*Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*

# On AUV Control Architecture

**P. Ridao*, J. Yuh+, J. Batlle*, K. Sugihara†**

*Informatics and Applications Institute
University of Girona
Avda. Lluis Santaló s/n Girona CP. 17007 Spain.
{pere,jbatlle}@eia.udg.es

+Mechanical Engineering Dept.
†Information and Computer Science Dept.
University of Hawaii at Manoa.
Honolulu, Hawaii 96822, USA
yuh@eng.hawaii.edu

## Abstract

*This paper surveys control architectures proposed in the literature and describes a control architecture that is being developed for a semi-autonomous underwater vehicle for intervention missions (SAUVIM) at the University of Hawaii. Conceived as hybrid, this architecture has been organized in three Layers: Planning, Control and Execution. The mission is planned with a sequence of sub-goals. Each sub-goal has a related task supervisor responsible for arranging a set of pre-programmed task modules in order to achieve the sub-goal. Task modules are the key concept of the architecture. They are the main building blocks and can be dynamically re-arranged by the task supervisor. In our architecture, deliberation takes place at the planning layer while reaction is dealt through the parallel execution of the task modules. Hence, the system presents both a hierarchical and an heterarchical decomposition, being able to show a predictable response while keeping rapid reactivity to the dynamic environment.*

## 1. Introduction

Modern development in the fields of control, sensing, and communication has made increasingly complex and dedicated robot systems a reality. Used in a highly hazardous and unknown environment, the autonomy of the robots is key to a mission solution. Control architecture is a framework that manages both the sensorial and actuator systems, thus enabling the robot to undertake a user-specified mission.

Since the *Shakey* robot was presented in 1971, a great number of control architectures have been implemented and applied to mobile robots, underwater robots, robots for planetary exploration, and so forth. Different approaches to autonomous underwater vehicle (AUV) control have been discussed in the literature [1, 2, 3, 4, 5, 6]. They are usually classified into three main categories: deliberative, reactive and hybrid.

### *Deliberative Architecture*

Deliberative architectures are based on planning and also on a world model. They allow reasoning and making predictions about the environment. Data flows from sensors to the world model (bottom-up), which is used to plan new actions to be undertaken by the actuators (top-down). When dealing with a highly dynamic environment, the delay in the response time is the main drawback.

This approach is used in the Planning Software Architecture proposed by Hall *et al.* [6]. It is a hierarchical planner arranged into three homogeneous layers. Rock and Wang [11] described an architecture applied to OTTER. It has a three level control structure including a *task level*. Barnett *et al.* [12] used a deliberative architecture called AUVC. It is organized in three hierarchical levels: planning, control, and diagnostic.

### *Reactive Architecture*

Behavioral architectures, also known as reactive architectures or heterarchies, have been discussed in the literature [7, 8, 9]. Decomposition is based on the desired behaviors for the robot and missions are normally described as a sequence of phases with a set of active behaviors. The behaviors continuously react to the situation sensed by the perception system. The robot's global behavior emerges from the combination of the elemental active behaviors. The real world acts as a model to which the robot reacts, based on the active behaviors. As active behaviors are based on the sense-react principle, they are suitable for dynamical environments. Since each behavior pursues its own goal, reaction actions issued by one behavior may cause another behavior to deviate from its respective goal. Then, at times, the robot behavior is not predictable.

The work on reactive architectures was started by Brooks [7] who proposed the subsumption architecture that layers the control system in a parallel set of competence-levels, tying the sensors with the actuators. It uses a priority arbitration through inhibition and/or supression. Bellingham *et al.* [13], adapted the subsumption architecture to the Sea Squirt AUV. They also extended the arbitration mechanism by proposing the *masking*. Zheng [14] introduced the cooperation concept within a subsumption-like control architecture. He also used a layered sensing subsystem dealing with fault tolerance. Payton *et al.* [15] used a behavioral approach to build a



0-7803-6348-5/00/$10.00 ©2000 IEEE.

distributed fault tolerant control architecture. Boswell and Leany [16] applied a layered control architecture to the Eric underwater robot, introducing protected and hormone modules. The Distributed Vehicle Management Architecture (DVMA) was presented by Fujii and Ura [17]. The basic idea is that a behavior can be created by a combination of specific functions given to the robot, and a mission is accomplished by the robot performing sequentially appropriate behaviors. Recently, Rosenblatt *et al.* [18] have applied the Distributed Architecture for Mobile Navigation (DAMN) to the Oberon submersible. Its main feature is the coordination mechanism that is a competence scheme using a voting mechanism to select an appropriate action.

