Robotica (1997) volume 15, pp 111–115. © 1997 Cambridge University Press

ORCCAD: software engineering for real-time robotics A technical insight

D. Simon,† B. Espiau,‡ K. Kapellos† and R. Pissard-Gibollet‡

SUMMARY
The ORCCAD programming environment for robotic systems gathers control laws in continuous time at the low levels and discrete time logical aspects at higher levels. Based upon a formal definition of robotic actions, complex applications can be designed, verified and generated incrementally. The approach and tools prototypes have been validated through several applications.

KEYWORDS: ORCCAD; Robotic systems; Control laws.

1. GOALS AND CONCEPTS

Actual robotic systems range from cooperative manipulators to autonomous vehicles. They have in common a continuously increasing complexity which makes more and more difficult the needed integration of issues raised by automatic control, sensor data processing and computer science areas. The goal of a control architecture is then to organize coherently all the involved subsystems so that the global system behaves in an efficient and reliable way to match the end-user's requirements.

Robotic systems belongs to the class of hybrid reactive and real-time systems in which different methods and tools of programming and control are to be used. From the users and programmers' point of view, the specification of a robotic application must be modular, structured and accessible to users with different expertise, from the *control systems engineer to the end-user.* The ORCCAD[1] environment is aimed at providing such users with a set of coherent structures and tools to develop, validate and encode robotic applications in this framework.

The formal definition of a robotic action is a key point in the ORCCAD approach. The *Robot-Task* (RT) models basic robotic actions where control aspects[2] are predominants, like the hybrid position/force control of a robot arm or the visual servoing of a mobile robot. The RT characterizes in a structured way closed loop control laws in continuous time, along with their implementation temporal features, and the handling of associated events. These events are pre-conditions, post-conditions and exceptions which are themselves classified in type 1 (weak), type 2 (strong) and type 3 (fatal) exceptions. This set of signals defines an event-based abstract view of the RT.

The characterization of the interface of a RT with its environment through typed input/output events allows to compose them easily in order to construct more complex actions, the so called *Robot-procedures* (RP), while hiding most implementation details. The aim in designing this entity is to be able to define a representation of a robotic action that could fit any abstraction level needed by the mission specification system. In its simplest expression, a RP coincides with a RT, while the most complex one might represent an overall mission. Briefly speaking, it specifies in a structured way a logical and temporal arrangement of RTs 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.

These formally defined structures[3] associated with synchronous composition, thanks to the use of the *ESTEREL* language,[4] allows to systematize and therefore to automatize formal verification about the expected controller behaviour. In complement of two other features of ORCCAD, i.e. the object-oriented model and the possibility of automatic code generation, this capability of verification is a key point for meeting the requirements of safety in the programming of critical applications.

2. ROBOT-PROCEDURES THROUGH AN EXAMPLE

We focus in this section on the highest level of programming in ORCCAD, the Robot-procedure, in order to emphasize original aspects of design and verification issues.

While the events of type 1 are locally processed in the RT, e.g. by parameters modification,[5] exceptions of type 2 are handled by the RP, leading either to synchronize with, or to switch to a different RT in nominal situations or to call the planning level, if any, in case of failure. Type 3 exceptions lead to the abortion of the mission through a safe emergency scenario which may be context-dependent. For the mission designer, this set of signals with the associated behaviours represents *abstract* views of RTs and RPs hiding all specification and implementation details. This allows the incremental design and validation of procedures of variable complexity.

For example, let us consider an inspection procedure to be achieved by an underwater manipulation system (Figure 1), described in details in another paper:[6] the goal of the procedure is to control the motions of an arm

† INRIA Sophia-Antipolis, B.P. 93, 06902 SOPHIA-AMTIPOLIS Cedex (France).
‡ INRIA Rhone-Alpes, 655 avenue de l'Europe, 38330 MONTBONNOT ST MARTIN (FRANCE).

