REMOTE TRAINING IN AUV CONTROL USING HIL SIMULATORS

P. Ridao, E. Hernandez, N.Palomeras and M. Carreras

Institute of Informatics and Applications
(University of Girona, Spain)

Abstract: This paper proposes a set of software tools aimed for promoting joint-research among partner institutions in the context of international projects about AUV control. The paper presents two basic tools developed in our lab: *NEPTUNE* and the Object Oriented Control Architecture for Autonomy (O^2CA^2). The first is a hardware-in-the-loop simulator with facilities for remote experimentation through the Internet. The second, O^2CA^2, is a distributed object oriented software architecture with facilities for real-time execution. Using both together, it is possible to carry out a real-time control experiment with a robot located in a remote lab .

Keywords: remote control, virtual reality, simulators.

1. INTRODUCTION

When dealing with underwater applications, an intensive set of experiments in an exhaustive number of scenarios is necessary for assuring reliability. Nevertheless, for testing purposes, the vehicle must often be shared among a number of researchers and engineers. Also, testing in real environments is expensive in both, resources and man hours. Hence, we are presented with the following problem: the number of experiments must be reduced while at the same time intensive experimentation is necessary. Moreover the access to the actual vehicle for research experimentation must be carefully scheduled, experiments should be done one after the other and hence, the turn-around time of the results is very long. Unmanned Underwater Vehicles (UUV) simulators can be of great help to overcome such problems. Over the past few years, different types of simulators have been proposed in the literature:

- **Offline simulators:** They are often built as toolboxes for MATLAB/Simulink.
- **Online simulators:** Their principal feature is that the simulation time matches the real-time. Hence, one minute of simulation time is needed to simulate a one minute experiment. Commonly they are executed on hardware compatible with the hardware to be used in the real experiment.
  - **Hardware in the loop simulators (HIL):** Online simulators where the software is executed in the actual robot hardware but controlling a real-time simulation.
    - **Hybrid Simulators:** HIL simulators where some of the sensor/actuators are virtual while others are real.

To simulate the robot movement, some simulators use the robot kinematics equations, but most of them, rely on the hydrodynamics equation of motion. They also provide models to simulate the internal sensors (navigation sensors), and world models to allow for the simulation of the external sensors (sonar and computer vision). For a detailed survey about UUV simulators the reader is referred to (Batlle, *et al*., 2005) and the references therein.

On the other hand, it is also very common to share experimental robots (test-beds) among several organizations which are partners in the same project. Since normally only few prototypes are available for experimentation, researchers use to spend long periods of time in their partner organizations to test their developments with the

corresponding increase of cost. This sort of simulators can also represent a step ahead in the research cooperation between institutions. They can be broadly available to all the partners, allowing them to locally experiment in their home countries until their development is ready for integration. Then, after running exhaustive online simulations, the developed system can be interfaced through the Internet with the real robot in order to execute remote experiments in real-time. In this case, the same graphical environment used for the simulation can be used as an augmented reality system to monitor the evolution of the remote experiment. In this context, the researchers of the Instituto Superior Tecnico (IST) in Lisbon interconnected their mission control system called CORAL to the ROMEO control architecture developed by the researchers at the CNR-IAN (Bruzzone, *et al*., 1999), making it possible to control, from Lisbon, the mission of a ROV working in Genoa. Before carrying out the real experiments, they used a Hardware In the Loop (HIL) graphical simulator to test their approach (Bruzzone, *et al*., 2001). Other teams, like the Autonomous System Laboratory in Hawaii and the researchers of the University of Tokio, have interconnected their respective graphical simulators DEVECS (Choi, *et al*., 2000) and MVS (Kuroda, *et al*., 1996) and have been able to carry out a multiple vehicle mission in a virtual environment. They used two real robots: (1) ODIN located in Hawaii and (2) the Twin-Burger placed in a water tank at the University of Tokyo. In this case, both vehicles were able to interact virtually with each other and the researchers on both sides were able to observe the multi-vehicle mission in a virtual representation. This is also the case with the *NEPTUNE* HIL simulator and the Object Oriented Control Architecture for Autonomy (O^2CA^2), developed at the University of Girona. Our system provides facilities to carry out remote experiments across the Internet allowing sharing our prototypes with partner laboratories.

In this paper we first present the *NEPTUNE* simulator and the O^2CA^2 software architecture (section 2), and then we describe a possible scenario for a remote experiment (section 3). Next section reports the results of a remote experiment involving two distant sites and section 5 presents the further work before concluding in section 6.