***Hybrid Architecture***

Hybrid architectures take advantage of the two previous architectures while minimizing their limitations. They usually consist of three layers [10]: (1) the deliberative layer, based on planning, (2) the control execution layer, and (3) a functional reactive layer.

Bonasso [19] described a situated reasoning architecture applied to the Hylas underwater vehicle. Behavior coordination follows a competitive approach and the architecture also includes a deliberative layer. The RBM architecture developed by Healey *et al.* [20] is organized in three levels: *Execution level, Tactic level,* and *Strategic level.* Borrelly *et al.* [21] presented an Open Robot Controller Computer-Aided Design Architecture (ORCCAD). They introduced the concepts of robot-tasks (RT) and module tasks (MT). A mission is built by sequencing RTs and defining event handlers. Borges *et al.* described the DCA architecture [22] that permitted the real-time parallel execution of tasks. It was based on a hierarchical structure consisting of three levels, *Organization, Coordination,* and *Functional* level. Choi *et al.* [23] developed an architecture for the ODIN AUV. It uses a supervisor to handle mission parameters on the basis of lower-level information and three separate blocks: *Sensory database, Knowledge base,* and *Planner.* Recently, Valavanis *et al.* [2] presented the state-configured embedded control architecture which is organized in two-layers: (1) a supervisory control level and (2) a functional control level. This architecture uses a Master Controller (MC) to coordinate the operation of the AUV by transferring control actions among several functionally independent modules.

***Summary***

Many control architectures recently proposed converge to a similar structure that addresses the use of reusable and modularized software packages such as task modules and behaviors that are linked together for both predictability and reactivity.

The paper is organized as follows: section 2 presents the Intelligent Task-Oriented Control Architecture (ITOCA) that is the high-level control of the SAUVIM vehicle. Section 3 deals with the simulation environment and the results, and section 4 reports the conclusions.

## 2. The Intelligent Task-Oriented Control Architecture

This section describes the ITOCA being developed for a new semi-AUV, SAUVIM, a research project funded by the U.S. Navy to design and build a semi-AUV for intervention missions. Its dry weight is about 6 tons and design depth is 6,000 m. It has multi-CPUs in VxWorks OS, various sensors, and a robotic manipulator. The SAUVIM is described in [24]. ITOCA is a hybrid control architecture organized into three layers (fig.1):

- Execution: contains sensor and actuator groups.
- Control: contains the vehicle low-level controllers. It is in charge of the non-linear control of the vehicle and the arm.
- Planning: is in charge of the high level control of the vehicle during the mission. It is responsible for the mission planning, execution, and supervision.

Each of these layers is described below.

***Execution Layer***

This layer is responsible for the interface between vehicle hardware, comprised of the sensor group and the actuator group. Sensors are accessed by the controllers of the upper layer and by the remains of the architecture components. The actuator group is responsible for interfacing with the hardware actuators.

***Control Layer***

The Control layer has the low-level servo control of the vehicle. For SAUVIM, the Adaptive Learning controller [25] is used to control the vehicle in 6 DOF. Fig.2 shows the low-level controller, where $\eta$ is the actual position and $\eta_d$ is the desired position.

***Planning Layer***

A user delivers a mission to the vehicle by using a Graphical User Interface (GUI). The mission is decomposed by the planner supervisor into a sequence of sub-goals. Each sub-goal has a related task supervisor responsible for arranging the task modules in a suitable configuration for the sub-goal undertaken. Task modules are the main building blocks of our architecture. Each task module is designed to perform a well-defined task and have a solid solution for it. Task modules read input values from sensors, other task modules or the task supervisor, and use the Task-Processing-Function to compute the outputs to be sent to other task modules, or the task supervisor (fig. 3).