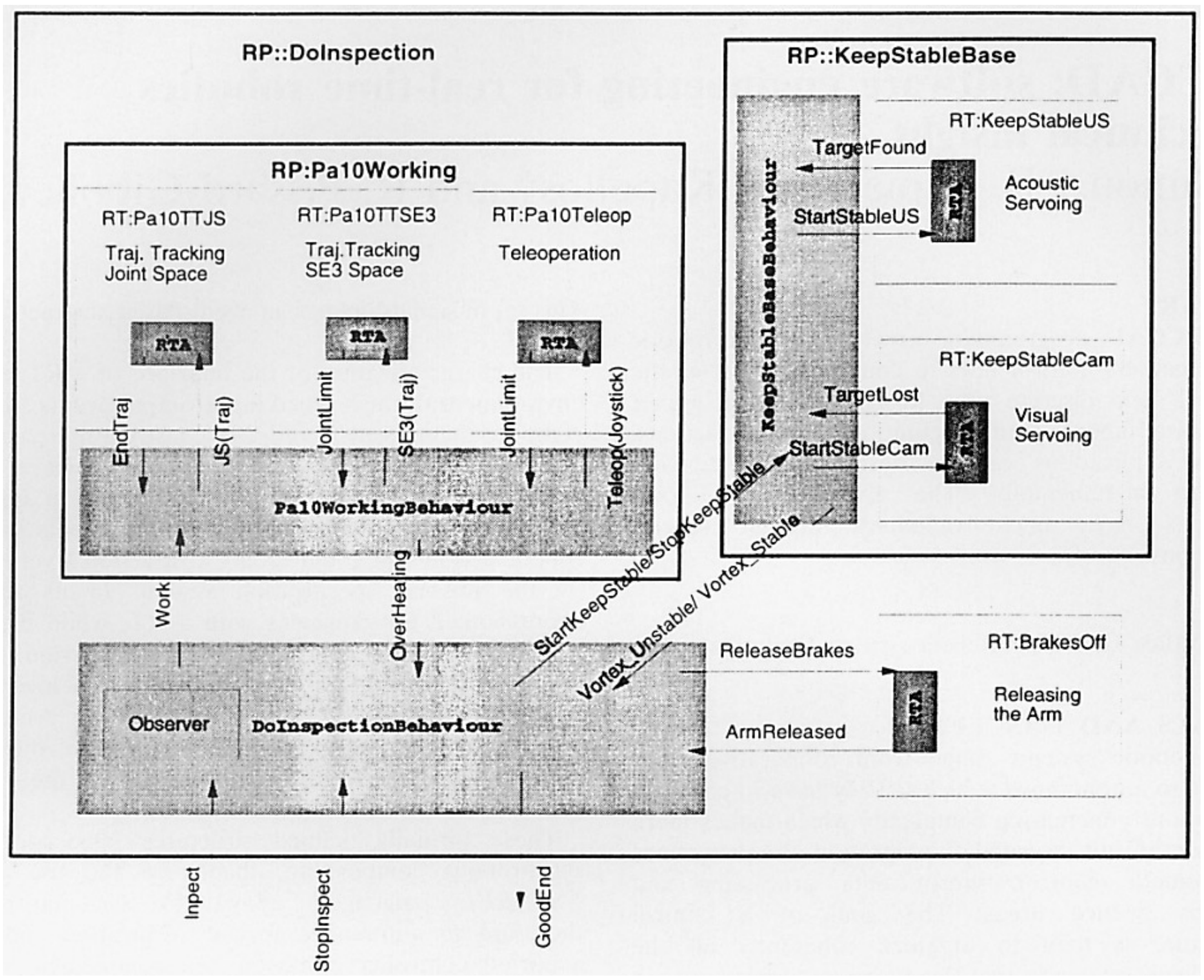


Fig. 1. Outline of an inspection procedure.

fitted on an underwater vehicle. The two subsystems are controlled separately but must be coordinated.

Firstly two first level procedures are designed, one for the arm and the another one for the vehicle. The trajectory tracking for the arm can be controlled in joint space, in operational space or through a teleoperated mode according to signals sent by the human operator or issued by the RP reactive behaviour. The vehicle is usually stabilized by visual servoing on a known target. In case of failure (video signal loss) the control is switched to station keeping using acoustic sensor upto recovering stability under visual control. This is a common situation where several RTs can be chosen to achieve a common goal according to the state of the mission. At run-time it is ensured that control laws never compete to control a same physical device (i.e. they are mutually exclusive).

