Distributed Control of a Free-floating Underwater Manipulation System

K. Kapellos D. Simon
INRIA, BP 93, 06902 Sophia-Antipolis, France
email: <kapellos, dsimon>@sophia.inria.fr

S. Granier V. Rigaud
ISIA, 06904 Sophia-Antipolis IFREMER, BP 330, 83500 La Seyne/Mer
email: <sgranier, rigaud>@toulon.ifremer.fr

Abstract: Robotic applications are real-time dynamical systems which intimately combine different components ranging from high-level decision making and discrete-event controllers to low-level feedback loops. Tightly coupling the two last components and considering them in a formal framework permitted, in a centralized approach, both crucial properties to be proved and efficient implementation. We examine this coupling in the case of an architecture where discrete-event and low-level controllers are spatially distributed and we propose, in the framework of the ORCCAD methodology, different methods for their implementation. Their impact to the verification process is analyzed. The experimental evaluation of the proposed techniques uses the IFREMER free-floating underwater manipulation system VORTEX-PA10. The real-time programming aspects of the experiment are handled in the framework of the ORCCAD programming environment targeting the PIRAT real-time controller.

1. Introduction

In applications like autonomous vehicles, aircrafts and robots, stand alone computerized controllers are being integrated to form cooperating subsystems which can together provide improved functionality, reliability, and reduced costs. Modern and future machinery will therefore, and to some extend already does, include *embedded distributed real-time computer control systems.* In this context, considering a given hardware architecture, programming of an application must deal with algorithms distribution over the different locations and implies exchange of real-time data between subsystems. The present contribution gives insight on the programming of some aspects of such a distributed robotic applications.

The work reported here is motivated by the particular objective of specifying, validating and implementing missions on the Vortex-Pa10 underwater manipulation system developed at Ifremer, where the control computers are distributed on a network. In [8] we specified and validated by formal methods and realistic simulations an underwater structure inspection mission considering the whole system. Distinct control laws were designed for each subsystem which

were coordinated by a centralized discrete-event controller which rhythmed the logical evolution of the mission. It has been shown that a reasonable stabilization of the vehicle was achievable using acoustic or visual sensor-based control despite the arm motions. Besides, the correctness of the logical behavior of the mission has been formally verified taking advantage of the centralized implementation of the discrete-event controller.

This work is carried out in the framework of the development of the ORC-CAD programming environment [9], which allows the specification, the validation and the implementation of robotic reactive applications. It clearly separates the specification of the discrete-event controller which rhythms the logical evolution of the application from the data handling used to implement the control laws on the target architecture. This is achieved by designing an application as a hierarchical and structured composition of *Robot-Tasks* (RTs), which harmoniously integrate discrete and continuous time aspects, in *Robot-Procedures* (RPs) which mainly handle logical behaviors. Using the synchronous approach as the semantics of composition for RTs and RPs pave the way to apply formal verification methods (and especially 'model-checking' approaches) to prove the mission correctness from a logical and temporal point of view [4].

This approach has been already used in various robotic applications [6]: even in the case of multi-processor systems the discrete-event controller was always centralized (i.e. compiled as a single automaton) thus allowing for its formal analysis. However, this centralized approach is not always timely efficient or may be unsafe or impossible to implement on a naturally distributed system as failures in the communication links between the controllers may leave some subsystems totally out of control. It is expected that distributing the logical control code over the distributed hardware may improve subsystems survivability.