Within the task module, fault tolerance is addressed by the fault-tolerance function. Some task modules are related to one low-level controller in the control layer. They have a one-to-one relationship with the physical hardware. The initial values for each controller of each task module are transferred from previous task modules or given by the task supervisor.

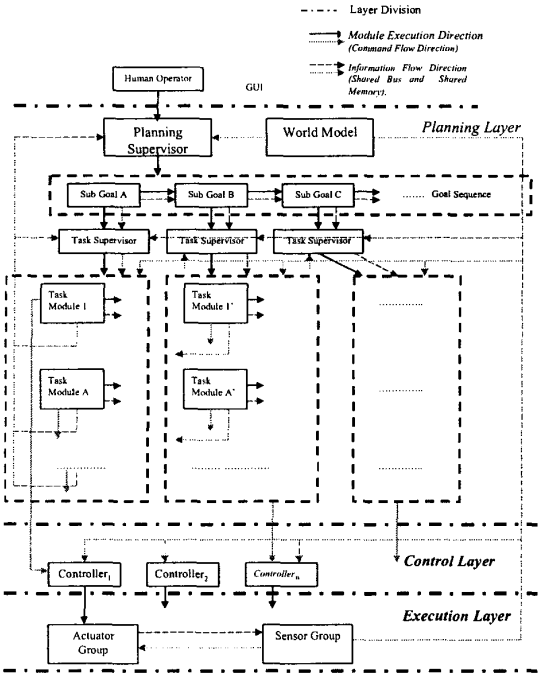


Fig.1 Diagram of the Control Architecture

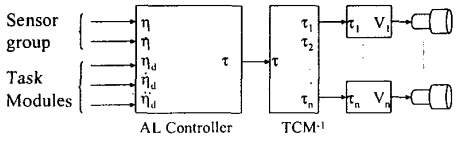


Fig. 2 Low-Level Controller

## Task modules

As stated above, task modules are generic components used as building blocks of our architecture. There are two types of task modules: Motion task modules and Functional task modules. Motion task modules are responsible for sending set-points to the low-level controller. Some of these modules being worked on include:

- Transit: moves the vehicle along the point-to-point local path.
- Repulsion: avoids obstacles.
- Tracking: follows an object of interest or a sequence of given points.
- Stroll: wanders around.
- Navigation: generates the global path (sequence of way-points).
- Hover: keeps the vehicle position.

Functional task modules are responsible for computing data structures needed by other task modules of the architecture. Some of these modules being worked on include:

- Data-sampling&analysis: samples environmental data for situation detection and exception generation.
- Photographing: takes images about the environment.
- Sleep: waits until new command.
- Local Map Building: builds a local map of the vehicle surroundings.
- Weight Average: merges the output of the enabled Motion task modules into a unique set point to be sent to the low-level controller.

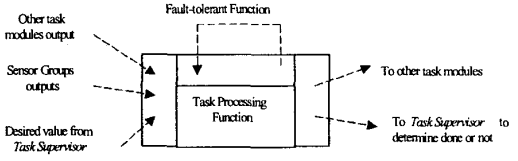


Fig. 3 Task modules

## Sub-goals

Sub-goals for AUVs include Docking, Survey, Searching, and Station Keeping. Organization of the task modules for each of these sub-goals includes:

- Docking={Navigation; Repulsion, Local Map Building, Sleep, Approaching/Departing routine, Weight Average}: approach to a docking position for data transmission or recharging the battery.
- Searching={Navigation; Repulsion; Local Map Building; Stroll; Data sampling & analysis module, Weight Average }: looking for objects of interest.
- Station keeping={Navigation; Repulsion; Local Map Building; Sleep; Hover; Weight Average}: keeping the vehicle position at the desired location.
- Survey={Navigation; Repulsion; Local Map Building; Transit; Stroll; Data Sampling & Analysis; Tracking; Photographing; Weight Average}: navigating through way-points while updating the world model.

