Proceedings of the 2001 IEEE
International Conference on Robotics & Automation
Seoul, Korea · May 21-26, 2001

A Distributed Environment for Virtual and/or Real Experiments for Underwater Robots

P. Ridao[✣], J. Batlle[✣], J. Amat[✦], M.Carreras[✣]

[✣]Informatics and Applications Institute
Univeristy of Girona
Avda. Lluis Santaló s/n Girona CP. 17071 Spain.
{pere,jbatlle,marcc}@eia.udg.es

[✦]Automatic Control and Computer Eng. Dept.
Politechnical University of Catalunya
Pau Gargallo nº5 08028 Barcelona-Spain.
amat@esaii.upc.es

Abstract

This paper presents the Distributed Environment for Virtual and/or Real Experiments for Underwater Robots (DEVRE). This environment is composed of a set of processes running on a local area network composed of three sites: (1) the onboard AUV computer, (2) a Surface computer used as human machine interface (HMI) and (3) a computer used for simulating the vehicle dynamics and representing the virtual world. The HMI can be transparently linked to the real sensors and actuators dealing with a real mission. It can also be linked with virtual sensors and virtual actuators, dealing with a virtual mission. The aim of DEVRE is to assist engineers during the software development and testing in the lab prior to real experiments.

1. Introduction

The development of an autonomous vehicle is not a simple task. Besides the mechanical and electrical issues, the vehicle needs an intelligent software architecture responsible for driving the vehicle during the mission. This software must be able to deal with unstructured and probably unknown environments in real time. In order to build this kind of software, an intensive set of experiments in an exhaustive number of environments is necessary. Nevertheless, the vehicle for testing purposes, must often be shared among a number of researchers and engineers. Also, testing in real environment is expensive in both in resources and man hours. Hence, we are presented with the following problem: the number of real experiments must be reduced while at the same time intensive experimentation is being carried out. On the other hand, almost any engineer involved in the software development of an AUV has experienced frustration due to a simple mistake in the software. Sometimes, the mistake can be corrected at the place, but often it means aborting the mission and returning to the lab to patch the software. For these reasons, a graphical simulator implementing a virtual vehicle and a virtual world are desirable tools for research. In our laboratory we have designed an AUV software architecture in such a way that it can indistinctly control the real vehicle or the simulated one. This means that the vehicle software can be simulated in the lab before real experiments take place. Moreover intensive testing is easy and feasible.

Popularity of graphical simulators and virtual worlds is growing daily in the AUV community. The necessity of these kind of tools was clearly shown by D. P. Brutzman who implemented an integrated simulator for the NPS AUV [1]. Several authors extended this concept to include multiple vehicles working in either a virtual or hybrid virtual/real environment. At the University of Tokio, Kuroda *et al.* [2] developed a Multi-vehicle Graphical Simulator for the twin-Burger underwater robots. S. Chappell *et al.* (Autonomous Undersea Systems Institute) developed the CADCON concept [3] which employs a distributed multi-agent simulation and visualisation system. In a similar way S. K. Choi *et al.* at (University of Hawaii) presented a Distributed Virtual Environment Collaborative Simulator for Underwater Robots [4].

In this paper we present DEVRE, a Distributed Environment for Virtual and/or Real Experiments which has been developed in order to aid the software development for our underwater vehicle GARBI [5]. This paper is organized as follows. In section 2, we present an overview of the DEVRE system. Section 3 describes the real and the virtual vehicle and section 4 describes the virtual world. Section 5 presents each component of the system and section 6 reports the results before the conclusions in section 7.

2. Overview of DEVRE System

DEVRE is an integrated software platform (see Fig. 1) composed of three modules: (1) the Human Machine Interface (HMI), (2) the Mathematical Model of the Vehicle and the Virtual Environment (MMVVE), and (3) the Object Oriented Control Architecture for Autonomy (O^2CA^2).

The HMI operates as an interface with the human operator. It allows for monitoring the state of the vehicle as well as sending commands to it.

The MMVVE has two functions: (1) simulating the movement of the vehicle under water using the dynamical model and (2) providing a virtual representation of the underwater world. The utility of the world model is twofold. Firstly, it allows the visualization of the vehicle within the environment and secondly, it allows the simulation of sensors providing environment dependent information like sonar.

The O^2CA^2 is the high level control of the GARBI vehicle. It is a hybrid deliberative-reactive architecture in charge of controlling the vehicle during a mission.