2. PROPOSED TOOLS FOR NETWORKED COLLABORATIVE RESEARCH

Using *NEPTUNE* and O^2CA^2, the following methodology can be used to carry out a remote experiment. First, a researcher placed at the local site develops his control algorithm within the O^2CA^2 software architecture on a local computer. This algorithm is tested with a local copy of *NEPTUNE*. The online simulation of the controller allows the researcher to test that his code is logically and temporally correct. Then the controller is uploaded to the control computer of the robot in the remote place. After, a experiment is carried out using the robot operating in the remote site with the help of *NEPTUNE*, located at the local site, which reproduces a virtual view of the experiment using the telemetry transmitted through the Internet. The logging mechanism of the O^2CA^2 is used to generate a file of results that can be downloaded and evaluated. In the following sections, *NEPTUNE* and O^2CA^2 are briefly described.

1.1 NEPTUNE HIL Simulator.

NEPTUNE is a real-time graphical simulator with capabilities for on-line, hardware in the loop and hybrid simulation (Fig.1). Externally, the virtual world is based on two components: (1) a VRML file containing the topography of the scene and (2) a set of objects also defined in VRML. Internally, the topography of the scene is converted into a bathymetry grid and the objects are considered spheres of a particular radius of action. This model, together with a conic beam sonar model allows a very simple and fast geometric method for obstacle and/or collision detection. Of course, there is a basic assumption which considers the bottom surface as convex.

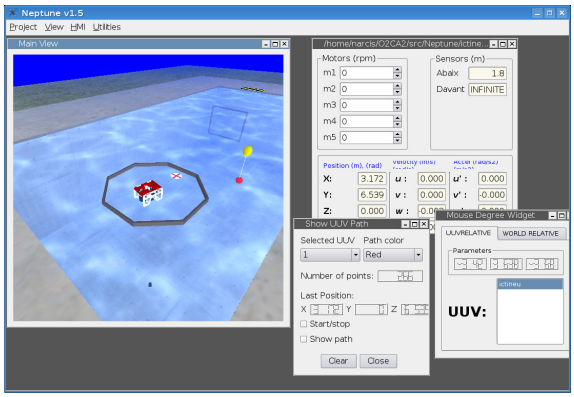

Fig. 1. View of the *NEPTUNE* simulator.

NEPTUNE is a multi-vehicle Simulator. Hence, more than one robot can be simultaneously simulated. Each robot is defined through three basic files. The first, is a VRML file containing the robot geometry. The second, is a file which contains the robot and thrusters hydrodynamics coefficients. For simulation, the hydrodynamic model (Fossen, 2002) is used. Thrusters are simulated using the affine model. Finally, the third file contains the file names of the previous two files plus a definition of the sensors included in the robot (number of sonar beams, position and attitude within the robot frame, number of cameras, etc ...). A custom developed language has been used for the definition of the last two files. Then, it is very easy to adapt *NEPTUNE* to simulate new robots. At this moment, simulated sensors include: (1) a range detection sonar, (2) an imaging sonar (3) computer vision and (4) navigation sensors (DVL, DGPS, USBL). It is also possible to introduce

