15th Triennial World Congress, Barcelona, Spain

EVENTS AS A KEY OF AN AUTONOMOUS ROBOT CONTROLLER

J.D. Carbou, D. Andreu, P. Fraisse

LIRMM, Robotic Department
161 rue Ada, 34392 Montpellier, France
(e-mail: andreu@lirmm.fr, tel: (33) 04 67 41 85 15)

Abstract: The paper focuses on the development of an event-driven autonomous robot controller. Such systems being intrinsically hybrid, the model that is exposed is a hybrid model based on the cooperation of a discrete-event part and a continuous part. Then, continuous and discrete models are concurrently dealt with; the articulation model's different parts results from event interactions. According to this event-based conception, the controller architecture is hierarchically organized in three levels. The supervisory control is based on the management of contexts of execution. *Copyright © 2002 IFAC*

Keywords: event-driven autonomous controller, architecture, hybrid modeling.

1. INTRODUCTION

Mobile manipulators, aimed to operate in hazardous and/or dangerous environments, are used to replace the operator when this intervention would have been difficult, unsafe or impossible. A typical example is a terrestrial mobile manipulator handling explosive charges in building sites. Such systems are often remotely operated but communication limitations - varying network time delays (Leleve *et al.* 2000) - do not allow the operator intervention to be directly done at the lower level (real-time control one). So the system must come with an autonomous controller at least able to do on-line adaptation, thus allowing the mission programming or the tele-operation to be done at a high level of abstraction.

This controller design is based on the dichotomy of the global system into a set of independent or coupled subsystems. They can be exploited in a coordinated way (concurrent execution with synchronization) or in a mutual exclusive one, depending on the effective context of execution. A context of execution, also called a situation, is characterized by the robot configuration, the current objective of the mission and the environment state. Consider for instance the case of our terrestrial mobile manipulator that is a terrestrial vehicle (6 directive and propulsive wheels electric vehicle) equipped with a rear PUMA 560 arm. Hence composed by two subsystems, a vehicle on the one hand and a robotic arm on the other. When motion and manipulation are not simultaneously required, depending on the mission objective, independent control laws can be used. But in some cases combining both subsystems to perform specific motions can be interesting. The simultaneous motion of both the vehicle and the arm in order to go from an initial position/orientation of the end-effector to a final one is performed using a control law involving state variables of the two subsystems. In such case, the robot control is not expressed as the coordination of independent actions on each subsystem (concurrently executed) but as a control mode strongly coupling them. Thus, the controller must use different methods to control the mobile manipulator according to the requirements of the mission objective. Moreover, when dealing with autonomous robots, the controller must also be able to face unpredictable constraints (obstacles) in reaching the given goal. As a consequence, the method of controlling the robot also depends on the robot configuration as well as on the environment state.

So, this autonomous robot controller design is based on contexts of execution. To each subsystem or combination of subsystems (this set being intrinsically restricted) is associated a context manager in charge of selecting the appropriated control method. This context manager, which ensures supervisory control, will only be described for the arm subsystem. However, the overall system architecture follows the same principle.

Furthermore, when dealing with such intrinsically hybrid systems the coexistence of two different "worlds", continuous and discrete ones, must be taken into account. From a modeling point of view, continuous (state-space equations) and discrete (Petri Nets, PN) models must be concurrently dealt with. From an implementation point of view, the controller has synchronous (sampled) and asynchronous (sporadic) processes. A control architecture of these complex systems is thus offered as well as a hybrid modeling based on PN. The impact of this characteristic is considered both for the controller architecture design and the controller specification representation.

This paper is organized as follows. The controller architecture is described in the next section. Both design and real-time considerations are evoked. In a third section, a hybrid modeling approach is presented and illustrated on the arm controller of the mobile manipulator on which this work has been applied.

2. THE CONTROLLER ARCHITECTURE

2.1. *The controller design*

Several levels of abstraction and a high degree of modularity are required to face the process complexity: the control architecture is hierarchically structured (Alami *et al.* 1998, Santos *et al.* 2000, Simon *et al.* 1994, Borelly *et al.* 1998).

