An Underwater Autonomous Agent. From simulation to experimentation.

Pere Ridao, Joan Batlle, and Marc Carreras

Computer Vision and Robotics Group
Intistute of Informatics and Applications, University of Girona
Edifici Politècnica II, Campus Montilivi, 17071 Girona, Spain
Tel: +34 972 418 767, Fax: +34 972 418 098
{pere,jbatlle,marcc}@eia.udg.es

Abstract

The development of an Autonomous Underwater Vehicle is a complex process that usually needs different sub-process like simulation, implementation and experimentation. Frequently these processes are disconnected among them and require rewriting the code increasing the probability of error. This paper presents an hybrid control architecture, named OOCAA, for an underwater agent. The OOCAA, Object Oriented Control Architecture for Autonomy, was designed as an object-oriented architecture that simplifies the interconnection of the different processes commented above. Simulation and real execution are interconnected through graphical applications allowing the use of the real code and real hardware of the robot to simulate a mission in the laboratory. Once the code is tested in the laboratory, real experiments can be done. The paper describes the overall OOCAA architecture, the reactive layer that has been already implemented and the tools used for simulation, implementation and experimentation. The control architecture is being implemented in the GARBI underwater robot.

Keywords: Autonomous Agents, AI architectures, Reactive Control, Robotics, Real Time Systems.

1 Introduction

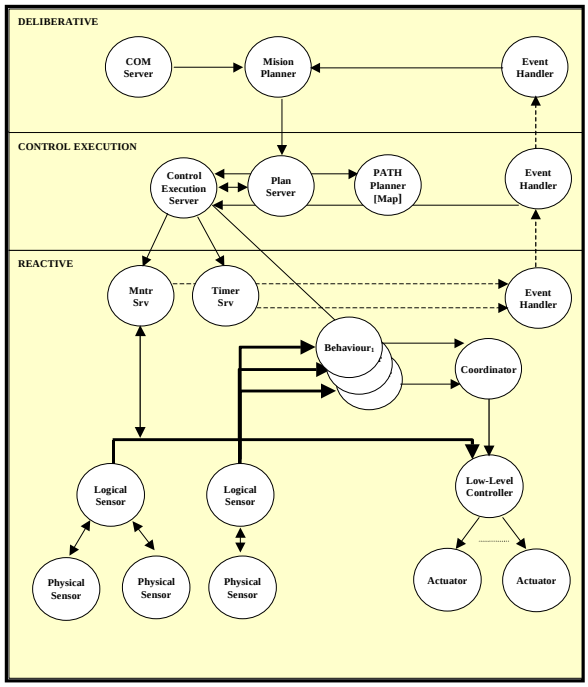
An autonomous vehicle may be defined as "a vehicle with a sensorial system and an actuator system, managed by a control architecture and able to undertake a user specified mission". Therefore, an architecture is a framework in which the following processes are carried out: sensing, control laws, errors detection and recovery, path planning, tasks planning and monitoring of events during the execution of a particular 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. Furthermore, in all these cases, three main approaches have been applied. The first, deliberative architectures, which are strongly based on planning and on a world model, allow reasoning and making predictions about the environment. The second, behavioural architectures, also known as reactive architectures or heterarchies, are decomposed based on the desired behaviours for the robot. Normally, these missions are described as a sequence of phases with a set of active behaviours. The behaviours continuously react to the situation sensed by the perception system. The robot's global behaviour emerges from the combination of the elemental active behaviours. The real world acts as a model to which the robot reacts based on the active behaviours. As active behaviours are based on the sense-react principle, they are very robust and quite suitable for dynamic environments. The third approach, hybrid architectures, takes advantage of the two previous ones, minimising their limitations. Usually, these are structured in three layers: (1) the deliberative layer, based on planning, (2) the control execution layer (turn on/turn off behaviours) and (3) a functional reactive layer. Comparative studies of these different approaches in the field of underwater robotics have been recently published in the literature [6] and [8].

This paper presents an overall description of the OOCAA (Object Oriented Control Architecture for

Autonomy) hybrid architecture. This architecture has been designed to control the GARBI robot, see Figure(2). The architecture has 3 layers: Deliberative, Control Execution and Reactive. The paper also describes the reactive layer that is already implemented in the robot. As a second goal, the paper summarises the different tools that have been used to design and test the reactive layer. The processes involved have been: simulation, implementation and experimentation. Concluding, the global purpose of the paper is to show the different processes of a real design and implementation of a control architecture to give autonomy to an underwater agent.