Consider the execution of the survey sub-goal as an example. When a survey sub-goal is issued at the planning layer, the corresponding task supervisor starts the correspondent task modules (Fig.4). Initially only the Navigation task module is enabled. Once the path is available, the Transit task module is responsible for driving

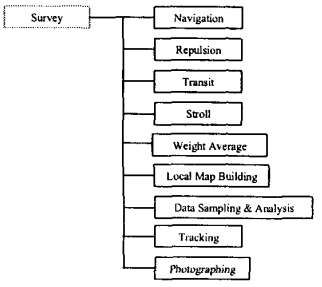


Fig. 4 Task modules used in the Survey Sub-Goal

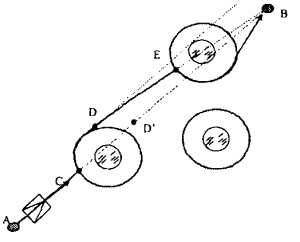


Fig.5 An example of navigation.

the vehicle from way-point A to way-point B following a straight line (fig.5). For each coordinate, it computes the speed that has to be used in order to do an isochronal movement (all coordinates start and end the movement simultaneously). While the path is free of obstacles, Transit is the unique enabled task module, hence the output of the weight average module matches the Transit output. When an obstacle is detected along the path by the sensor-group (point C), the Local Map Functional task module registers it within its internal data structures. Repulsion task-module is then enabled and its output is merged with the output of Transit. The final set points direct the vehicle on a path free of obstacles. When the robot has surpassed the obstacle, Transit must re-evaluate its output (point D). This is needed since, the robot position is far from the predicted one (point D'), hence it must readjust the speed in each coordinate. In fact, Transit is continuously re-adjusting its output speed. At point E, the situation is a bit different, since the repulsion from the obstacle cancels the Transit output. This is the well-known local minimum problem. When the survey supervisor recognizes that the speed set point has dropped to zero, it enables the Stroll task module for a while in order to break the equilibrium point. After a while it is disabled another time. Finally the vehicle reaches point B.

## *The World Model*

The World model plays an important role within our architecture, since it is only partially known. Before operating the vehicle, a survey is conducted in order to make a coarse-grained model of the environment. The model consists in a net of points where the altitude is known. Nevertheless, between two nodes of this net we have no information about the environment. Hence, the vehicle must rely on its own sensors to sense and react. This can be done using reactive navigation. While the vehicle carries out a mission, it continuously updates the model. The world model resides in the shared database and can be accessed when needed by any architecture component like task modules, task supervisors, and/or the plan supervisor.

## *The Local Map*

While the robot navigates through an unknown part of the environment, a local map is built to keep track of the vehicle's surroundings. It is a double resolution grid. The high resolution squares shown in fig.6 represent the sonar readings while low resolution squares represent spatial zones where sonar pings have detected obstacles. In the example shown, low resolution items are about 2x2 meters and there are 10x10 high resolution items for each low resolution one. This double resolution allows a fast access while keeping an accurate knowledge of the environment. A counter of the readings concerning each low resolution item is computed. When this counter is greater than a threshold, it is considered a fixed obstacle and then it can be updated within the world model. Otherwise, this counter is decremented periodically (aging process) and the item is removed when it drops to zero. This is necessary to remove false objects due to sonar glitches and/or moving obstacles. Then, only contrasted objects are kept within the map.

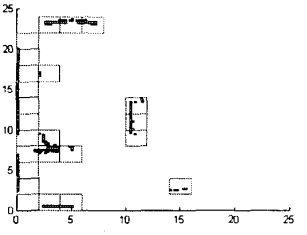


Fig.6. Local map built while navigating the environment shown in fig.8.

## *Coordination of the Task Modules*

The outputs of the Motion task modules are the speed set points for the vehicle. Since more than one task module can be enabled simultaneously, some coordination mechanism is needed in order to generate a unique set point to the low-level controller. There are two main approaches to do this: competition and cooperation. Using competition, the output of one of the task modules is chosen as the output to be sent to the low-level controller. In the case of cooperation, the output of all the enabled task modules are

conveniently merged into a unique set point for the low-level controller. In this paper we focus on cooperation using a weight average approach. This means that the task supervisor chooses a weight (gain) for each task module. Then, the Weigh Average module computes the average. Finally, the desired speed is adjusted using a trapezoidal, and then integrated and differentiated to compute the desired position and acceleration, respectively. Although at this moment fixed gains are used, in the future, new techniques will be explored for choosing online the gains which will allow a new way of adaptation.