It is composed of several modules, organized in three layers; namely, the mission layer (irrelevant to this paper), the supervisory and execution control layers, as depicted in Fig. 1. This organization is divided in two levels of abstraction, even if structured in several layers (Simon *et al.* 1998, Wang *et al.* 1991, Fleury *et al.* 1994). The upper level, constituted by the managers, is concerned with behavior's decisional aspects system. The mission manager is working on sequences of goals and precedence relationships among them. As a supervisory controller, the context manager implements tactical specifications on execution module calls, switching and sequences (each goal corresponding to a sequence of sub-goals). The lower level is concerned with real-time execution aspects of the system, the event based supervisory and control loops belong to this level. The context manager is at the interface of the two levels; on the one hand it belongs to the upper one as it is in charge of "local" decision-making, but on the other hand it works under real-time reactivity constraints.

The context manager has to handle exteroceptive (obstacle) or proprioceptive (singularity) constraints. So, it may sporadically be forced to leave the current sub-goal to face the encountered constraint. Such an unpredictable reaction usually implies the addition of a context dependent sub-goal into a previously established schedule (logical and temporal arrangement) (Simon *et al.* 1998). The context manager is not really like a sub-goal scheduler as it deals only with the immediate sub-goal. The sub-goal schedule is thus dynamically established according to the context and the upper goal (memorized final objective), so it is more an "adaptive sequence" than a schedule. The sequences of sub-goals to execute in order to achieve the upper goals, as well as contextual sub-goal switching, are specified on the control manager Petri Net based model. The effective executed sequence reflects the autonomous control switches, resulting from the "local" (context manager) decision- making.

From an architectural point of view, the context manager role, a tactical one, is consequently to guide (adapt) the robot behavior by a dynamical configuration of the synchronous real-time control modules according to the effective situation (i.e. the current context of execution). This approach contributes to the robustness of the system and preserves its reactivity.

2.2. *An event-driven robot controller*

In the case of a pure continuous system, supervisory control is in charge of monitoring the system. Thus it detects when it is no longer in its optimal state, and acts on the local control law to compensate it. When autonomous robots are involved, supervisory control also has to select the appropriate control law, according to the effective context of execution. It then performs a one-line adaptation of the robot behavior. The supervisory controller may be viewed as enabling or disabling continuous systems (control laws) according to a specified tactical sequencing of sub-goals and a specified way of dealing with unpredictable constraints (e.g. singularity). That means that each control law performs, if enabled, a local closed-loop control that can be started/ended at any time by a supervisory control decision. The sequence evolution then implies a control law switching.

This evolution results from event occurrences such as position reached, contact reached, singular configuration, etc. However, for a given context of execution, only a sub-set of events corresponds to pertinent phenomenon. For instance, singular configuration detection is not required when using a joint space control law and obstacle presence has not to be treated when the mobile gets stopped. This means that only "significant" events can be monitored when managing situations. An event detection configuration is consequently performed each time the context manager evolves. Such a dynamical configuration is obtained by the definition of the "selected" events from the open set of specified types of events. Two types of event observation functions carry on the event detection: the event generator (EG) and the event estimators (EE) (Andreu *et al.* 1996). The latter are dedicated to particular events (model based detection). An EG (it is more a functionality than a module, so several event generators can be used) has to extract event occurrences from periodic measures according to given event generation rules. Crossing threshold by one, or a set of, (sampled) continuous variable is a usual rule of detection. An EE (also a functionality) has to build event occurrences from model based computation (cf. 3.3).

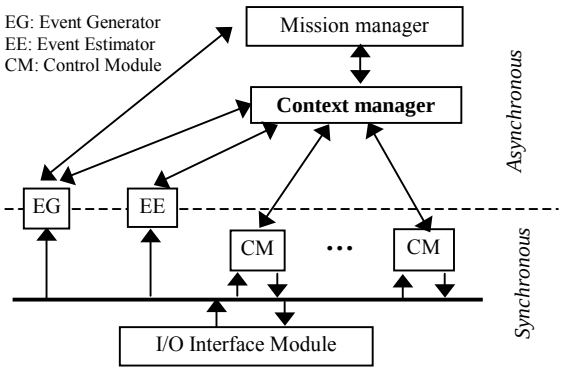


Fig 1: arm controller architecture

From a real-time point of view, the event detection is at the interface between asynchronous and synchronous worlds (Fig. 1). Events correspond to sporadic phenomenon, whose occurrences are factual rather than periodical. Their detection is carried on periodic measures; it is an asynchronous notification based on a synchronous system observation.