The paper is organised as follows. Section 2 presents an overall description of the OOCAA. Section 3 describes the reactive layer of the architecture. Section 4 briefly describes the robot GARBI. Section 5 shows the simulation process used to design the architecture. Section 6 describes how implementation is done. Section 7 shows some experimentation tools used to test the architecture. Finally the paper is summarised in section 8.



Figure(1). OOCAA three-layer control architecture.

2 Overall description of the OOCAA architecture

The OOCAA is control architecture to give control autonomy to GARBI robot. Also it can be adapted to another underwater vehicle or, contemplating only the two horizontal dimensions, to a land vehicle. Its structure is arranged in three layers, see Figure(1): Deliberative, Control Execution and Reactive.

The Deliberative layer is used for mission planning. It is in charge of inserting new tasks in the plan structure trying to minimise cost function. The planning process takes place when the user specifies a new task or when some previously planned task fails and needs re-planning. Planning and execution is de-coupled so both processes can be executed concurrently.

The Control Execution layer is responsible for plan representation and controlling its execution. The plan is represented as a Finite State Machine where each state is related to one task. The control execution layer makes the actual state evolve from the begin state to the end state through a sequence of states. For each state, the execution of the related task means turning on or off a set of behaviours.

The reactive layer is in charge of the real-time control of the vehicle. This layer provides three reactive mechanisms: (1) behaviours, (2) monitors and (3) timers. Each behaviour has its own goal and can be executed independently or concurrently with the others. There are safety behaviours, such as obstacle avoidance or surfacing if there is water leakage. There are also navigational behaviours such as going to one point and bottom following.

3 The reactive layer of the OOCAA

The reactive layer is responsible for the real-time control of the underwater vehicle. This is achieved by using the following constructions: (1) behaviours, (2) monitors and (3) timers. This layer is responsible for interfacing with the vehicle sensors doing sensor fusion when needed. It is also in charge of interfacing with the low-level controller.

3.1 Sensor Sub-system

The sensor sub-system is arranged hierarchically. For each robot sensor, there is a physical sensor object which acts as an interface. This entity contains the code and data needed for accessing the sensor and for data filtering. A set of related physical sensors are grouped in a logical sensor. They provide physical magnitudes as inputs for the behaviours. For instance, the logical sensor, which provides the angular speed (yaw derivative), makes use of the physical compass

sensor while the logical sensor providing the x position makes use of the physical linear speed sensor and the physical compass sensor. When needed, sensor fusion takes place at the logical sensor level. For instance, a robot containing two compasses will have two physical sensors, and one logical sensor which merges the values of the two previous ones. When redundancy in the sensor sub-system exists, the homogeneous sensors (those which provide the same physical magnitude) can be fussed in a unique logical sensor. In future, this could be useful for fault tolerance, allowing the system to continue functioning when one of these physical sensors fails.

3.2 Robot Behaviours and Coordination

The control of the robot is broken down based on the desired behaviour. A robot-behaviour is a control module which is responsible for sensing the actual situation using the perception and reacting sub-systems driving the robot towards the desired goal. Each behaviour has its own goal. More than one behaviour can be active at the same time. In this case, the coordinator module is responsible for merging the outputs of the active behaviours. A complex behaviour emerges from the combination of individual behaviours. When dealing with behavioural robotics, two important questions arise: (1) how can we define a robot-behaviour? and (2) how elemental behaviours are merged? With respect to the first question, fuzzy logic for behaviour-description is used, and with respect to the second, different strategies are described in section 6 (weight average, priorities). Other architectures like those described in [4] or [9], among others, use fuzzy logic for behaviour description. Using fuzzy logic, behaviours can be defined simply, since fuzzy IF-THEN rules are close to human reasoning. Hence, in our architecture, a behaviour reads its input variables from the logical sensors, and uses a set of fuzzy rules to compute the outputs. These outputs are the set-points of the low level controller which is a Fuzzy-like PD controller [5] in charge of the non-linear control of the vehicle. The inputs to this controller are the Yaw, Depth and Speed set-points.