All these programmes are written in different languages and run on different computers under different operative systems. HMI and MMVVE modules are programmed in LabWindows and executed on two individual PCs under Win32 OS. The O^2CA^2 is programmed in C++ and executed on the onboard x486 computer running VxWorks. All components are networked through a TCP/IP LAN (10 Mbs ethernet).

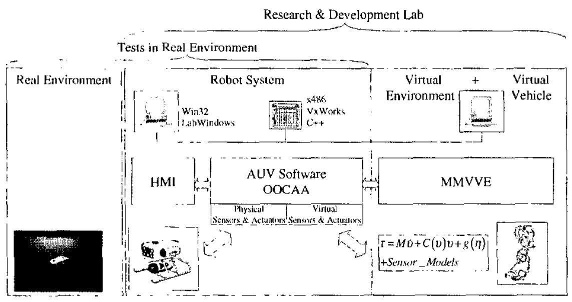

Fig. 1 DEVRE System

3. The Real and the Virtual Vehicles

In this section we present the real vehicle and its virtual partner V-GARBI.

3.1. GARBI: The Real Vehicle

GARBI was first conceived as a Remotely Operated Vehicle (ROV) for exploration in waters up to a depth of 200 meters. At this time, a control architecture is being implemented to transform the vehicle into an Autonomous Underwater Vehicle. GARBI (see Figure 2) was designed

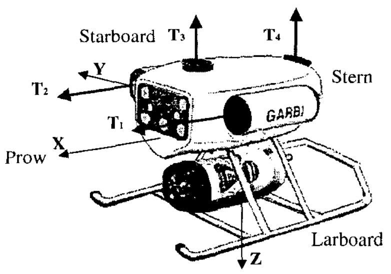

Fig. 2 Garbi Underwater Robotic Vehicle

with the aim of building an underwater vehicle using low cost materials such as fiber-glass and epoxy resins. To solve the problem of resistance to underwater pressure, the vehicle is servo-pressurized to the external pressure by using a compressed air bottle, like those used in scuba diving. Air consumption is required only in the vertical displacements. When the robot dives in the heave direction, a servo system introduces air into the hull in order to increase the internal pressure until it reaches the external pressure. On the other hand, when the robot goes to the surface decompression valves release the required amount of air to maintain the internal pressure at the same level as the external one. The vehicle allows for the incorporation of two mechanical arms which would perform some tasks of object manipulation through tele-operation.

The vehicle has 4 thrusters, see Figure 2, two for horizontal movements (axis x) and two for vertical movements (axis z). Additionally it is possible to add another thruster in the transverse direction (axis y). Due to the distribution of the weights, the vehicle passively stable in Roll and Pitch. 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 the weight is 150 Kg aprox.

3.2. V-GARBI: The Virtual Vehicle

The virtual vehicle is a software module which behaves as the real vehicle does. It is composed of the dynamic model of the vehicle and the onboard sensors model.

3.2.1. Dynamical Model of the Vehicle

As described in the literature [6], the hydrodynamic equation of motion of an underwater vehicle with 6 DOF can be conveniently described as follows:

$$Bu = \left({}^{G}M_{RB} + {}^{G}M_{A}\right)\cdot {}^{G}\dot{\upsilon} + \left({}^{G}C_{RB}({}^{G}\upsilon) + C_{A}\left({}^{G}\upsilon\right)\right)\cdot {}^{G}\upsilon$$
$$+D\left({}^{G}\upsilon\right){}^{G}\upsilon + {}^{G}G(O) \qquad \text{(eq. 1)}$$

where ,

B is the thruster configuration matrix
$u=(\omega_1^2, \omega_2^2, \omega_3^2, \omega_4^2)^T$ is the control inputs vector
ω_i is the angular speed of the propeller i
${}^{G}M_{RB}$ is the inertia matrix
${}^{G}M_{A}$ is the added-mass matrix
${}^{G}\dot{\upsilon} = \left({}^{G}a_{EG}, {}^{G}\alpha_{EG}\right)^T$ is the acceleration vector
${}^{G}\upsilon = \left({}^{G}v_{EG}, {}^{G}\omega_{EG}\right)^T$ is the velocity vector
$O = (\phi \quad \theta \quad \psi)^T$ are the Roll, Pith & Yaw angles
${}^{G}C_{RB}$ is the rigid-body coriolis matrix
${}^{G}C_{A}$ is the added coriolis matrix
${}^{G}D$ is the damping matrix
${}^{G}G$ is 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 we are interested in computing the evolution of the robot position and orientation as a function of forces acting on the vehicle. This can be computed easily arranging the terms of (eq.1) as follows:

