, C., Consi, T. R., Goudey, C. A.: Demonstration of a High-Performance, Low-Cost

Autonomous Underwater Vehilce, *MIT Sea Grant College Program, Technical Report, MITSG 93-28*, MIT, Cambridge, Massachusetts, USA.

## Acknowledgment:

The Authors wish to thank Dr. Denis Gracanin and Dr. Ramesh Kolluru for their useful suggestions. This work has been partially supported by an NSF grant, NSF BES 95-06771 and LEQSF R-3130.

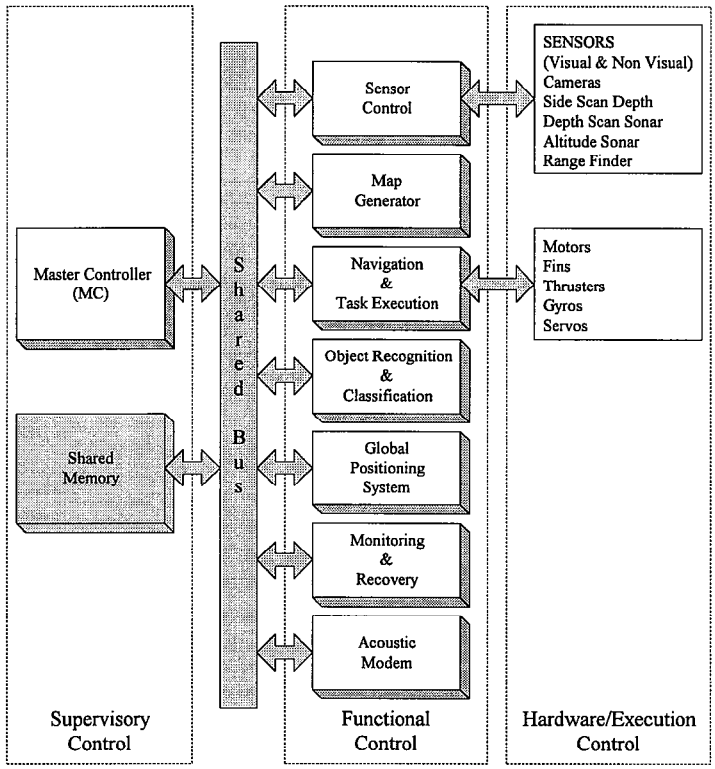


**Figure 1: Block Diagram of the Control Architecture of the AUV**

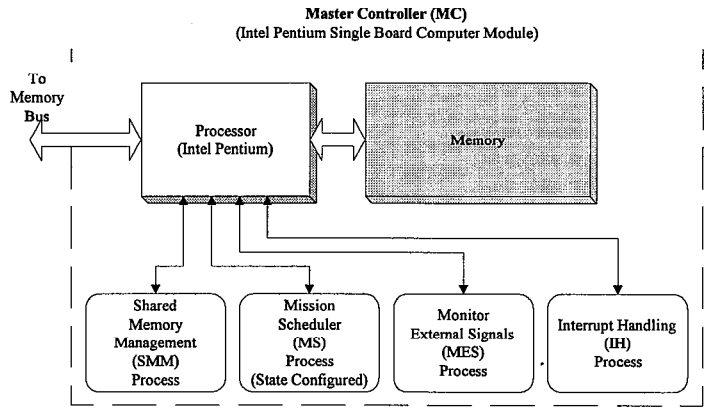


**Figure 2: Master Controller**

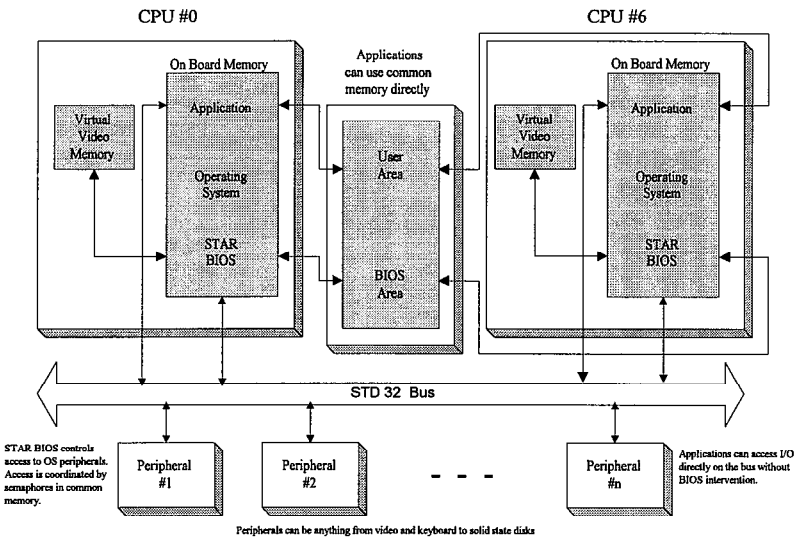


**Figure 3: STD-32 STAR System Architecture**

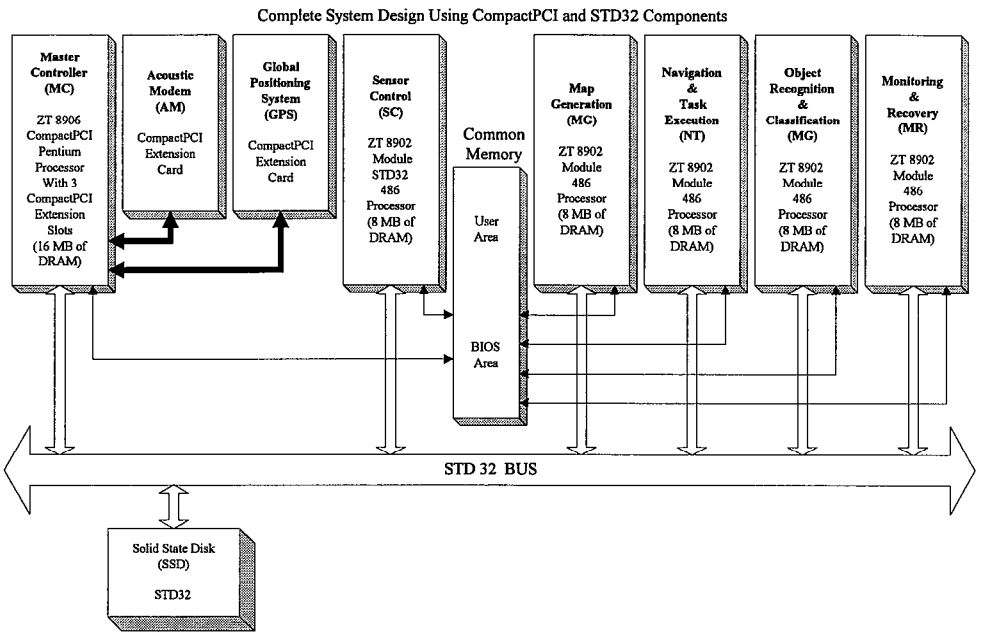


**Figure 4: Control Architecture of the AUV using STD-32 and CompactPCI products**

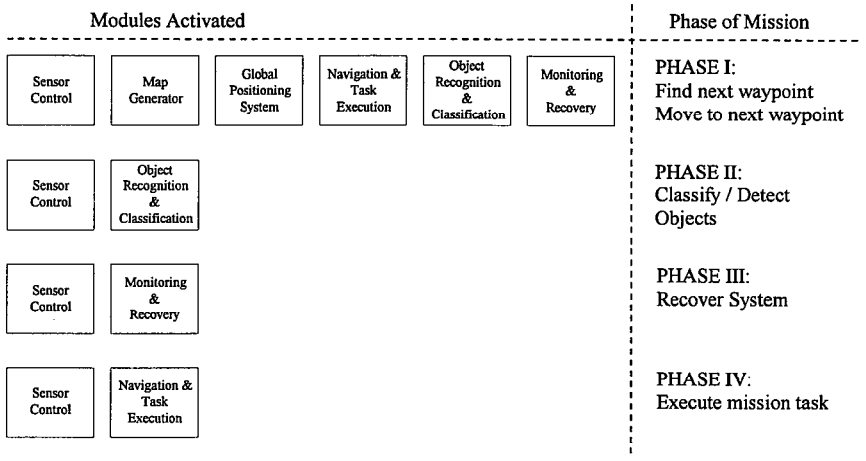


**Figure 5: Phases of the Case Study Mission**

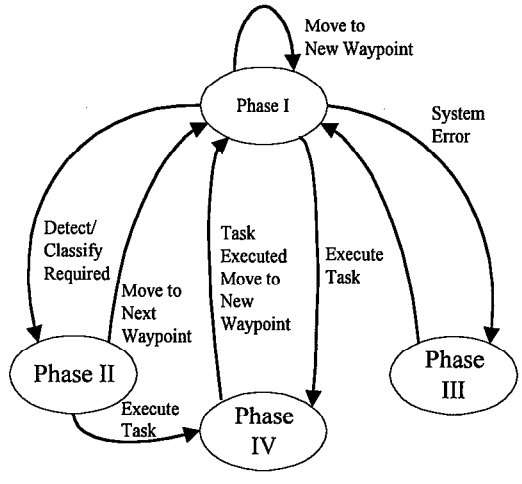


**Figure 6: State Diagram of the Case Study Mission**

**Table 1: Comparison of Existing AUVs**

| AUV | Developer | Hardware | Software | Advantages | Disadvantages |
|---|---|---|---|---|---|
| **AUVC** | Texas A&M. | Processors (nodes) arranged in Plannar-2 Network Topology. | 18 Software processes control the AUV functionality. | Knowledge Base System (KBS) allows it to learn and refer back to its environment.<br>Great recovery procedures.<br>Capabilities to achieve multiple goals simultaneously with its many processors/processes that function independently. | Very complex software. |