## 3. Simulation

Survey sub-goal in ITOCA was simulated with an AUV to show how to achieve reactive navigation within ITOCA using different task modules. A Matlab/Simulink simulator was built (fig.7) including blocks for a dynamic model of the vehicle, the low-level controller, the ITOCA, and a virtual environment through which the vehicle is moved (fig.8). In the simulation, we used the dynamic model of the ODIN vehicle. The model is shown in equation 1 [29].

$$\upsilon' = \left(M_{RB} + M_A\right)^{-1}\left(T + G(\eta) - D(\upsilon)\cdot\upsilon - \left(C_{RB}(\upsilon) + C_A(\upsilon)\right)\cdot\upsilon\right)$$

$$\upsilon = \int \upsilon' \; dt \qquad \eta' = J(\eta)\cdot\upsilon \qquad \eta = \int \eta' \; dt$$

*where*:

$\upsilon'$, $\upsilon$ : acceleration and speeds vector; $\eta$ : position & orientation vector

$M_{RB}$, $M_A$ : rigid body inertia and added mass matrixes

T : input forces vector; $G(\eta)$: gravity and buoyancy vector

$D(\upsilon)$ : Hydrodynamic damping matrix

$C_{RB}(\upsilon)$, $C_A(\upsilon)$: rigid body and added coriolis matrixes

$J(\eta)$ : angular velocity transformation

Eq. 1. Dynamic model of the vehicle

The numerical value of these parameters can be found in ref. [28]. These equations are used for the *"ODIN MODEL"* block (fig.7). In the same figure, the *"Underwater Environment"* block is used for monitoring the vehicle movement across the virtual environment (fig.8) and for the simulation of the sonar readings. The block labeled *"AL Controller"* is the low-level controller and the numerical gains used for the simulation can be find in ref. [28]. The blocks labeled *"Survey Supervisor"*, *"Transit"*, *Stroll"*, *"Repulsion"*, *"Local Map"* and *"Weight Average"* correspond to the respective task modules of the ITOCA planning layer.

### ***Results***

The results described hereafter correspond to different simulation experiments performed to test the ITOCA architecture. The virtual environment for the experiments corresponds to the swimming pool of the University of Hawaii at Manoa (fig.8), where actual experiments are

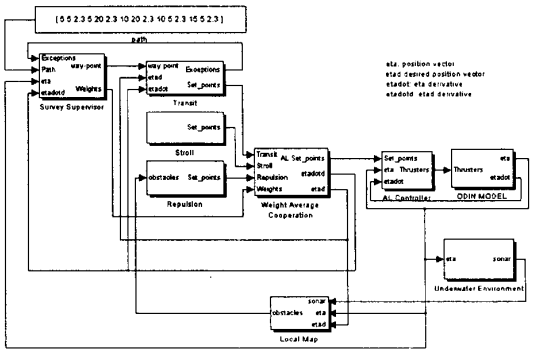


Fig.7 Matlab/simulink simulation.