These first level RPs are encapsulated in a new one designed to synchronize the motions of the arm and the effective stabilization of the vehicle. This RP may be specified through an user-oriented language[7] as shown down below in Figure 2.

```
PAR{
SEQ(do
          Pa10BrakesOn
     until  BaseStabilized;
     repeat 2 times
          Pa10SafeMoveJS;
          Pa10SafeMoveSe3;
     end repeat
     ;
     do
          Pa10SafeMoveJs
     until  ParkingPositionReached
     emit  InpectionOK
     ) ;

SEQ(do
     loop
          BaseSearchTarget;
            emitBaseStabilized
          KeepStableBase;
     end loop
     until  (InpectionOK)
   ) }
```

Fig. 2. Control code of a procedure.

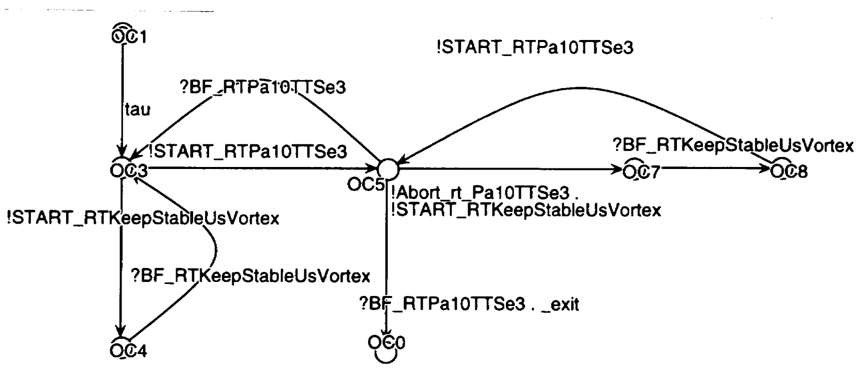


Fig. 3. Abstract view at RT level of the inspection RP.

When compiled the control code of this procedure expands in more than 300 *ESTEREL* statements which correspond to the logical behaviours of all the embedded RPs and RTs and also to the management (suspension and resumption) of the real-time tasks involved in the mission. Obtaining all the logical control code with *ESTEREL* allows to perform formal verification at every step of RTs and RPs design.

Crucial properties like *safety* (any fatal exception must always be correctly handled) and *liveness* (the RP always reaches its goal in a nominal execution) can be checked by building abstract criteria from the user's exceptions specification. For example it can be proved that the underwater system will perform an emergency ascent in case of water leak detection in any state of the mission.

More specific properties like the *conformity of the RP behaviour with respect to the requirements* are verified by observing abstract views of the logical behaviour. The control automaton is then reduced to an equivalent one where only relevant signals are visible. For example the abstract view at the RT level of the inspection procedure (Figure 3) allows to check that the arm and vehicle motions are correctly synchronized, i.e. that the arm is allowed to move only when the vehicle is stabilized in front of its target. Conflicts detection can be checked in the same way.

Finally, temporal properties and performance assessment can be checked by both formal methods using model checking techniques,[3] and by additional simulations[8] with SIMPARC.

3. SOFTWARE COMPONENTS IN ORCCAD

Briefly speaking, the ORCCAD system is a set of tools (see Figure 4) to design, verify, simulate and execute a robot control application, following the approach previously described.

The **Orccad Kernel** is a C++ library implementing the class hierarchy proposed for designing a robot controller in the ORCCAD model.[1]

The **Orccad Interface** is a Very Graphical User Interface (VGUI) aimed at providing an easy way to specify a robot controller following the ORCCAD methodology, i.e. to create instances of the ORCCAD class hierarchy. The interface is used to design objects of increasing complexity starting from elementary items called Modules, up to Robot Tasks and Robot Procedures. Both the continuous and the discrete-time aspects can be considered. Verification and code generation tools for simulation and execution can be activated through the interface.

The **Orccad Verif** components allows to formally analyze selected properties of the controller behaviour after composition of RTs. It uses AUTO and AUTOGRAPH tools.[9] Original aspects has been previously emphasized through an example.

The **Orccad Simul** tool generates the code of a robot controller for use in the SIMPARC[10] simulator. Compared with others simulation softwares commonly used in robot control, a specific feature of Simparc is that it allows to simulate both the plant dynamics and the important temporal characteristics of the controller, like sampling rates and communication delays, even including the basic features of real-time operating systems. We can therefore simulate as far as possible the effect of actual code execution.