As an extension of this previous work we now experiment parts of an underwater mission in a pool [8]. In this paper our interest is twofold and combines theoretical considerations and experimental validations:

- propose a methodology to distribute the discrete-event controller of the mission. Two solutions are envisioned:
  1) separately design the controllers of the subsystems (i.e independently of its context) and the global controller, and compile them into several object programs located on the different nodes. In that case, we obtain asynchronously connected controllers for which global formal verification is impossible and validation must be carried out using simulation.
  2) globally design the mission controller, compile the mission specification into a single object program, and then distribute it on the nodes. Thus, the specification of the logical evolution of the mission and its formal analysis can be globally carried out while its implementation will be automatically distributed ensuring that the verified properties hold in the resulting distributed program. This methodology is based on recent theoretical results in the field of reactive systems theory [3] and is currently integrated in our control architecture.
- experimentally validate the simulated control laws. Simulations taking into

account most of the temporal features of the controlled system enlightened limits of the hardware and suggested improvements e.g. adaptive triggering of acoustic sensors. Preliminary experiments will be used to assess these simulation results and will be used to better calibrate the simulation model. We consider several phases of the mission: preparing the vehicle for cruising, reaching the working area, and finally performing motions of the arm while the vehicle is actively stabilized.

2. The Discrete Event part of the Hybrid Controller

2.1. ORCCAD overview

ORCCAD ([9]) is a development environment for specification, validation by formal methods and by simulation, and implementation of robotic applications. Its conception is articulated around two entities which formally characterize a robotic action, the *Robot-Task* and the *Robot-Procedure*.

The *Robot-Task* (RT) models basic robotic actions where control aspects are predominants. Typical examples are the trajectory tracking of an arm manipulator, the positioning at a desired pose of an underwater vehicle, ... It characterizes in a structured way continuous time closed loop control laws, along with their temporal features related to implementation and the management of associated events. A data-flow well identified mode of execution implements the algorithmical (control law) part of the RT. The considered events may be pre-conditions, post-conditions and exceptions which are themselves classified in type 1 (weak), type 2 (strong) and type 3 (fatal) exceptions. The reception of these events rhythms the evolution of the action according to a pre-defined scheme: the satisfaction of the pre-conditions leads to the activation of the control law. During its execution, if a specified exception occurs, it is handled according to its type; the reception of the post-conditions implies the ending of the action. A reactive process is used to implement this discrete-event part of the RT. Therefore, in a RT intimately cooperate a data-flow scheme and a discrete-event controller; section 2.2 details this connection. RT design mainly falls in the field of automatic control design using block-diagrams and sampled time while calls to the underlying operating system and the encoding of the reactive program are automatically generated.

The *Robot-Procedure* (RP) entity models a robotic action of variable complexity. For example, the cruising phase of the inspection mission schedules trajectory tracking navigation followed by automatic wall following associated to starting and stopping conditions. It specifies in a hierarchical and structured way a logical and temporal *composition* of RTs and RPs in order to achieve an objective in a context dependent and reliable way, providing predefined corrective actions in the case of unsuccessful execution of RTs. The composition concerns only the logical aspects of the RT and it is systematized thanks to a well defined interface. Therefore, the RP can be considered as a discrete-event controller which rhythms the sequencing of RTs (through their local discrete-event controllers) following a user-defined scheme (section 2.3 details this aspect). Thus the end-user is able to design a full robotic application using a library of pre-defined validated high-level actions without worrying with

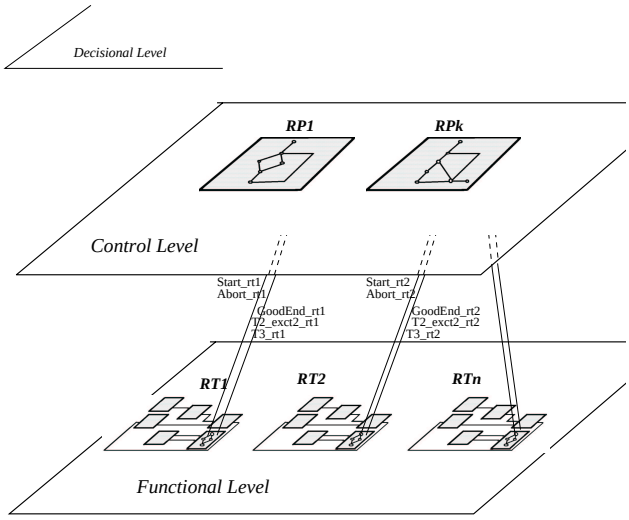

Figure 1. Control architecture

low level programming and implementation tricks.