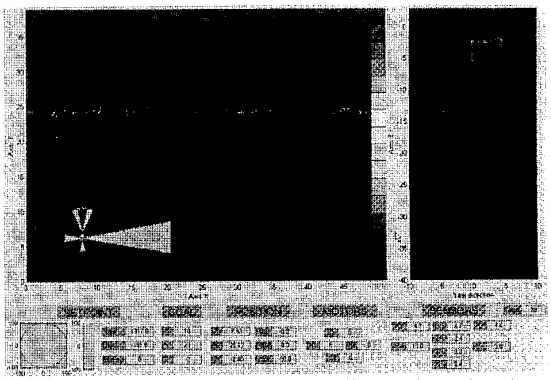

Fig. 8 Virtual environment corresponding to the swimming pool

scheduled for the summer'2000. Hence, in the future, simulation and actual experiments will be compared. In all the experiments, the robot initial position was (2.5,2.5,2.3). The first experiment was used to test the Repulsion task module. The gain of Repulsion was set to one and other gains to zero, hence only the Repulsion module was operational. A radius of 5 meters was considered as the action radius of the detected obstacles (the swimming pool walls). As shown in fig.9, the vehicle moved away from both walls and reached the point (5,5,2.3) where it remained static. The goal of the second experiment was to test the Transit task module, so its gain was set to one while other gains were set to zero. The input path was [(5,5,2.3), (5,20,2.3), (10,20,2.3), (10,5,2.3), (15,5,2.3)] and there were no obstacles in the environment. Fig.10 shows the result. The third experiment tested the cooperation between the task modules Repulsion and Stroll, hence both gains were set to one. Other gains were set to zero. The result (fig.11) shows the robot moving away from the walls and then wandering across the environment. Note that it always keeps x>5 and y>5 (5 being the action radius considered for the obstacles). Finally, the fourth experiment

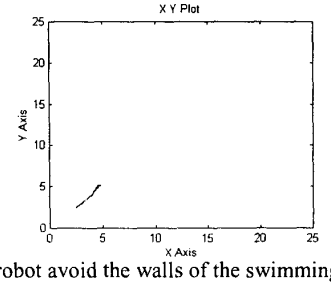


Fig. 9. The robot avoid the walls of the swimming pool.

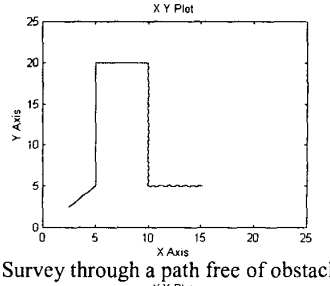


Fig.10 Survey through a path free of obstacles

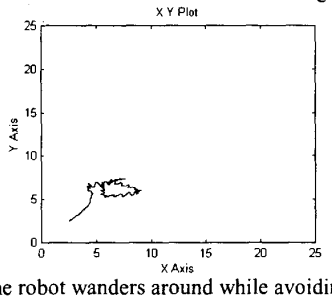


Figure 11. The robot wanders around while avoiding contact with the walls

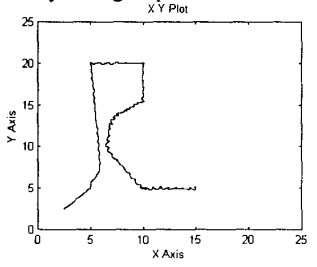


Fig.12. Survey through obstacles.

corresponds to the survey sub-goal as it has to be executed in real environments. Initial Transit and Repulsion were enabled (gain one) and Stroll was disabled (gain zero). The input path was [(5,5,2.3), (5,20,2.3), (10,20,2.3), (10,5,2.3), (15,5,2.3)], and the environment is shown in fig.9 and fig.12 shows the result.

***Discussion***

As discussed above, reactive navigation with appropriate strategies is a good approach to navigate the robot when knowledge about its environment is not available. Nevertheless, some situations (like the local minimum) can trap the vehicle. Escape behaviors like Stroll or others proposed in the literature [27] can help the vehicle to recover from these types of traps. However, due the locality of the knowledge used for the reactive navigation, there will always be a complex environment where the reactive system alone cannot help the vehicle find a path to the goal position or will take too long to find its path. This failure must be handled by the task supervisor by using path planning [26]. Therefore, the reactive system is responsible for using all the available strategies in trying to achieve the way-point, while the deliberative system must re-plan when the strategies fail. Results of a hybrid system merging the reactive and the deliberative approach will be presented in the future.

## 4. Conclusions

In this paper, different control architectures were surveyed and a new architecture was described with recent simulation results of reactive navigation. The ITOCA architecture focuses on the task modules as the basic elements in the decomposition of a mission, and they are also the elements that enable the system to achieve reactivity. A hierarchical structure of the architecture that has three layers was adopted, and in the planning layer lateral decomposition of sub-goals in task modules was adopted. The hierarchical deliberative structure is produced by the planner according to the world model in order to get a predictable scheme of the execution of the mission. Reactivity is guaranteed through the parallel execution of the task modules coordinating sense and action. Exceptions are handled by the task supervisor provoking changes in the organization of the task modules corresponding to the sub-goal in execution.

**References:** Due to the limited space, a list of refs. is available at http://eia.udg.es/~pere/iros2000