The **Orccad Exec** component generates the *C*++ code for a dedicated robot controller. This generated code is independent of the real-time target environment. The generated system is afterwards integrated with the run-time libraries of the target environment. The present target systems are VxWorks* and Solaris (using Posix threads). The robot control software is divided in three parts:

– The computation of the continuous-time part of the system (control laws and observers). The tasks are usually periodic, using asynchronous communications and message passing mechanisms between them (producer/consumer type).

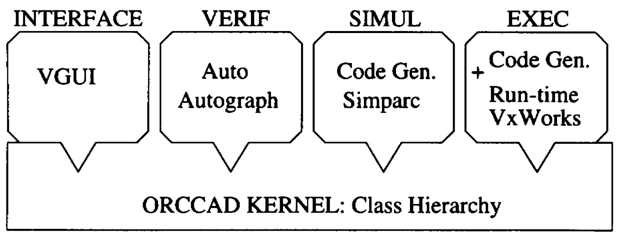


Fig. 4. ORCCAD Tools.

* Trademark of Wind River System.

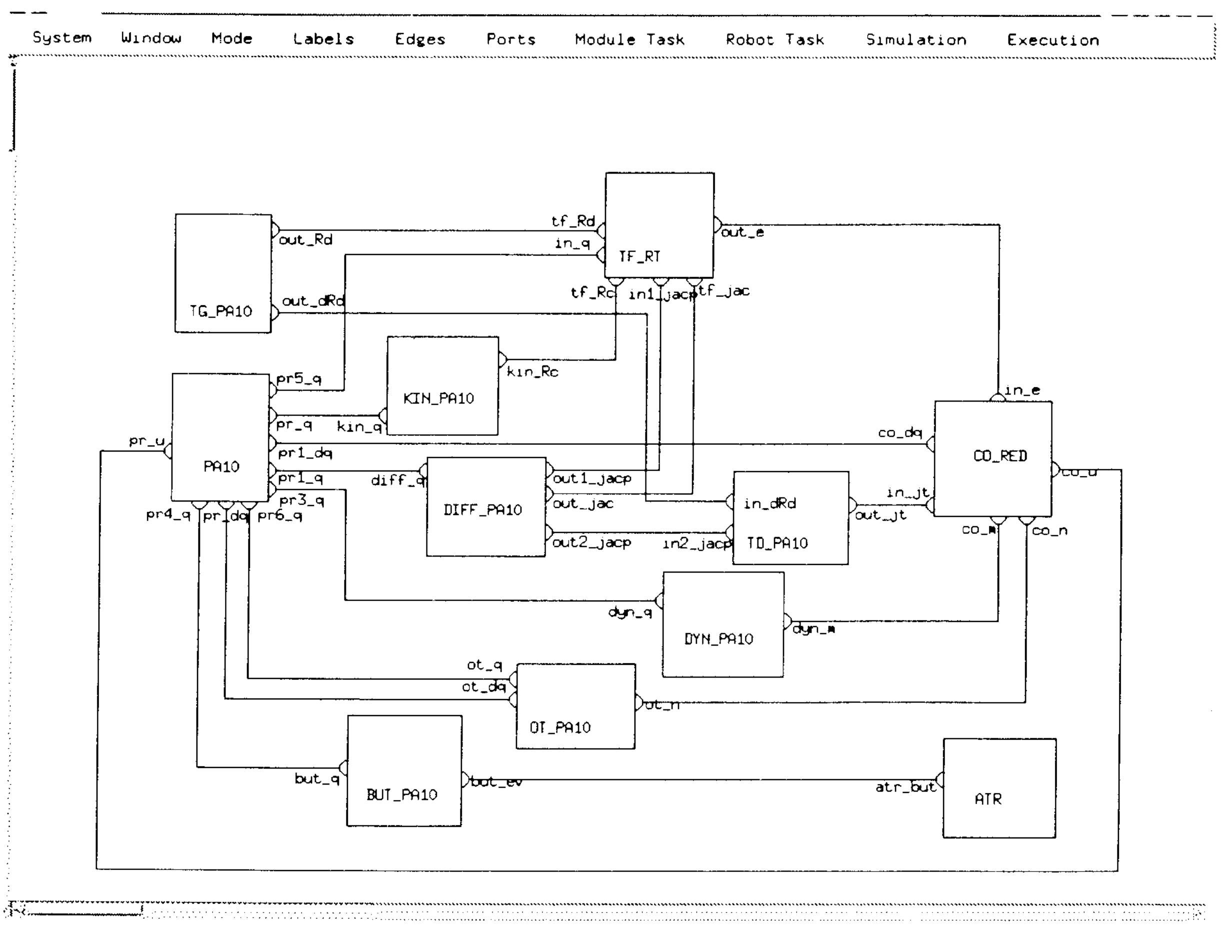
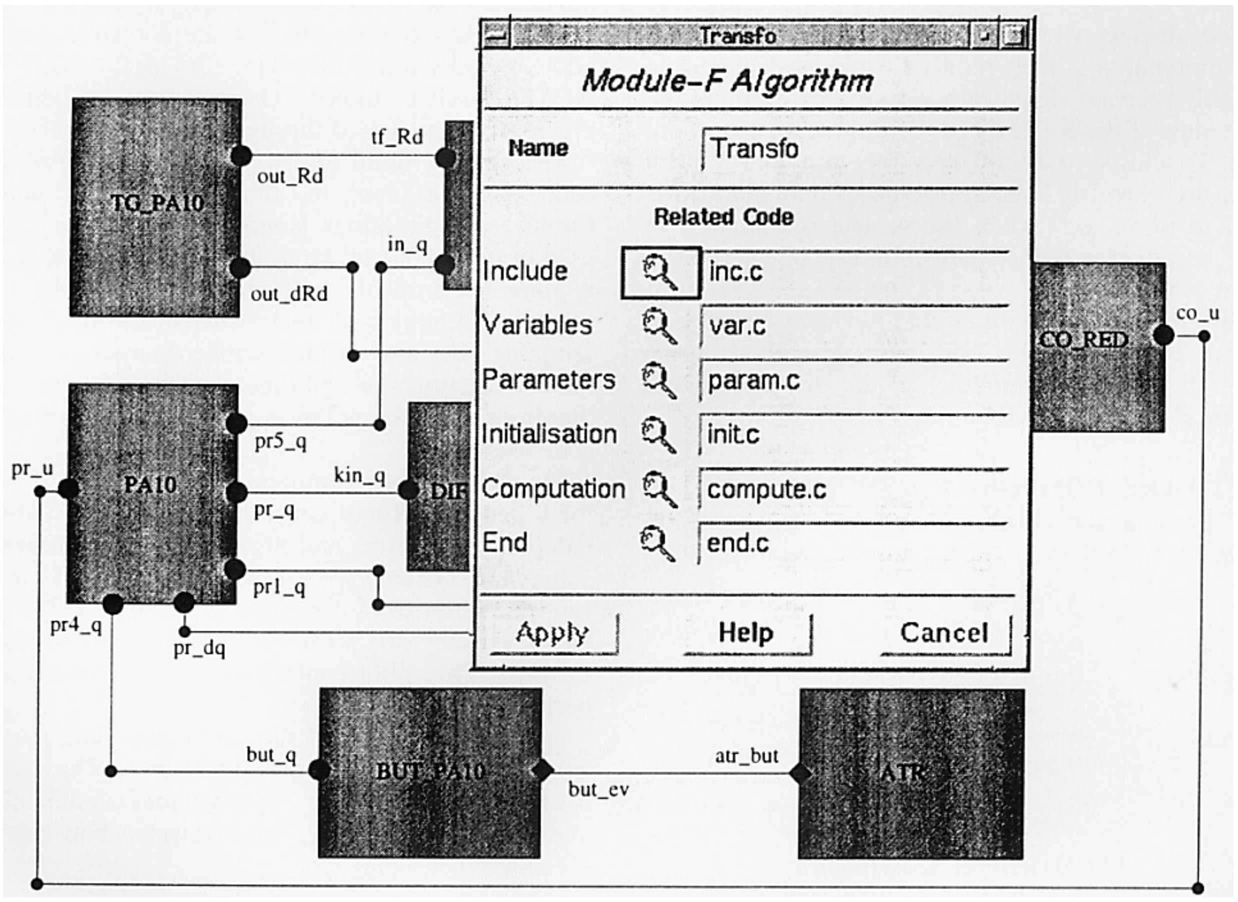


Fig. 5. An example of Robot-task Design.



Name: Transfo

Related Code

Include	inc.c
Variables	var.c
Parameters	param.c
Initialisation	init.c
Computation	compute.c
End	end.c

Fig. 6. VGUI for Robot-tasks Design.