Using robust control-laws and tuning the gains and parameters with a simulation tool like SIMPARC ([1]) ensures the stability of the physical system during RTs execution with specified performance. On the other hand the logical behavior of the RT is verified to satisfy critical properties of liveness and safety. The RP ensures a robust control of the physical system seen as a collection of RTs. Here, simulations are used to validate the transition phases and formal verification to prove the correctness of the logical and temporal behavior by checking critical properties and conformity with application's requirements. These well defined structures allows to systematize and thus automatize formal verification on the expected controller behavior.

The resulting control architecture is organized in three levels (Figure 1): in the *functional* one reside the RTs executing the low level control laws; thanks to the event driven interface they are sequenced by the RPs which are elements of the *control* level. Finally a *decisional* level should be ideally added on the top of the architecture to provide automatic or manual replanning.

2.2. Robot-Task Discrete-Event Controller

The evolution of a RT execution is characterized by three main phases, briefly speaking initialization, control law application and controlled ending. The activation of a RT starts after an external request, follows its local behavior rhythmed by the events monitored during the RT execution and signals its end, thus providing an *external abstract view* allowing for its manipulation. The role of the RT discrete-event controller is to model and implement this evolution.

We illustrate in Figure 2) the discrete-event RT controller in the form of an automaton where transitions are labeled by events : input one are prefixed by '?' and the output by '!'. The first phase copes with initialization aspects and starts after reception of the synchronization pre-condition event *Start_rt*. By

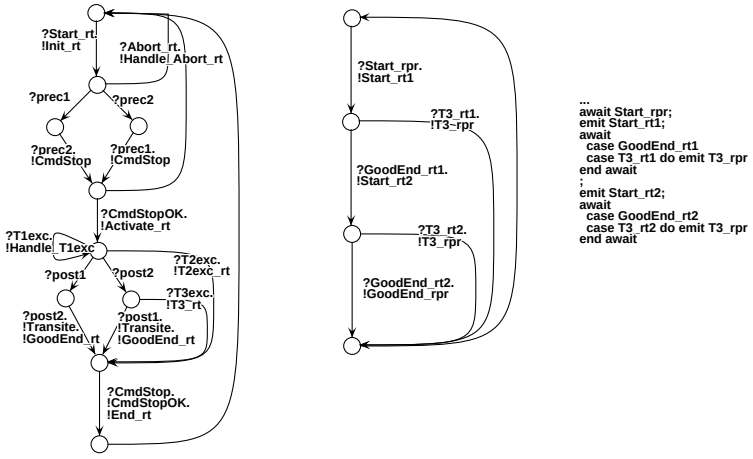

Figure 2. Robot-task and Robot-procedure local behaviors

the emission of the *Init_rt* output signal the discrete-event controller asks the services of a concurrently running process to prepare the RT execution : the necessary threads implementing the data-flow part of the RT are spawned and the used algorithms are initialized (communication devices, filters, memories, ...). Sequently, the inter-connected modules network is periodically executed, without sending commands to the actuators while the previous RT ensures the robot command until all the pre-conditions are satisfied. A request is then send to the previous RT discrete-event controller and after acknowledgment (*CmdStopOK*) the *Activate_rt* output signal asks from the current RT to start sending commands to the actuators thus entering in its second phase. During this phase the control law is applied to the robot and the presence of the specified exceptions and post-conditions are observed and handled. The post-conditions satisfaction implies the logical end of the RT, signaled by the *GoodEnd_rt* event and drives the RT to its third phase of execution, named transition phase. During this phase the commands are always send to the robot during the initialization of the following RT after which the ending code of the algorithms is executed and the threads are deactivated. We note that *Abort_rt* requests and type 2 or type 3 exceptions are handled at each phase of the execution of the RT signaled by the *T2_exception_rt* and *T3_rt* events. For clarity purposes in figure 2 a limited number of these transitions are designed.