2.3. *Control laws description*

An open set of control laws (Degoulange *et al.* 1993) has been developed according to the required manipulator capabilities, and in terms of actions the robot has to be able to perform.

The joint space position control law (CPA). This control law, not sensible to singularities (singular positions linked to the lining up of some axis of the manipulator), is used to stop the manipulator, for instance for commutation purposes. Indeed, trajectory generation being locally computed (i.e. within the control module), the arm is stopped while switching in order to avoid latency effects, such as unstability.

The joint space position control law referenced in the task space (CPARC). Based on the CPA law, the difference dwells in the task space defined final point. This point is then converted in the joint space. It has the advantage of allowing the operator to define a motion in the task space, regardless of the manipulator model. This law is used to deal with singularities or to carry out any motion in a unconstrained space (the prediction of the arm trajectory in the task space not being immediate).

The hybrid position/force control law (CHP). The cartesian space is divided into two complementary and orthogonal spaces respectively the position and force, whose selection is expressed by a matrix. This law requires less computation time than CHE (see below). It is used to perform regular motions or surface following, but only in absence of constraint on environment preservation. Indeed, if the force control-loop breaks, there is no command left in the direction of force regulated axis.

The external force control law (CHE). This law can be used for the same goals as CHP, moreover all the axis being regulated by the position control-loop, it guarantees a non-destructive behavior, even in case of force regulation error.

Impedance control law (CIMP). This particular law allows getting a spring-damper like behavior. It is used as such for palpation and contact searching goals, as well as for cushioned transport.

3. HYBRID MODELING OF THE CONTROLLER

Automation of robotic systems raises difficult issues, others than mechanical, sensor data processing and computer architecture ones. The reason is that the behavior results from interacting dynamics both continuous and discrete (events). As a consequence, when dealing with control, a comprehensive model of such hybrid systems has to include both discrete events and continuous aspects. Each of these worlds has a different mathematical framework for its description. Several approaches try to combine these two frameworks into a single one able to describe this mixed discrete/continuous behavior; an overview of several approaches in hybrid systems is given in (Antsaklis *et al.* 1998). Both aspects have not been integrated in this work because it is better to combine two well-established theories, each one being well suited to describe one aspect of the system. So, cooperation and interaction have been established between the PN model of the discrete aspect of the system, such as situations, and the continuous one for the control law aspects. The articulation between the two "worlds" is based on events.

PN provide a formalism which on one hand has a high descriptive power (parallelism, choice, interactions between distributed control entities, incremental design, etc.) and on the other hand has a strong theoretical basis (for analysis purposes for instance). Moreover these models can be directly executed (i.e. without any code translation) by means of an inference engine usually called a "token player". PN have already proved their suitability for modeling robotic applications (Freeman 1991) and have been widely used for analysis and performance evaluation (Simon *et al.* 1994, Medeiros *et al.* 1996, Lima *et al.* 1998) as well as for design and/or execution (Causse *et al.* 1995, Wang *et al.* 1991, Marco *et al.* 1996, Healey *et al.* 1996). In our case, the class of PN used is Hybrid PN with Objects. This class allows the description of both control flow (model's structure) and data flow (objects carried on the model) within a same model.

3.1. *Hybrid Petri nets with objects*

The class of PN used is the Hybrid PN with Objects (H-PNO), defined by *H-PNO=(PNO, F_c, F_e, F_j) with PNO* being the definition of Petri nets with objects. *F* corresponds to different functions associated to the net: F_c continuous (discrete-time) functions associated to places, enabling functions F_e and switching functions (junction functions) F_j associated to transitions (Champagnat *et al.* 1998). The set of places describes the states of the controller, to which can be associated a continuous function to be computed (active states are given by the PN marking). Transitions describe events according to which the controller's state evolves. A transition enabling function defines the condition for the corresponding event to occur (threshold crossing for example). The switching function defines the effects of the event occurrence in terms of continuous variables updating (parameters, state variables, etc.). It has to ensure a smooth switching between two

given continuous functions as control laws for instance. The continuous variables are carried on by attributes of the objects, whose updating is only performed at transition firing time points. All the modules, as well as their inherent interactions, are specified by H-PNO. This homogeneous modeling allows analysis of the whole application.