– The discrete event controller: the synchronous reactive approach (*ESTEREL*[4] language) is used to compose RT-based behaviours as well to coordinate RTs switching and synchronization.

– The interface between the synchronous controller and the asynchronous items.

4. SOFTWARE AVAILABILITY AND APPLICATIONS

The ideas and tools of ORCCAD have been already tested on different systems through prototype versions. RTs for robot arm control (task-space servoing, singularity handling) have been designed and encoded using a first version of the G.U.I. depicted in Figure 5.[5,11] Sequencing of RTs to achieve a robot mission, where visual servoing and Cartesian space control were involved, was experimentally assessed with a wheeled mobile robot.[12] The design and the formal verification of RPs, both from the logical and temporal point of view, have been tested through the control of a virtual train of electric cars.[13,14] Several events had to be considered, like the loss of sensory information or the need for additional braking power. The control of complex underwater systems[6] is currently a support for improvements of the ORCCAD system from both the programming easiness and run-time efficiency points of view.

Based on the returns from all these tests, performed in real conditions, a first stabilized version of ORCCAD is presently under development (see an example of the final interface in Figure 6). We hope it will be available as a freeware by Spring 1997.

References

1. 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* **1,** No. 4, 213–229 (December, 1993).
2. C. Samson, M. Le Borgne and B. Espiau *Robot Control: the Task-Function Approach* (Clarendon Press, Oxford Science Publications, U.K., 1991).
3. B. Espiau, K. Kapellos, M. Jourdan and D. Simon, "On the Validation of Robotics Control Systems. Part I: High level Specification and Formal verification" *Inria Research Report* No 2719 (November, 1995).
4. G. Berry and G. Gonthier, "The Synchronous Programming Language ESTEREL: Design, Semantics, Implementation" *Science Of Computer Programming* **19,** No. 2, 87–152 (1992).
5. K. Kapellos and B. Espiau, "Implementation with Orccad of a Method for Smooth Singularity Crossing in a 6-DOF Manipulator" *Inria Research Report* No 2654 (September, 1995).
6. D. Simon, K. Kapellos and B. Espia, "Control Laws, Tasks and Procedures with ORCCAD: Application to the Control of an Underwater Arm" *Preprints of 6th IARP workshop on Underwater Robotics,* Toulon, France (March, 1996). (In press).
7. B. Espiau, K. Kapellos, E. Coste-Manière and N. Turro, "Formal mission specification in an open architecture" *Preprints of Intl Symposium on Robotics and Manufacturing ISRAM96,* Montpellier, France (May, 1996). (In press).
8. D. Simon, E. Castillo and P. Freedman, "On the Validation of Robotics Control Systems. Part II: Analysis of real-time closed-loop control tasks" *Inria Research Report* No 2720 (November, 1995).
9. G. Boudol, V. Roy, R. de Simone and D. Vergamini, "Process calculi, from theory to practice: Verification tools" *International Workshop on Automatic Verification Methods for Finite State Systems, LNCS,* Grenoble (Springer Verlag, Berlin, 1990) pp. 1–10.
10. C. Astraudo and J.J. Borrelly, "Simulation of Multiprocessor Robot Controllers" *Proc. IEEE Int. Conf. on Robotics and Automation.* Nice (May, 1992) pp. 573–578.
11. K. Kapellos "Environnement de programmation des applications robotiques réactives" *Ph.D. dissertation* (Ecole des Mines de Paris, Sophia Antipolis, France, November, 1994).
12. R. Pissard-Gibollet, K. Kapellos, P. Rives and J.J. Borrelly, "Real-Time Programming of Mobile Robot Actions Using Advanced Control Techniques" *4th Int. Symp. on Experimental Robotics,* Stanford, USA (June 30–July 2, 1995) pp. 345–357.
13. K. Kappelos, S. Abdou, M. Jourdan and B. Espiau, "Specification, Formal Verification and Implementation of Tasks and Missions for an Autonomous Vehicle" *4th Int. Symp. on Experimental Robotics,* Stanford, USA, (June 30–July 2, 1995) pp. 257–262.
14. M. Parent, S. Abdou and B. Espiau, "Specify, Validate and Implement Missions for Autonomous Vehicles" *Preprints of Intl Symposium on Robotics and Manufacturing ISRAM'96,* Montpellier, France (May, 1996) (In press).