3.3 Monitors

Monitors are reactive constructions used for situation recognition. They are in charge of generating events whenever a pre-programmed situation is detected. These events will be handled hierarchically by the event-handlers which, at different levels, can enable/disable behaviours, abort a task, fire a re-planning process, communicate monitoring information to the user or even abort the entire mission. A monitor can be thought as a black box with several inputs and one output. The inputs are the values read from the logical sensors and the output is an event which is sent to the Reactive Event Handler. There are two kinds of monitors: (1) polling monitors and (2) Interruption Monitors. Polling monitors are independent processes which evaluate the condition at a pre-programmed frequency. On the other hand, interruption monitors are tightly linked with the sensor sub-system. Whenever a new value is available for one sensor, it is sent to the related monitor. When all the values needed for evaluating the condition are available, it is evaluated.

3.4 Timers

Timers are the constructions used for generating time-based events. They are mainly used for computing task deadlines.

3.5 Reactive Event Handler

At the Reactive layer, events are processed by the Reactive-Event handler. Sometimes the event can be completely handled at this layer. For instance, imagine that the communication with the primary compass fails and the physical sensor responsible for reading the values through the serial port sends an event. The Reactive Event Handler catches it and produces a reconfiguration of the Yaw logical sensor in order to abort sensor fusion, making use of the secondary compass only. In other cases, the event has to be handled at other layers, then the event is forwarded to the event handler in the layer above.

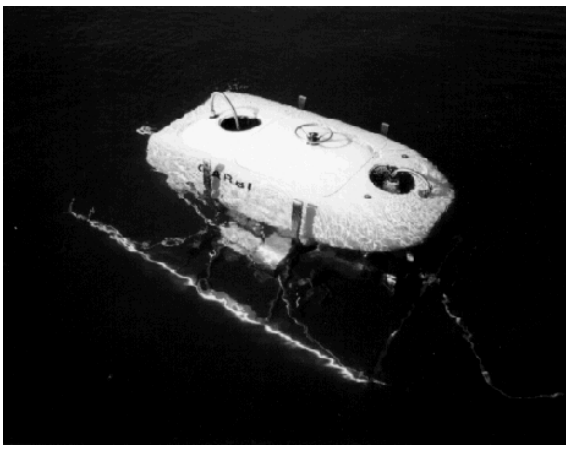
3.6 Discussion

What our architecture has in common with other behavioural architectures is its heterarchical organisation of the reactive layer and the ability to deal with dynamic environments. What it has in common with deliberative approaches is the existence of a world model (map) and a mission plan which is executed hierarchically giving predictability. As in other architectures, the low level controller and the sensing code are separated from the behaviours code. Then, behaviours are kept simple and different low level controllers can be tried without modifying the behaviours code. A key point of our architecture, is its distributed object oriented design. Each component is built as a dynamic object. A software object, which runs on its own thread of execution, is called a dynamic object. When an object needs data provided by another object it presents an interface to this object.

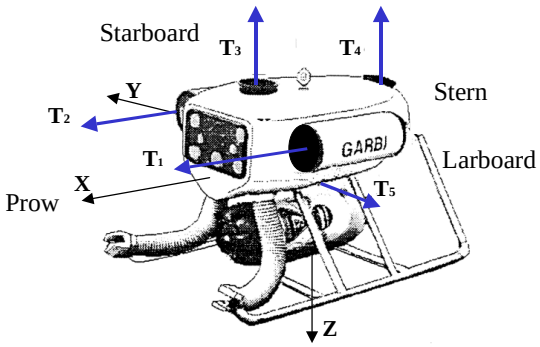
This interface provides access to the public methods of the target object. In consequence of this encapsulation, robot components are interchangeable with other robotic systems achieving software re-usage. As the system grows, some of the objects can be placed on another CPU if needed, instantiating the corresponding interface. Actually, message passing is used for object communication, but other mechanisms (i.e. sockets and shared memory) can be easily programmed since they are embedded within the object interface. On the other hand, since the interface of the behaviours is homogeneous (they always provide set-points to the low level controller), and since each behaviour is self-contained, it is simple to add new behaviours. These features allow our system to be extended and adapted to new needs.