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

$${}^{G}\dot{\upsilon} = \left({}^{G}M_{RB} + {}^{G}M_{A}\right)^{-1} \cdot K \qquad \text{(eq. 2)}$$

The velocity vector can be computed through integration:

$${}^{G}\upsilon = \int {}^{G}\dot{\upsilon}\, dt \qquad \text{(eq.3)}$$

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

$${}^{E}\begin{pmatrix} \dot{r}_G \\ \dot{O} \end{pmatrix} = \begin{pmatrix} {}^{E}R_G & 0_{3x3} \\ 0_{3x3} & T(O)^{-1} \end{pmatrix} {}^{G}\begin{pmatrix} v_{EG} \\ \omega_{EG} \end{pmatrix} \qquad \text{(eq.4)}$$

where:

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

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

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

so finally the robot position and orientation can be computed as follows:

$${}^{E}\begin{pmatrix} r_G \\ O \end{pmatrix} = \int {}^{E}\begin{pmatrix} \dot{r}_G \\ \dot{O} \end{pmatrix} dt \qquad \text{(eq.5)}$$

3.2.2. Sensor Models

Concerning the sensors, we can consider three different types. First, the sensor related to the vehicle state (speed, acceleration, depth, position, etc...). Second, the sensors responsible for sensing a relation between the robot and its surrounding environment (sonar, computer vision). Finally, the mission payload sensors. When dealing with control architectures we are only interested in the first two types.

State Sensors

At this point in time, 5 virtual state sensors have been implemented for our vehicle:

- 2 compasses
- 2 water speed sensor
- 1 Depth sensor
- 1 Pressure sensor for the air compressed bottle

Several experiments where conducted for each of these sensors allowing us to compute the mean and the standard deviation of the absolute error (see table 1). Hence, each virtual sensor was implemented using the corresponding output of the dynamic vehicle model adding a gaussian error signal with the corresponding mean and variance.

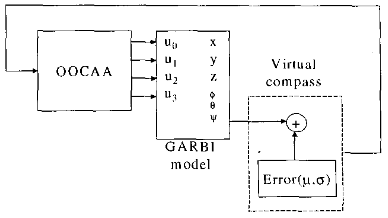

Fig. 3 Virtual compass

Figure 3 shows how the virtual compass is implemented. The virtual sensor for depth and speed follow the same approach. The last one has a dead zone so, it only provides response for speeds greater than 0.05 [m/s]. The virtual sensor related to the pressure sensor used for monitoring the air consumption is modeled as follows.

Let

- p_{k-1} is the external pressure at instant t_{k-1}
- b_{k-1} the pressure in the air bottle at instant t_{k-1}
- V_h the volume of the hull
- V_b the volume of the air bottle

the pressure of the air bottle at instant t_k, can be computed as:

$$\Delta p = p_k - p_{k-1}$$

$$\Delta b = \begin{cases} \dfrac{\Delta p \cdot V_h}{V_b} & \text{if } \Delta p > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$b_k = b_{k-1} - \Delta b$$

where the temperature has assumed to be constant.

Since b_k is the real pressure, we can obtain a realistic measurement by adding an error source similar to the one stated in table 1.

Sensor	μ	σ
Compass 1	2.9°	2.7133°
Compass 2	-2.14°	4.14°
Speedometer	-0.02357 [m/s]	0.02765 [m/s]
Depth Sensor	0.06767 [m]	0.03336 [m]
Pressure Sensor	0.006767 [bars]	0.003336 [bars]
sonar	0	0.057 [m]

Table 1 Error parameters of the robot sensors

Environmental Sensors

Concerning the environmental sensors, only the sonar has been implemented. Sonar can be simulated at different levels. From the physical point of view, we can use the sonar equation. Each transducer can be considered as a sound source, and taking into account the transmission losses (spreading and attenuation), we can compute the sound level in the position where the beam impacts with the surrounding objects. Each of these points of impact can then be considered a new sound source and we can compute the way back in order to compute the sound level of the reading at the receptor. When this sound level is higher than a threshold, we consider its corresponding time of flight in order to compute the range. With this kind of simulation, important effects like the multi-path can be taken into account. The drawback is the computation time. Since we are interested in simulating sonar behavior in real-time, such a method is unaffordable.