2.3. Robot-Procedure Discrete-Event Controller

The RP discrete-event controller is aimed both to coordinate RTs and RPs in order to achieve a predefined behavior and to offer, as for the RT, the adequate event based interface which constitutes its abstract view. Consider for example a very simple RP, named *rpr*, sequencing two RTs, *rt1* and *rt2*. The behavior of the corresponding discrete-event controller is illustrated by the automaton figure 2 : upon the reception of the *Start_rpr* synchronization pre-condition signal the first RT is asked to be launched (*Start_rt1* event) and sequently

the controller waits possible terminations of the RT issued by its discrete-event controller. In the case of the *GoodEnd_rt1* the second RT is asked to be launched (*Start_rt2* event) while in the case of *T3_rt1* the RP is aborted in order to apply a security procedure.

2.4. Specification using the ESTEREL language

The activity of the RT and RP discrete-event controllers is naturally described in terms of responses to stimuli originated by the environment. The use of the *reactive* model to formalize their behavior is therefore a natural choice. Following this model one considers communicating systems that continuous interact with their environment. When activated with an *input event* a reactive system *reacts* by producing *output events* and returns waiting for the next activation. Moreover this choice is justified by the efforts of the computer science community to provide mathematically sound formalisms and integrated environments for the specification, verification and efficient code generation of such systems.

The ESTEREL language [2] is a member of the synchronous languages family based on a rigorous mathematical semantics, especially designed to program reactive systems where control aspects are predominant. Due to an adequate set of primitives an ESTEREL program is very close to the system's behavior specification. The RP discrete-event controller given as example in section 2.3 is simply specified by the ESTEREL program and is easy to understand without particular knowledge of the language. For verification and simulation/implementation purposes the ESTEREL code is compiled into automata which are translated into several target languages such as C or Ada.

In ORCCAD system, ESTEREL is used both to encode RT and RP discrete-event controllers. From the user point of view, the ORCCAD programming environment automatically generates RTs discrete event controllers through a dedicated window while currently the RP programming must be left to the end-user. However, an ORCCAD compliant task-level language targeting ESTEREL is currently under development [5].

3. Centralized vs. Distributed Control

In the ORCCAD methodology for the design of the robot controller the specification of the RTs and RPs is followed by: i) their verification, aiming to prove that the specification is conform to the end-user's requirements and ii) their implementation w.r.t. the specification. For the discrete-event part of the robot controller a crucial question is the choice of the semantics of the composition of the specified discrete-event controllers in order to obtain the global one which rhythms the logical evolution of the whole application. Let us take the particular example of the *Start_rt* synchronization event which is an output one for the RP discrete-event controller and input for the RT; how this synchronization signal is exchanged with respect to the reaction of the two controllers? Two solutions can be adopted: the *centralized* one where the global discrete-event controller is executed as a single process and the synchronization signals are synchronously broadcast within each controller, and the *distributed*

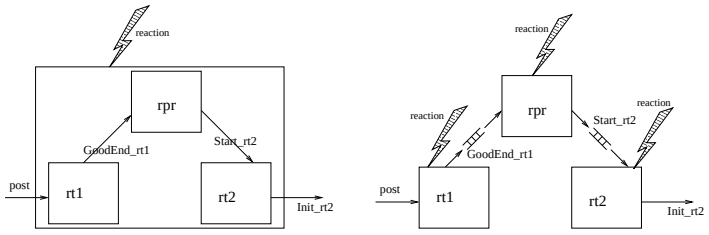

Figure 3. a) Synchronous and b) asynchronous compositions

one where the discrete-event controllers run on different processes (possibly on different processors) and the synchronization signals are asynchronously transmitted. These two solutions and their impact to the verification process are now detailed.