4 GARBI robot

The OOCAA has been designed to carry out the control guidance of the GARBI robot. The vehicle [1] was built by the University of Girona together with the Politechnical University of Catalunya. GARBI was first conceived as a Remotely Operated Vehicle (ROV) for exploration in waters up to 200 meters in depth. The architecture here presented has been designed to transform this vehicle into an Autonomous Underwater Vehicle (AUV). GARBI, see Figure(2), was designed with the aim of building an underwater vehicle using low cost materials, such as fibre-glass and epoxy resins. To solve the problem of resistance to underwater pressure, the vehicle is servo-pressurised to the external pressure by using a compressed air bottle, like those used in scuba diving. Air consumption is required only in the vertical



Figure(2). GARBI robot.



Figure(3) GARBI schema

displacements during which the decompression valves release the required amount of air to maintain the internal pressure at the same level as the external one. The vehicle is equipped with two arms which allow it to perform some tasks of object manipulation through tele-operation.

The vehicle has five thrusters, see Figure(3), two for horizontal directions, two for vertical directions and one for the transverse direction. The vehicle also has several sensors: 2 compass, 2 pressure sensors and 2 water speed sensors. Dimensions are: length 1.3 m., height 0.9 m and width 0.7 m. Maximum speed is 3 knots and weight is 150 Kg.

5 Simulation

The experimental platform that is being used to design and validate the different aspects of the OOCAA architecture is a simulated environment. An AUV model and a virtual underwater environment compose the simulated environment. The simulation module provides a simple and fast method to carry out a first validation of the processes of the OOCAA. Once the new processes are tested, they will be implemented and verified with the experimentation tools.

5.1 The AUV Model

An AUV can be modelled by using two approaches [3]. The first one uses predictive techniques which build up the model from the basic physical laws governing the dynamic behaviour of the system. The second uses testing techniques which are based on parameter estimation from real tests. The model used for GARBI is based on predictive techniques. As described in the literature [7] and [2], the hydrodynamic equation of motion of an underwater vehicle with 6 DOF can be conveniently described as follows:

$$Bu = \left({}^{G}M_{RB} + {}^{G}M_{A}\right)\cdot {}^{G}\dot{V} + \left({}^{G}C_{RB}({}^{G}V) + C_{A}\left({}^{G}V\right)\right)\cdot {}^{G}V + \\ + D\left({}^{G}V\right){}^{G}V + {}^{G}G(O) \quad (1)$$

$${}^{G}G(O) = {}^{G}F_{B} + {}^{G}F_{W} \quad (2)$$

where,

B: thruster configuration matrix

$u=(\omega_1^2, \omega_2^2, \omega_3^2, \omega_4^2, \omega_5^2)^T$: control inputs vector

ω_i: angular speed of the propeller i

${}^{G}M_{RB}$: inertia matrix

${}^{G}M_{A}$: added-mass matrix

${}^{G}\dot{V} = \left({}^{G}a_{EG}, {}^{G}\alpha_{EG}\right)^T$: acceleration vector

${}^{G}V = \left({}^{G}v_{EG}, {}^{G}\omega_{EG}\right)^T$: velocity vector

$O = (\phi \quad \theta \quad \psi)^T$: Roll, Pith & Yaw angles

${}^{G}C_{RB}$: rigid-body coriolis

${}^{G}C_{A}$: added coriolis matrix

${}^{G}D$: damping matrix

^{G}G: gravity &buoyancy vector

The super-index denotes the coordinate system where vector components are expressed. {G} is a robot fixed coordinate system and {E} is an earth fixed coordinate system. For simulation purposes it's interesting to compute the evolution of the robot position and orientation as a function of the forces acting on the vehicle. This can be easily computed, arranging the terms of (eq.1) as follows:

$$K = \left(Bu - \left({}^{G}C_{RB}({}^{G}V) + C_{A}\left({}^{G}V\right)\right){}^{G}V - D\left({}^{G}V\right){}^{G}V - {}^{G}G(O)\right) \quad (3)$$

$${}^{G}\dot{V} = \left({}^{G}M_{RB} + {}^{G}M_{A}\right)^{-1}\cdot K \quad (4)$$