The second method is graphical. Each sonar beam is simulated as a cone with β degrees of aperture. While the robot moves in the virtual environment, the points belonging to this cone are explored in order to see if they impact with objects in the vehicle surroundings. In this case, the axial distance plus a gaussian error are considered as the measurement given by the transducer. Moreover, in order to consider the multi-path problem, a second error source is added periodically. This second source has a high variance and a mean greater than zero producing glitches in the sonar ranges.

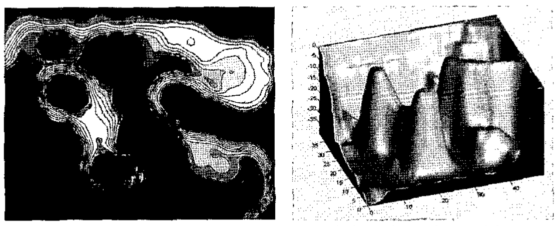
Fig. 4 2D and 3D rerpresentation of the virtual world

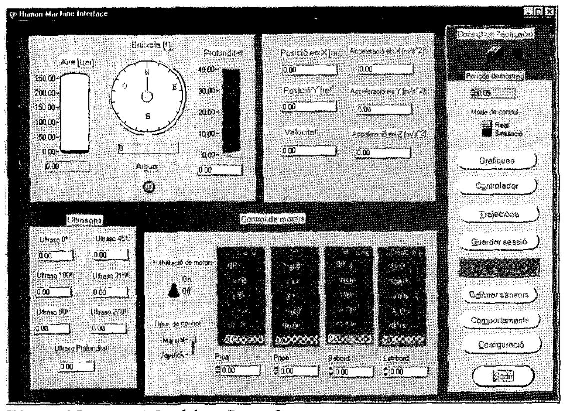
Fig. 5 Human Machine Interface

4. The Virtual World

In order to keep the software simple, we have chosen a world model representation based on contour lines (Fig.4). Although this representation does not allow representation of caves, it is general enough to represent most common underwater environments. The world is considered a 2D grid of arbitrary dimension, containing the height of the corresponding area. These kinds of worlds can be easily generated using conventional drawing tools.

5. DEVRE

In this section, we describe the components of the DEVRE and their relationship.

5.1. HMI

The HMI is the interface between the O^2CA^2 control architecture (described below) and the operator. Its main functions are the following:

- Monitoring the state of the sensors during the mission: their values are shown in a control panel (see Fig.5).
- Logging sensor readings: sensor data is saved in a file for post-mission study purposes.
- Sending commands to the control architecture (such us enabling or disabling robot behaviors)
- Sensor calibration (such us the definition of the zero depth pressure).
- Low-level teleoperation of the vehicle (using a joystick and sending commands to the low level controller).
- Switching between real and virtual environments.

In order to provide this function, the HMI uses a custom defined protocol which provides message frames for each one of these functions. It also provides frames for communicating the O^2CA^2 with the MMVVE, like sending the angular speed of the thrusters to the MMVVE or receiving the position, velocity and acceleration from the MMVVE. This protocol runs on a 10 Mbits Ethernet link.

5.2. Overall Description Of The O^2CA^2 Architecture

The control software of GARBI is arranged in three layers 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 minimize a cost function. The planning process takes place when the user specifies a new task or when a 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 the 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 beginning 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 behaviors.

The real-time control of the vehicle is in charge of the reactive layer which provides three reactive mechanisms: (1) behaviors, (2) monitors and (3) timers. Each behavior has its own goal and can be executed independently or concurrently with the others. There are safety behaviors such as obstacle avoidance and navigation behaviors like going to one point or circumnavigation. Behaviors read input values from the sensor subsystem and use fuzzy IF-THEN rules to compute the new set-point to the low level controller. A cooperative coordinator is responsible for merging the set-points provided by the active behaviors into a unique set-point which is then sent to the low-level controller. Monitors are used for situation recognition (the robot has reached the programmed position, or depth, etc...) and event handlers provide pre-programmed reactions to these situations. Timers are used for computing deadlines, allowing the event handler to execute when the robot is unable to reach its goal within a deadline. See [7] for a detailed description.

5.2.1. Sensor Subsystem

The sensor subsystem is arranged hierarchically. For each robot sensor there is a physical sensor object which acts as an interface to the real sensor. 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 behaviors. For instance, the logical sensor, which provides the angular speed (yaw derivative), makes use of the compass physical sensor while the logical sensor providing the x position makes use of the lineal speed physical sensor and the compass physical sensor. When needed, sensor fusion takes place at the logical sensor level. For instance, since our robot uses two compasses it has two physical sensors, and one logical sensor which merges the values of the two previous ones by means of a Kalman Filter.