| **Ocean Voyager II** | Florida Atlantic University (FAU). | Distributed control system that divides the system into individual control subsystems (nodes) composed of sensor(s), actuator(s) and a micro-controller. The nodes are arranged in a serial communication network. | Each node has its own control software and communication software that allows it to communicate with the other nodes. | The distribution of the system increases reliability and capability while complexity decreases.<br>Simple software.<br>Modular system (software and hardware). | No recovery techniques and no learning capabilities. |
| **Sea Squirt** | Massachusetts Institute of Technology (MIT) . | GESPAC 68020 computer.<br>Micro-controller based subsystems. | OS-9 real time operating system.<br>State configured layered control. | Ability to accomplish multiple goals simultaneously.<br>Modular system.<br>Expandable system. | |
| **NPS Phoenix** | Naval Postgraduate School (NPS). | GESPAC M68030, computer, SUN SPARC-4 computer and SGI Indigo Elan. | A tri-level software control architecture comprising strategic, tactical and execution levels.<br>Execution level is based on the layered control architecture. | Simple software.<br>The execution level offers great expandability. | No learning capabilities<br>Recovery techniques are not very sophisticated. |
| **MARIUS** | Instituto Superior Tecnico (IST). | Hierarchical architecture. | Libraries of primitives (elementary tasks). | Simple guidance system that makes it easy to navigate and operate.<br>The libraries of primitives allows expansion. | Many systems coordinate the operation (it can lead to unpredictable situations). |
| **OTTER** | MBARI / Stanford University. | A network of UNIX workstations. | Object Based Task Level Control (OBTLC). | Simple operation since all functions are performed in the task level. | No reasoning or judgment.<br>Operator is responsible for issuing tasks. |
| **Odyssey** | Massachusetts Institute of Technology (MIT) Sea Grant. | GESPAC 68020 computer.<br>Micro-controller based subsystems.<br>SAIL local area network. | OS-9 real time operating system.<br>State configured layered control. | Simple software.<br>Expandable system.<br>Modular system. | |
| **EAVE III** | University of New Hampshire | Four level layered hierarchical architecture. Each level has a action control and data manipulation. | Four Levels of hierarchy. | Levels are divided according to required speed for data manipulation and action control.<br>Modular. | Complex operation.<br>Complex communication between the levels and modules. |
| **ERIC** | University of Technology in Sydney | Three level subsumption architecture. | Three levels of software each responsible for a different part of the operation. | Simple software.<br>Expandable system.<br>Modular | Timing problems that are associated with subsumption architecture. |
| **ABE** | Woods Oceanographi c Institution | Two layers distributed hierarchical architecture. | Individual modules of software. | Modules are divided into two levels according to computational capabilities.<br>Modular. | Not easily expandable. |