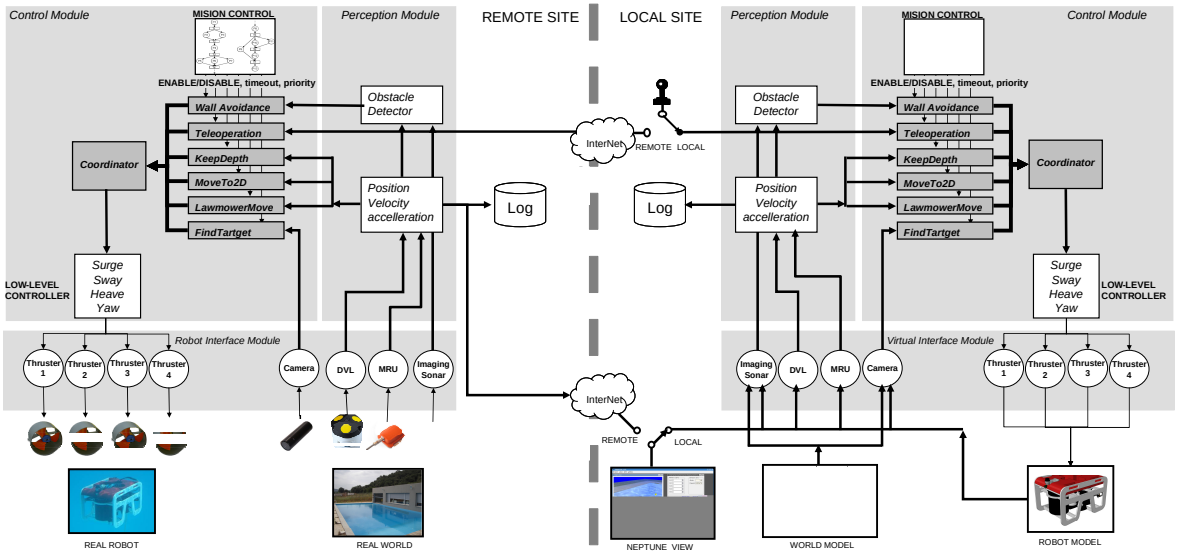

Fig. 2. Scheme of a Remote Experiment.

irrotational ocean currents modeled as first order Gaussian-Markov processes.

In order to allow a real time performance, the system has been build as a distributed application including several processes: (1) the *NEPTUNE* main program, (2) one robot dynamics process for each simulated robot, (3) a name server. All the components are built as CORBA-RT distributed objects. Refer to (Ridao, *et al.*, 2004) for a detailed description of *NEPTUNE*.

1.2 The O^2CA^2 software Architecture.

O^2CA^2 is a distributed object oriented software architecture organized in three modules (Fig.2): (1) an Interface module, (2) a Perception module and (3) a Control module. In the following subsections these modules are briefly introduced.

Robot interface module
This module encapsulates all the software objects that dialog with the hardware. There are basically to types of objects: (1) sensor objects responsible for reading data from sensors and (2) actuator objects responsible for sending commands to the actuators. Common sensor objects include a DVL, an imaging sonar, a MRU, and cameras within others. Actuator objects correspond to the robot thrusters. All the sensor and actuator objects have virtual counterpart, with the same interface, that instead of communicating with the robot, communicate with a real-time simulation.

Perception module.
This module contains two basic components: (1) the *Navigator* and (2) the *Obstacle Detector*. he *Navigator* object has the goal of estimating the position of the robot. To accomplish this task, there exists an interface called NavigationSensor from which all the localization sensors (DVL, MRU, depth sensor) inherit. This interface provides to all these sensors a set of methods to return the position, velocity and acceleration in the six DOF together with an stimation of the quality of these measurements. The *Navigator* can be dynamically connected to any *NavigationSensor* and, using the quality factor fuses the data to obtain an accurate estimate of the position, velocity and acceleration. The Control module uses the navigation data provided by the *Navigator* keeping the behaviours independent of the physical sensors being used for the localization. The *Obstacle Detector* uses the same philosophy to provide the obstacles position in the world fixed frame. To achieve its goal, it makes use of the acoustic images provided by the imaging sonar and the robot altitude provided by the DVL.

Control module.
The control module receives sensor inputs from the perception module and sends command outputs to the *Actuators* residing in the *Robot Interface Module.* A behaviour (Arkin, 1998) is function that maps the sensor input space (stimuli) into a velocity setpoint (behaviour response) for the robot low level controller. The behaviour response is chosen in a way that drives the robot towards its corresponding goal. In this way, the goal corresponding to the *keepDepth* behaviour is considered to be achieved when the robot is within an interval around the desired depth. A task is a set of behaviours that are enabled together to achieve a more complex goal. For example, *KeepDetph* and *MoveTo2D* can work together to allow for planar navigation. The control module follows the principles of the hybrid control architecture organized in three layers: (1) Mission Level, (2) Task Level and (3) Vehicle Level. The vehicle level is composed by a MIMO PID velocity controller for each DOF. The Task level is a conventional behavioural (Arkin, 1998) layer including several behaviours running in parallel and a coordinator module which fuses all the behaviour responses into a unique coordinated response to be used as velocity set point for the low level controller. Finally, the upper layer (mission level) is responsible

This projects ties the virtual world (”ICTINEU.MON”) and all the AUVs that will be involved in the simulation, in our case one (”ICTINEU.UUV”). Finally, *NEPTUNE* starts the simulation and the O^2CA^2 control architecture can be tied to the simulator in order to run an online simulation. This phase corresponds to the operation of the modules represented in the right part of Fig. 2.

3.4. Remote Experiment

Once the online simulation has been executed satisfactorily, a remote experiment involving the real robot has to be executed (corresponding to the left part of Fig.2). At the time of writing this paper, the ICTINEU *Navigator* (Ribas, *et.al.* 2006) does not work in real-time yet. For this reason, in the experiments described hereafter, the real robot was replaced by a real time simulation. Hence a second instance of *NEPTUNE* runs in the remote place playing the role of the real robot.