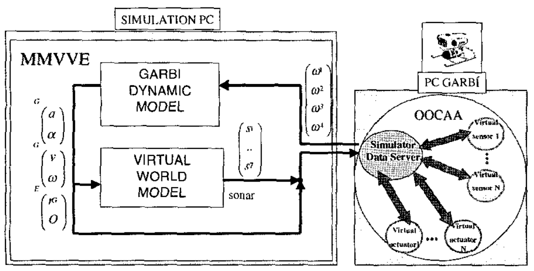
SIMULATION PC
MMVVE
GARBI DYNAMIC MODEL
VIRTUAL WORLD MODEL
sonar
$^{G}\begin{pmatrix} a \\ \alpha \end{pmatrix}$
$^{G}\begin{pmatrix} v \\ \omega \end{pmatrix}$
$^{E}\begin{pmatrix} p^{G} \\ O \end{pmatrix}$
$\begin{pmatrix} \omega^1 \\ \omega^2 \\ \omega^3 \\ \omega^4 \end{pmatrix}$
$\begin{pmatrix} s_1 \\ .. \\ s_7 \end{pmatrix}$
PC GARBÍ
OOCAA
Simulator Data Server
Virtual sensor 1
Virtual sensor N
Virtual actuator1
Virtual actuator N

Fig. 6 Software Architecture of the MMVVE and its relation with the O^2CA^2

5.2.2. Actuator Subsystem

Each actuator is represented within the software architecture by an actuator object. This object acts as an interface, receiving the actuator set point and sending it to the hardware.

5.3. MMVVE

As stated above, the MMVVE is composed of the dynamic model of the vehicle and the model of the virtual world (Fig.6). The dynamic model block sequentially computes from eq.2 to eq.5. The outputs of this block are the acceleration, velocity and position vectors. The position vector is used by the virtual world module to compute the range measured by the sonar (vector s). The four vectors are periodically sent to the Simulator Data Server object. This object acts as a mediator between the simulator and the O^2CA^2. The virtual sensors which simulate the corresponding physical sensors access the Simulator Data Server and then add an error signal as defined in Table 1. On the other hand, the set points sent to the virtual actuators ($\omega_1,\ldots, \omega_4$) by the low-level controller are sent to the Simulator Data Server and then forwarded to the MMVVE. Note that the logical sensor code and the code corresponding to the low-level controller is the same when we run both, real and virtual experiments. Hence, the interface between the MMVVE and the O^2CA^2 is contained in the virtual sensor and virtual actuators plus the Simulator Data Server. This is a powerful feature of DEVRE, since the O^2CA^2 code is always the same regardless of experiments being carried out in a real or virtual environment.

5.4. Modes of Function

DEVRE has three modes of function although only two are ready to work at this moment:

- **Real Mode:** In this mode the logical sensors are linked to the physical sensor objects and the low-level controller is linked to the real actuators. Hence, the control commands carried out by the O^2CA^2 are executed by the robot. This is the mode used when real experiments are carried out in a real environment.
- **Virtual Mode:** In this mode the logical sensors are linked to the virtual sensor objects and the low-level controller is linked to the virtual actuators. Hence, the control commands carried out by the O^2CA^2 are executed by the MMVVE. This is the mode used in the lab while developing.
- **Hybrid Mode:** This mode will be ready in the near future. When using this mode some of the sensors are real while others are virtual. The low-level controller is attached to the real actuators, so the robot moves through a real environment. Nevertheless, since some of the sensors are virtual, it can sense a virtual synthetic environment. From our point of view, this mode is very interesting. It allows experimenting in virtual environments, while

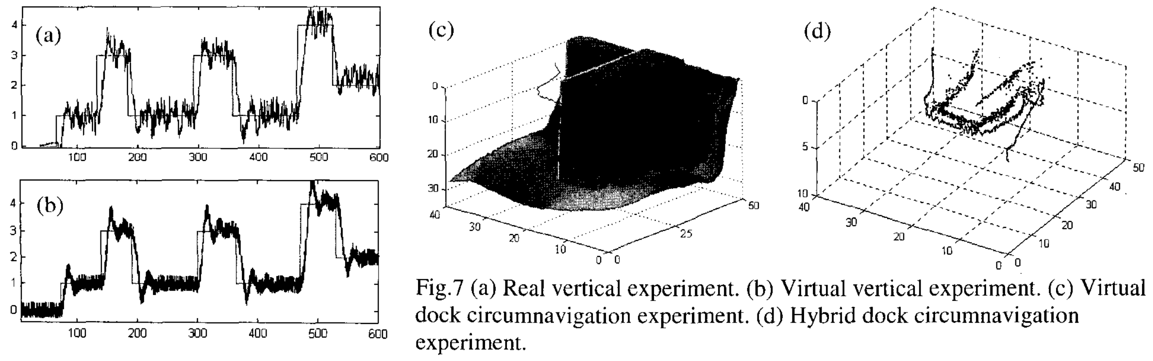
Fig.7 (a) Real vertical experiment. (b) Virtual vertical experiment. (c) Virtual dock circumnavigation experiment. (d) Hybrid dock circumnavigation experiment.