The velocity vector can be computed through integration:

$${}^{G}V = \int {}^{G}\dot{V}dt \quad (5)$$

and finally, the rate of change of the position and the orientation can be computed as follows:

$${}^{E}\begin{pmatrix} \dot{r}_G \\ \dot{O} \end{pmatrix} = \begin{pmatrix} {}^{E}R_G & 0_{3x3} \\ 0_{3x3} & T(O)^{-1} \end{pmatrix} {}^{G}\begin{pmatrix} v_{EG} \\ \omega_{EG} \end{pmatrix} \quad (6)$$

where,

${}^{E}R_G$ is the rotation matrix

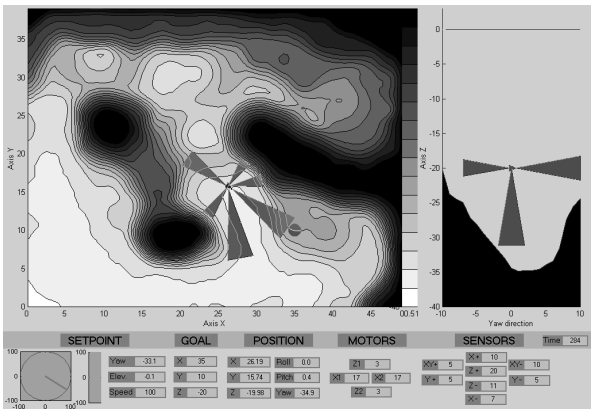
${}^{E}\dot{r}_G = (\dot{x} \quad \dot{y} \quad \dot{z})^T$ and

$$T(O)^{-1} = \begin{pmatrix} 1 & sin\phi\,\mathrm{tg}\,\theta & \cos\phi\,\mathrm{tg}\,\theta \\ 0 & \cos\phi & -sin\phi \\ 0 & \dfrac{sin\phi}{\cos\theta} & \dfrac{\cos\phi}{\cos\theta} \end{pmatrix}$$

A simulator was developed based on these equations. The parameters of the equations were obtained from the GARBI robot and the model was validated with real experiments.

5.2 The simulation environment

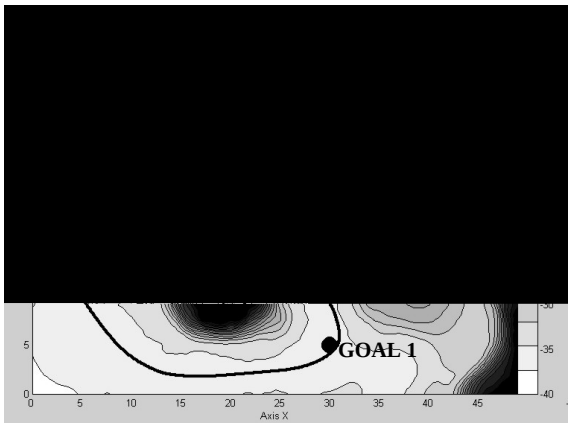
In order to visualise the 3-dimensional behaviour of the vehicle, a virtual environment has been implemented using Matlab-Simulink. Contour lines are introduced to evaluate the response of the control architectures. The vehicle is simulated with seven sonars that perceive the vicinity objects. Figure(4) shows an image of the underwater environment.



Figure(4). Underwater environment.

The simulation cycle starts with the OOCAA module providing the set-point which must be followed by the vehicle. Then, a fuzzy-logic base low-level controller transforms the setpoints into motor speeds. The model then returns a new position and orientation of the vehicle. Different sensors are simulated: a compass, 7 sonars and a mechanical water speed transducer. For each sensor a model and a uniform noise is used. Sonar sensors use the information provided by an underwater environment module that contains an environment built from a bitmap file. Objects and goal points can be added. During the simulation the underwater environment module shows the position and orientation of the vehicle as well as the distances detected by sonars. Sensory information is finally used by the control architecture to generate another set-point.

Figures(5,6) show a two and three dimensional result paths respectively. The mission consisted in navigating through 3 way-points avoiding obstacles. The simulation proved the validity of the reactive layer of the OOCAA.



Figure(5). Two-dimensional result.

Figure(6). Three-dimensional result.