3.1. Centralized approach

The centralized approach consists in *synchronously* [2] compose the specified discrete-event controllers. This composition paradigm is illustrated figure 3a.

In reaction to some external event an atomic reaction occurs which propagates activation instantaneously to all discrete-event controllers at all levels of the hierarchy. Let us suppose the presence of the postcondition external event of the RT *rt1*. The reaction of the global controller to this event consists in activating the RT controller which broadcasts the *GoodEnd_rt1* signal. During the same reaction the RP controller receives this synchronization event and emits the *Start_rt2* event to the controller of the second RT. Thus *rt2* enters in its initialization phase emitting to the environment the *Init_rt2* output event which asks, as indicated in section 2.2, the preparation of the RT execution. We note here that the events *post, GoodEnd_rt1, Start_rt2, Init_rt2* are received and emitted at the same atomic reaction. Given also that ESTEREL compiler completely disappears the instantaneous communications from the resulting code time efficiency is ensured.

A second major advantage of the centralized approach is related to the verification aspects. It is known that a property verified by each of two discrete-event controllers separately is not necessarily verified when their interaction is considered. In the centralized approach a particular property is verified over the global behavior of the system taking into account even the particular details of each RT local behavior.

3.2. Distributed approach

Robotic or more generally mechanical systems are often composed by spatially distributed actuators and sensors in order to improve modularity, functionality and performance. Therefore, a centralized approach is not always feasible nor desirable in particular from the dependability point of view. Firstly, external events handling, in particular those concerned with fatal exceptions must be fast processed at the location they have been produced . In addition, for safety reasons, a minimum logical control must be provided at each robotic sub-system in order to give a certainly restricted but crucial decisional autonomy facing

failures such as communication interrupts.

Therefore, face to a distributed system a distributed approach for the implementation of the discrete-event controller of the system must be envisioned. Several methods are available to built such controllers :

- separately consider each RT discrete-event controller and the RP one, compile them into several object programs located on the different nodes and make them communicate through asynchronous channels (first-in/first-out files for example). Let us revisit our pedagogic example: in Figure 3b the three reactive discrete-event controllers communicate through two channels. In this configuration *rt1* discrete-event controller as reaction to the presence of the *post* event sends through the network the *GoodEnd_ rt1* event to the *rpr* discrete-event controller and ends its reaction; the *rpr* reacts emitting the *Start_ rt2* event to *rt2* and ends its reaction; finally the *rt2* reacts and emits the *Init_ rt2* event.
  This approach has a major drawback. Even if each reactive part can be analyzed and verified individually their composition through non-deterministic communications inhibit any verification on the global system. A second approach tends to remediate this drawback.
- synchronously compose all the RTs and RPs discrete-event controllers, compile them into a single reactive program and then distribute it on the nodes in such a way that the parallel execution of these codes implements the initial program and each node has only to perform its own computations. Given a synchronous program (in oc format) and distribution directives locating each signal to a node, the *ocrep* tool [3] automatically distributes the centralized program to the corresponding nodes. The major advantage of this approach lies on the global verification capability as properties verified on the centralized program remains valid on its distributed implementation. Nevertheless, this approach increases the load on the communication links since additional dummy signals are exchanged in order to correctly synchronize each program.

4. Experimental results

Let us now focus to the case study we used to evaluate some of the proposed methods. This example is an extension of the work done in the Union project [7] to assess the design and verification methods of ORCCAD with an underwater test-bed application. The mission takes place in a pool, using the Vortex vehicle fitted with a manipulator.