observing the response of the real robot to the virtual environment sensed by the virtual sensors.
The selection of the mode of function is done using a simple switch of the HMI.

6. Results

In order to test the capabilities of the DEVRE system, two experiments were undertaken. The first experiment consisted in programming the robot to alternate among depths $z_d=1$, $z_d=3$ and $z_d=4$. A simple PI controller was used (P=200, I=1). First, the experiment was executed in the lab on the virtual simulator (Fig.7b), and then it was reproduced in the lake (Fig.7a). Both Figures show the readings of the virtual and real depth sensor before filtering.

The second one was an hybrid experiment. The robot was considered to be located at position (0,0,0) and it was programmed to track the trajectory [(35,10,0), (35,10,3), (35,33,3), (35,33,0)] while circumnavigating a dock located in the virtual world. The experiment was first executed in virtual mode obtaining the result shown in fig. 7c, and then reproduced in the lake but sensing the same virtual world (sonar was simulated). Fig.7d shows the local map built during the experiment and the tracked trajectory as sensed by the vehicle.

The execution of the experiments in virtual mode before executing the real experiments proved to be very useful, allowing us to detect and solve development problems in the lab before going to the trial environment.

7. Conclusions and further work

DEVRE is a distributed system which integrates graphical simulation and real execution of a mission by an underwater robot. It is composed of three main components: HMI (operator interface), MMVVE (the virtual vehicle + the virtual world) and the O^2CA^2 (thehigh level control of the robot). The O^2CA^2 can work equally well with the real and/or the virtual vehicle. This feature has been easily achieved as a result of the distributed object oriented structure of the O^2CA^2. Although the system works with a 3D virtual world, at this moment it only provides two two-dimensional views (XY plane, and Z). This is probably the weak point of DEVRE. In the future, the MMVVE will be partitioned in two applications: (1) the dynamic model of the vehicle and (2) a 3D representation of the virtual world using a graphical package like OPENGL.

Acknowledgements

The authors wish to acknowledge the support of the Spanish government for the financial project MAR99-1062-C03-02 (CICYT).

References

[1] D.P. Brutzman, Y. Kanayama & M.J. Zyda, "Integrated Simulation for Rapid Development of Autonomous Underwater Vehicles," Proceedings of the IEEE Oceanic Engineering Society Autonomous Underwater Vehicle 92 Conference, Jun. 1992.
[2] Y. Kuroda, K. Aramaki, T. Fujii & T. Ura, "A Hybrid Environment for the Development of Underwater Mechatronic Systems," Proceedings of the 1995 IEEE 21st International Conference on Industrial Electronics, Control, and Instrumentation, Nov. 1995.
[3] S.G. Chappell, R.J. Komerska, L. Peng &Y. Lu, "Cooperative AUV Development Concept (CADCON) – An Environment for High-Level Multiple AUV Simulation," Proceedings of the 11Th International Symposium on Unmanned Untethered Submersible Technology, Aug.1999.
[4] S.K.Choi, S.A. Menor and J.Yuh, "Distributed Virtual Environment Collaborative Simulator for Undewater Robots", IEEE/RSJ Int. Conf. on Robots and Systems IROS'2000, November 2000.
[5] Amat, J. Batlle, J. Montferrer, A. Salvi, J. and Ridao, P., 1998, "Capabilities of GARBI - A Low cost underwater Vehicle". IEEE/RSJ Int. Conf. on Robots and Systems IROS'98, WS1- Workshop on Recent Trends in Intelligent Underwater Robotics.
[6] Fossen, T.I, "Guidance and Control of Ocean Vehicles", John Willey & Sons, 1995.
[7] P. Ridao, M.Carreres, J.Batlle, J. Amat, "O^2CA^2: A New Hybrid Control Architecture for A Low Cost AUV", to be published in the Proc. of the Control Application in Marine Systems, Scotland, 2001.