6 Implementation of the OOCAA

The OOCAA is an object oriented control architecture. This means that every architecture component is built as an object. For example, the code functions related to the compass are encapsulated inside the Compass object. The utility of an object oriented code is not only the common advantages of this programming paradigm (a more clearer code, independence of the internal implementation of the different objects, etc.) but the possibility to build dynamic objects. Such objects have internal threads that are independent of the global execution thread of the main application. For example, in the compass sensor case, there is a thread running inside the object pooling the sensor, and another thread in charge of providing the reading to the rest of the architecture components. The main advantage of this kind of programming is the high concurrency degree, optimising the use of the CPU. However the principal disadvantage is the difficulty of coordination.

At this moment the communication among the different objects is done through message queues. Nevertheless, since the communication method is encapsulated within the interfaces, it is completely transparent to the objects. For this reason, it is very straight forward to introduce new communication mechanisms like TCP/IP sockets for instance.

All objects in this architecture belong to an strict hierarchy. For example, physical sensors inherit from a SensorSrv object, who inherits from a generic Sensor object. Interface objects inherit from a GenericInterface, an object which has methods to send and receive messages through message queues. There's still another level of inheritance: all objects that don't belong to the interfaces group inherit from UnitatComunicacional, an object with methods for handling the message passing communication.

An example of the source code is shown below. First, a monitor server and its interface are created. Then, they are linked together (by means of the enterSendQueue function). After this we create a new monitor to test the air pressure through the appropriate interface:

```
monitorServer = new MonitorServer();
interficie = new InterficieMonitorServer();
interficieMonitorServer->enterSendQueue(
                    monitorServer->getReceiveQueue());
monitorServer->Main();


idMonitor=interficieMonitorServer->afegeixMonitorBotella(
                  cuaRebrePeticionsEvents,
                  condicio, valor,
                  identificadorComportament);
```

This architecture is being implemented under the real time operative system VxWorks 5.2 from WindRiver. This O.S. gives a really robust base from which all the architecture can be safely built.

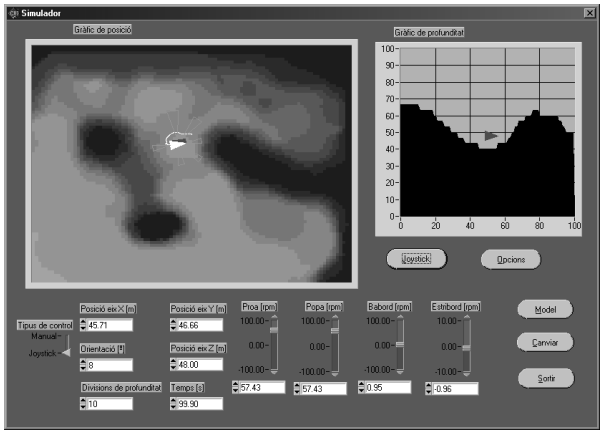
7 Experimentation tools

Once the architecture is implemented in the operative system of the robot, the experimentation phase starts. To carry out different experiments, a Human Machine Interface has been implemented using LabWindows/CVI. The HMI allows the interaction between the human user and the underwater robot, using a communication protocol to access to the

allowed control functions of the OOCAA architecture and to receive the sampled data and status of the robot. The HMI comprises three modes of operation:

7.1 Realistic simulation

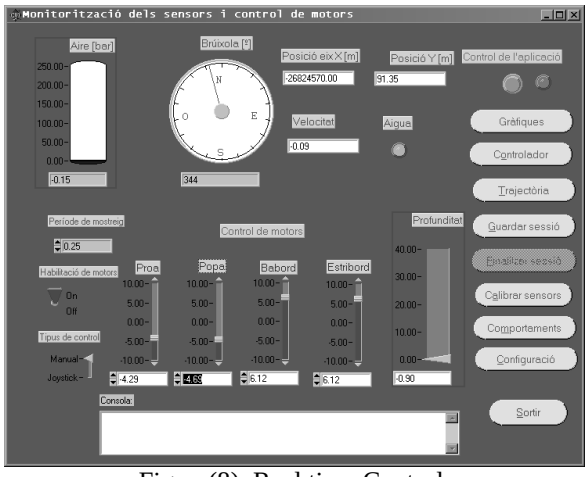
The realistic simulation mode is used to verify and test the OOCAA architecture in the lab, previously to real experiments. In this case, the system is compossed by 3 main components: (1) the HMI (2) the OOCAA and (3) the robot simulator. The HMI is exactly the same used for controlling the vehicle during real experiments. The OOCAA architecture is executed on the robot hardware but instead of using the physical sensor objects related to the robot sensors, it instantiates virtual sensors which emulate the real sensors by accessing the robot simulator. This is done transparently to the OOCAA code. Finally the robot simulator implements the robot model (as explained in the section 5.1) and a virtual environment which is used for emulating the sonar readings. The important fact is that the software implemented in the robot is used during the simulations, and this is useful to find implementation errors. A mission can be introduced with the HMI. Then a simulation screen Figure(7) shows the evolution of the robot in a virtual world generated from a bitmap file.