4.1. Experiment setup

Vortex is a Remotely Operated Vehicle (ROV) designed by Ifremer as a test-bed for control laws and control architectures. It is equipped with a set of six screw propellers and with a traditional sensing set such as compass, inclinometers, depth-meter and gyrometers allowing to measure most of its internal state. A video camera is used for target tracking tasks and a belt with eight ultrasonic sounders allows to perform positioning and wall following tasks. Vortex is also

equipped with an electric powered Mitsubischi Pa10 arm with 7 degrees of freedom to perform manipulation tasks (see Figure 5).

The vehicle control algorithms are computed on an external VME backplane running Pirat, a C++ library of objects dedicated to underwater vehicles control. At a higher level, i.e. control laws and mission management, the reactive synchronous ESTEREL language is used to design Robot-Tasks and Procedures, consistently with the ORCCAD approach. The control algorithms for the arm are run on a second VME backplane. As the two controllers only have a low bandwidth communication capability, control laws for Vortex and the Pa10 arm run independently and only short synchronization messages are exchanged on the communication link. This distributed architecture (Figure 4) is used to test the ORCCAD approach given the following informal end-user's requirements.

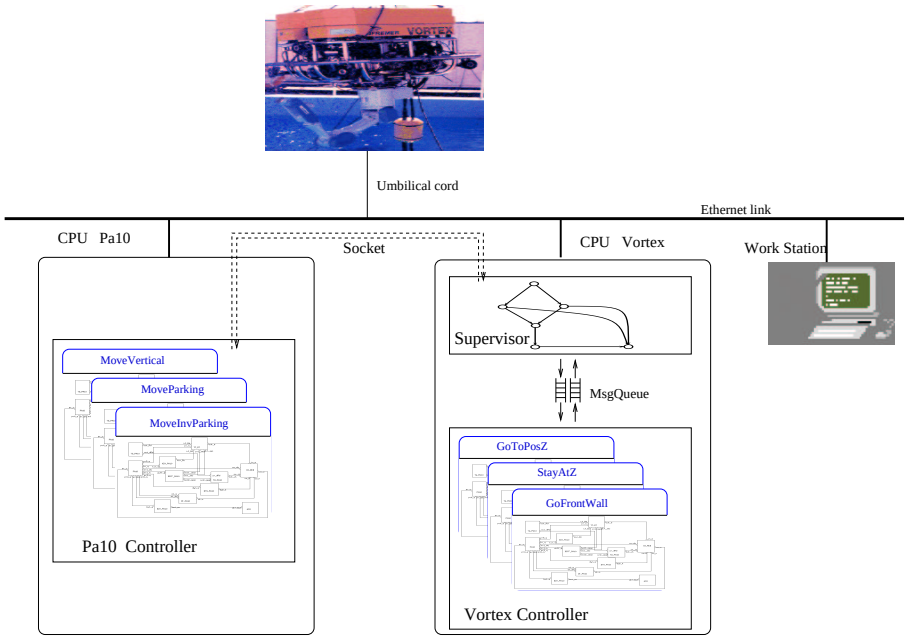

Figure 4. Experimental plant

Starting from the initial position, Vortex is set in station keeping mode at a pre-defined depth, using the on-board flux-gate for heading control. After completion of all necessary initialization[1] (e.g. the heading and depth set points are reached), the arm is moved backward in a safety position for navigation. Then the vehicle locks itself at a predefined distance of the front wall using acoustic sensors servoing. Once the vehicle is locked in the chosen corner the arm is moved forward and backward to check the platform stability under inertial and hydrostatic disturbances. The vehicle is then driven back to its

[1]It must be pointed out that the first initialization phase is also mission dependent e.g. choosing the initial heading set-point pointing towards the East direction will further drive the vehicle in the South-East corner of the pool.

initial position under the crank.

At any time the detection of a water leak or of an hardware failure must lead to a mission abortion leaving the system in a safe situation, i.e. setting an alarm and emergency surfacing with the arm locked in folded position. Other exceptions more specific to a given system or control algorithms are also defined inside the subtasks involved in the mission. Some actions are automatically triggered e.g. the first arm motion is triggered by an observer checking that the vehicle has reached its working depth. Others, e.g. the event starting the mission, are given through the keyboard thus sketching an embryonic teleoperation master station to be further developed.