To run the remote experiment, first the software object of the *FindTarget* behaviour is uploaded to the CVS and it is compiled within the robot computer. *NEPTUNE* and the software object corresponding to the joystick are launched at the local site. Taking profit of the name server, both component locate and connect to the rest of the O^2CA^2 software architecture which runs at the remote site. The experiment is then started and the remote *Navigator* object is read periodically (~25 Hz.) to update the virtual view shown in the local *NEPTUNE*. The Joystick is used to navigate the robot towards the desired starting point and signals the start of the experiment. It is worth noting that during the experiment the robot works autonomously, and only the joystick commands and the navigation data (position) are transferred through the network. The logging mechanisms of O^2CA^2 and *NEPTUNE* are enabled, recording the local and remote data of the whole experiment. In order to be able to use a shared base of time for comparison, all the involved computers are synchronized using the Network Time Protocol (NTP) protocol. At the end, the file of results gathered during the experiment is downloaded and evaluated by the researcher.

4. RESULTS

In order to validate the proposed approach, we run several remote experiments, all of them corresponding to the execution of the SAUC-E mission. In all cases the local site was located in the research laboratory of the Vision Systems and Robotics (SVR) research team of the University of the Balearic Islands (UIB) in Mallorca while the remote site was located at the Underwater Robotics Lab of the University of Girona (approx. 300 Km far). In all the experiments all the robot software, except the joystick and the local *NEPTUNE* simulator, were located remotely. Even though that the joystick software we used was exactly the same we commonly use for local experiments, it was quite easy to drive the robot to the starting position. Once the experiment started, it was possible to observe its evolution using different virtual views provided by the *NEPTUNE* application in real time. During the experiments it was observed that the movement shown by the robot in the local screen was not as smooth as the one shown in the remote screen. This effect can be easily attributed to the variable latency experimented by the telemetry transmitted over the network.

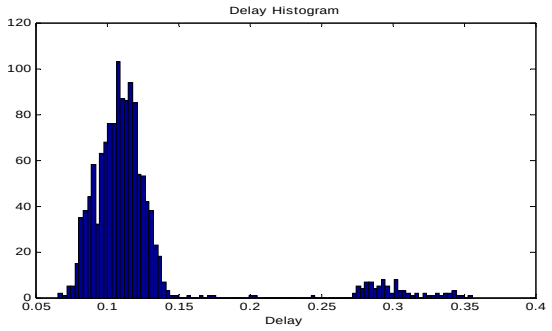

Fig. 4. Histogram of the delays experimented by the telemetry data.

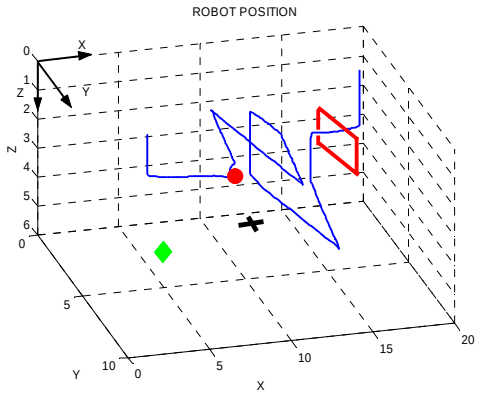

Fig.5. Robot Trajectory during the remote experiment

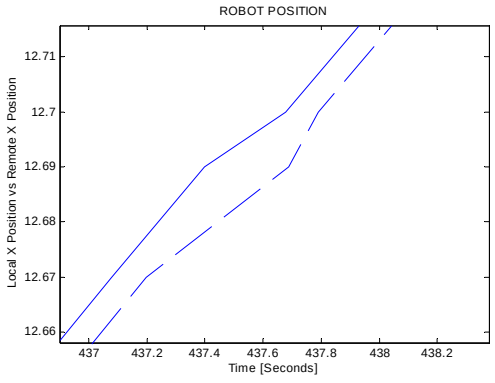

Fig. 6. Detail of the Local (dashed line) vs. Remote (Solid line) X Position during the experiment.

The Fig. 4 shows, for a representative experiment, the histogram of the delays suffered by the telemetry packets. An average latency of 0.1215 [seconds] with a standard deviation of 0.0502 [seconds] was observed during the experiment with a maximum and a minimum delay of 0.3560 and 0.0660 respectively. The histogram shows that although most of the

delays are very close to mean, there still a few number of them distributed near the value of 0.3 [seconds], which are the responsible for the robot flickering in the local screen. Nevertheless, even in the worst case, the latency was small enough that allowed us to take manual control of the robot through the *Teleoperation* behaviour, allowing us to drive the robot, manually and in real-time, to a desired position. Fig. 5 shows the 3D trajectory followed by the robot during the experiment and Fig.6 shows a detail of the delayed X position, shown by *NEPTUNE* in the local site, compared to the real X position gathered in the remote site.