Figure(7). Realistic Simulation.

7.2 Real time control

The real time control mode is used in real experiments with GARBI vehicle. The principal functions are monitorisation of the vehicle, Figure(8), graphics representation and manual control of the vehicle with a joystick. When this mode of operation is active, the control architecture uses the real sensors of the robot and its positioning system. The Real time control mode is useful to test and tune the parameters of the vehicle. This is the last tool used in the design and implementation of the control architecture.



Figure(8). Real time Control.

7.3 Hybrid control

Finally an hybrid mode can be used. This mode uses the OOCAA control architecture with the real robot but with simulated sensors. The main goal is to use a virtual underwater environment with the real robot. This allows to test the OOCAA with different environments in an open underwater environment like a lake or a swimming pool. The sonar sensors are simulated to perceiving obstacles. This mode is very usefull to real test in a safety and fast way the behaviour of the OOCAA in front different situations.

8 Conclusions

An overall description of the OOCAA control architecture has been done. Also the reactive layer has been detailed with a description of the different tools used to test it (the simulation module and the experimentation tools) and to implement it. The main goal of the paper has been to show how to design and validate a control architecture for a physic agent. It has to be noted that this validation procedure before real tests is necessary with a robot like GARBI that needs a big quantity of human and material resources to realise every expedition.

Acknowledgements

The authors wish to acknowledge the support of the Spanish CICYT for the financial project MAR97-0925-C02-02.

References

[1] Amat, J., J. Batlle, A. Casals & J. Forest. GARBI: a low cost ROV, constrains and solutions. In: *6ème Seminaire IARP en robotique sous-marine* (1996), pp. 1-22. Toulon-La Seyne, France.

[2] Fossen, T.I. *Guidance and Control of Ocean Vehicles*. Chapter 2. John Willey & Sons. ISBN 0-471-94113-1. pp.6-55, 1995.

[3] Goheen, K. Techniques for URV Modelling. *Underwater Robotics Vehicles.* ed. J. Yuh. TSI Press, 1995.

[4] Gracanin, D., Valavanis, K.P., Tsouveloudis, N.C. & Matijasevic. Virtual-Environment-Based Navigation and Control of Underwater Vehicles. *IEEE Robotics & Automation Magazine*, (1999), pp. 53-62.

[5] Akkizidis, I.S. & Roberts, G.N. Fuzzy Modelling & Fuzzy-Neuro Motion Control of an Autonomous Underwater Robot. *The 5th International Workshop on advanced Motion Control*, (1998), pp 641-646. Coimbra ,Portugal.

[6] Ridao, P., Batlle, J., Amat, J. & Roberts, G. N. Recent Trends in Control Architectures for Autonomous Underwater Vehicles. *International Journal of Systems Science*, (1999), Vol.30, number 9, pp 1033-1056.

[7] Yuh, J. Modeling and Control of Underwater Robotic Vehicles. *IEEE Transactions on Systems, man, and Cybernetics*, (1990), Vol.20, nº 6, pp.1475-1483.

[8] Valavanis, P.K., Gracanin, D., Matijasevic, M., Kolluru R., & Demetriou, G.A. Control Architectures for Autonomous Underwater Vehicles. *IEEE Control Systems Magazine*, (1997), pp.48-64.

[9] White, K.A., Smith, S.M., Ganessan, K., Kronen, D., Rae, G.J.S. & Langenback, R.M. Performance Results of a Fuzzy Behavioural Altitude Flight Controller and Rendezvous and Docking of an Autonomous Underwater Vehicle with Fuzzy Control. *Proceedings of the Autonomous Underwater Vehicles Technology AUV'96*, (1996), pp.117-124.