This mission scenario naturally lead to split the mission in five main procedures: InitCruiseConfig consists in preparing the vehicle for cruising with the GoToPosZ RT. ReachWorkingArea is used to navigate in the pool (using the GoFrontWall vehicle's RT) until the vehicle reaches the inspection place. DoInspection is used to coordinate actions of the platform and of the arm to simulate the inspection of an underwater structure with the arm tip: it respectively uses the StayFrontWall and MoveJS RTs. GoHome is in charge of driving the vehicle to its homing position and preparing it to be pulled out. Emergency is always active to handle permanent recovery behaviors like triggering a fast ascent in case of water leak. Eleven RTs are used in this mission even if some of them are very simple like the Brake used to lock the arm during vehicle's motions.

Each of the reactive programs is generated automatically by the Orccad programming environment in the following way. The discrete-event controllers of the set of the RTs concerning the same physical resource are composed synchronously forming as many reactive program as the number of the controlled physical resources. RPs are synchronously composed in a separate program downloaded in the vehicle backplane. Asynchronous communications are also generated and established following a description of the target architecture. In our particular example (Figure 4) sockets are created for the communication between the RP and the PA10 reactive programs running on different backplanes while message queues are established between the RP and the Vortex programs which run on the same VME board. Instanciating the right communication mechanisms is automatically done from the software mapping specification.

4.2. Experiment results

Figure 5 on the right presents experimental results carried out while running the aforementioned mission scenario. The top plot shows the front acoustic sensors signals which are used in particular to stabilize the vehicle in front of the pool's wall while the arm is in motion. The bottom plot shows the pitch angle of the vehicle which is, due to the low accuracy of time profiling between the two backplanes, the best way to detect the moments during which the arm is in motion.

The necessary communicating devices (sockets or message queues according to the respective controllers localization) are automatically instantiated at

boot-time. Even if the two backplanes are linked through a non real-time Ethernet, signals between the different controllers are exchanged according to the mission specification e.g. observers checking the state of the vehicle are able to trigger actions of the arm and conversely. Signals sent through the keyboard are also correctly handled.

Some properties of the mission program have been checked before launching: for example, for safety reasons we want that the arm be motionless and locked when the vehicle is cruising. Thus, an abstract view of the mission automaton is built by reducing the global automaton to the only relevant signals. Figure 5 on the left shows this reduced automaton where it can be easily check that this property is verified whatever is the state of the mission: we can see that the activation of motions of the vehicle always immediately follows the activation of the BRAKE RT.

Besides the logical correctness of the reactive controllers behaviors the success of such an experiment relies on the efficiency of the used control laws. Although the gains which have been used for the experiment are very close to those found by simulation using Simparc [8] for most basic actions, the stabilization front of a wall using acoustic sensors was disappointing as the acoustic sensors loose a lot of data or provide absurd measures as soon as the pitch velocity is not negligible, may be due to multiple path propagation in the pool's corners. The gains of this control law had to be set to rather low values thus reducing the robustness of the stabilization w.r.t. the tether stiffness leading to non-repetitive results. It is expected that after careenage the new set of sensors and actuators will allow to improve the vehicle control efficiency.

4.3. Further improvements

The work presented here is a preliminary phase in the design of an underwater *distributed* teleoperated manipulation system. Further improvements will include the following:

- In this experiment we have only actually test one way of distributing the control code using asynchronous communication between the controllers. The *ocrep* tool for automatic distribution of synchronous programs will be tested and the additional communication cost will be evaluated.
- The code repartition has been done on a Robot-Task basis rather than on a Procedural one. Thus, some signals which are only relevant for the vehicle control are still exchanged between the two backplanes while the granularity of the actions described at the mission level should be made larger. The way of functionally distribute a robotic application will be further explored.
- Signals sent through the keyboard only sketches a master teleoperation station where an additional reactive program will be in charge of the management of the station (e.g. for managing the displays on a context dependent basis) to provide assistance to the human operator.