5. FURTHER WORK

For the future work, we are planning several upgrades to the software. First, the client-server communication paradigm that currently uses *NEPTUNE* to gather the position from the remote site will be substituted by a publish-subscribe paradigm. Hence, instead of continuously interrogating the *Navigator* from the local site (which means sending a request and receiving the new position), the remote *Navigator* will send to the local site any new position it estimates (this means sending only one message for any new position). Moreover, we plan to introduce a prediction module within the local *NEPTUNE* in order to deal with the flickering. Therefore, the robot position in the local screen will be regularly updated with a new position if available or with a prediction if no new position has arrived. Initially, we plan to use a simple kinematics prediction assuming constant velocity. Then, depending of the results, and taking into account that *NEPTUNE* makes use the hydrodynamic model of the robot, a more complex prediction based on the hydrodynamics equation will be considered.

6. CONCLUSSIONS

This paper has presented a set of software tools that can be used for remote training in AUV control. The control loop is closed locally at the remote site where the robot is operating and hence the AUV works autonomously. The only information traversing the network is telemetry (position and attitude), used to represent a virtual view of the remote experiment, and the operator commands send using a joystick. In our experience, working with high speed communication links commonly available at universities and research centers, it is possible to run a remote experiment in real time. The observed time delay was small enough to allow the user to take manual control of the robot if needed. Nevertheless, the variability of this delay provoked a small flickering of the robot position in the local screen. It is worth noting that the *AvoidWall* behavior that operates on the robot prevents any possible collision with the walls of the water tank, ensuring the safety.

ACKNOWLEDGMENTS

This research was sponsored by the Spanish commission MCYT (DPI2005-09001-C03-01).

REFERENCES

Arkin, R. (1998). *Behavior-Based Robotics*, MIT Press, Cambridge, Massachussetts, USA.

Batlle E., Ridao P. And Palomeras N., (2005). A Survey Of Graphical Simulators For UUV Development, In: *Proceedings of the IARP International Workshop on Underwater Robotics for Sustainable Management of Marine Ecosystems and Environmental Monitoring*, Genoa, Italy. Page(S):185 - 194

Bruzzone G., Bono R., Caccia M., Veruggio G., Ferreira C., Silvestre C., Oliveira P., Pascoal A. (1999), Internet mission control of the romeo unmanned underwater vehicle using the coral mission controller, In: *Proceedings of the OCEANS 1999 MTS/IEEE Conference and Exhibition*. **Volume. 3** Page(s):1081–1087.

Bruzzone, G.; Bono, R.; Caccia, M.; Veruggio, G. (2001). A simulation environment for unmanned underwater vehicles development In: *Proceedings of the OCEANS, 2001. MTS/IEEE Conference and Exhibition*, **Volume 2**, Page(s):1066 - 1072.

Choi, S.K.; Menor, S.A.; Yuh, J (2000). Distributed virtual environment collaborative simulator for underwater robots. In*: Proceedings of the IEEE/RSJ International Conference Intelligent Robots and Systems*. **Volume 2**. Page(s):861 - 866 vol.2

Fossen, T.I (2002). Marine Control Systems: Guidance, Navigation and Control of Ships, Rigs and Underwater Vehicles, Marine Cybernetics AS, Trondheim, Norway.

Kuroda, Y.; Aramaki, K.; Ura, T. (1996). AUV test using real/virtual synthetic world; In: *Proceedings of the 1996 Symposium on Autonomous Underwater Vehicle Technology.* Page(s):365 – 372. Monterey (USA).

Palomeras, N., Carreras, M., Ridao, and Hernández, E. (2006). Mission control system for dam inspection with an AUV. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and System*, To appear. Beijing, China.

Ridao, P.; Batlle, E.; Ribas, D.; Carreras, M. (2004). Neptune: a hil simulator for multiple UUVs. In*: Proceedings of the OCEANS '04. MTS/IEEE TECHNO-OCEAN'04*,**Volume 1**, Page(s):524 – 531. Kobe (Japan).

Ribas, D., Neira, J., Ridao, P., Tardos, J.D. (2006). AUV Localization in Structured Environments Using an a Priori Map. In*: Proceedings of the 7th International Conference on Manoeuvring and Control of Marine Craft (MCMC'2006)*, To appear. Lisbon, Portugal.