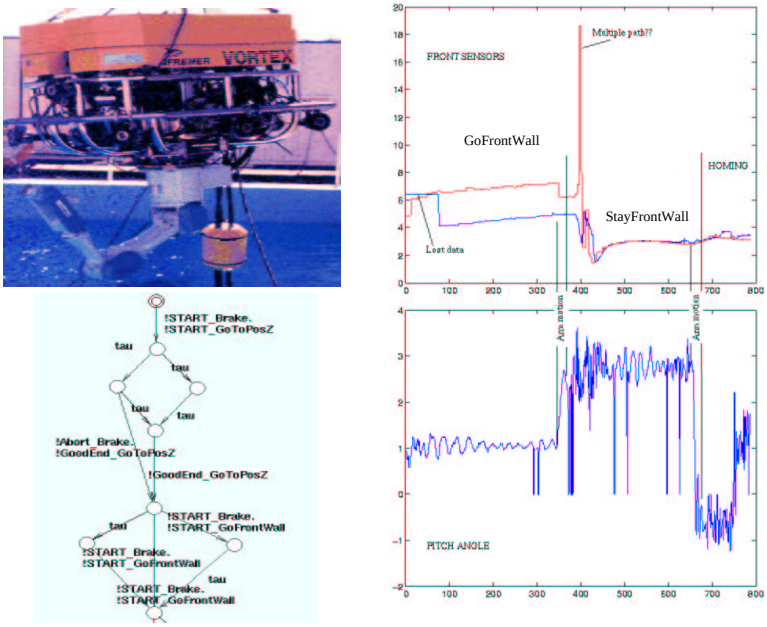

Figure 5. Some experimental results

References

[1] C. Astraudo, J.J. Borrelly, "Simulation of Multiprocessor Robot Controllers", *Proc. IEEE Int. Conf. on Robotics and Automation*, Nice, May 1992

[2] G. Berry, G. Gonthier: " The Synchronous Programming Language ESTEREL: Design, Semantics, Implementation", *Science Of Computer Programming*, Vol 19 no 2, pp 87-152, 1992.

[3] P. Caspi, A. Girault, and D. Pilaud. Distributing reactive systems. In *7th Int. Conf. on Parallel and Distributed Computing Systems,* Las Vegas, Oct. 1994.

[4] B. Espiau, K. Kapellos and M. Jourdan : *Verification in Robotics: Why and How?*, Robotics Research, the seventh International Symposium, octobre 1995, Giralt and Hirzinger editor, Springer Verlag.

[5] B. Espiau, K. Kapellos, E. Coste-Manière and N. Turro: "Formal Mission Specification in an Open Architecture", *Isram'96*, Montpellier, May 1996.

[6] K. Kapellos, S. Abdou, M. Jourdan and B. Espiau: "Specification, Verification and Implementation of Tasks and Missions for an Autonomous Vehicle", *4th Int. Symp. on Experimental Robotics*, Stanford, 1995.

[7] V. Rigaud e.a. (Union team): "Union: Underwater intelligent operation and navigation", to appear in *IEEE Robotics and Automation Magazine*, Special Issue on Robotics & Automation in Europe.

[8] D. Simon, K. Kapellos and B. Espiau, "Control laws, Tasks and Procedures with ORCCAD: Application to the Control of an Underwater Arm", *preprints of 6th IARP workshop on Underwater Robotics*, Toulon, March 1996.

[9] D. Simon, B. Espiau, E. Castillo and K. Kapellos: "Computer-aided Design of a Generic Robot Controller Handling Reactivity and Real-time Control Issues", *IEEE Trans. on Control Systems Technology*, vol 1, no 4, Decembre 1993.