In order to avoid a cumbersome representation of PN figures, only the structure of the H-PNO is drawn: associated objects, functions, etc. are not shown. Also some branches are described by aggregated transitions (on the control module model) and models are sometimes split into several different ones (on the context manager model).

3.2. *The Control Modules*

A generic model of a control module, specified by HPN with objects, is depicted on Fig. 2. It is organized into several coupled blocks. Block (1) is dedicated to request management: reception and treatment of the objects <request> whose priorities depend on the type of interaction. These interactions between a control module and its upper level (the context manager for instance) are described as asynchronous communication based on the client-server model. A generic control module owns a set of defined services as *program*, *start*, *stop*, *query* and *kill* possible requests (in a master-slave relation, the control module being the slave). The module only treats a *program* request if the latter is available (i.e. in an inactive state). No re- programming of a control module is allowed if it is still executing a control law. Such a request allows to specify parameters values, set points, periodicity, time out reaction, etc. (information carried on the object token <request>). A *query* request is used to know the control module state as well as its parameter values (the status report corresponds to the object token <sr>). If this request is received while performing a control law, it is treated only in the remaining time (i.e. when the control cycle has been executed). *Start*, *stop* and *kill* requests are purely asynchronous without response exchanges; the formers respectively enable and disable the clocks (disabling thus the control law execution). The latter kill correctly the process corresponding to the given real-time task. Whatever the sequence, on which constraints can be specified, all the requests are immediately taken into account (reception), even if their effective treatment can be delayed as it is the case for a *query* request for instance (low priority).

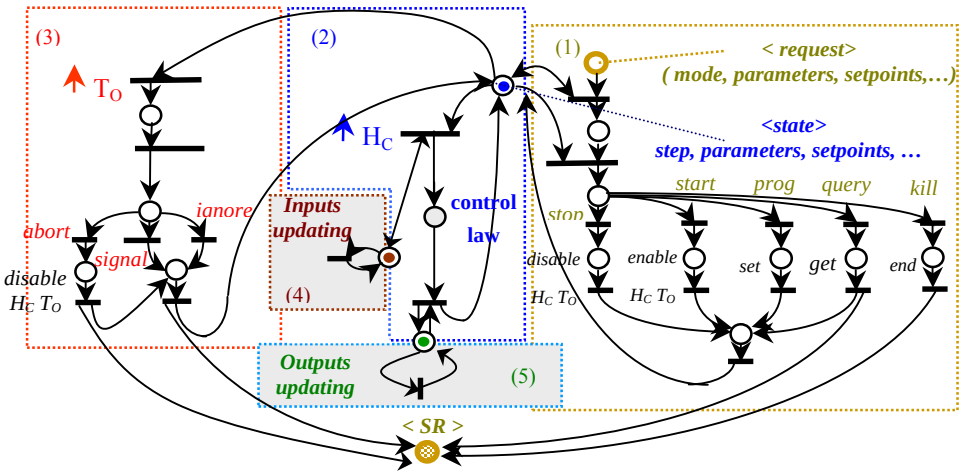


Fig 2: PN based generic model of control modules

Blocks (2) and (3) respectively correspond to the control law execution (periodicity defined by Hc) and the watchdog (periodicity defined by To). Signals To and Hc (time out and clock) are issued from dynamically programmed timers. Notice that the place called "control law" is a place associating the control law to be executed. From an implementation point of view, it is an aggregated representation of the sequence of computation performed by the control law. This sequence, described by short duration steps, can momentarily or definitively be interrupted at any time (between each step). Such interruptions may occur when receiving a request. Blocks (4) and (5) represent external inputs and outputs updating; they do not belong to the control module.

3.3. *The Observation Modules*

The illustrated observation module is the one in charge of singularity detection (SI).

Figure 3: PN based singularity estimator module

It is configured by means of a request whose treatment enables the computation of the associated

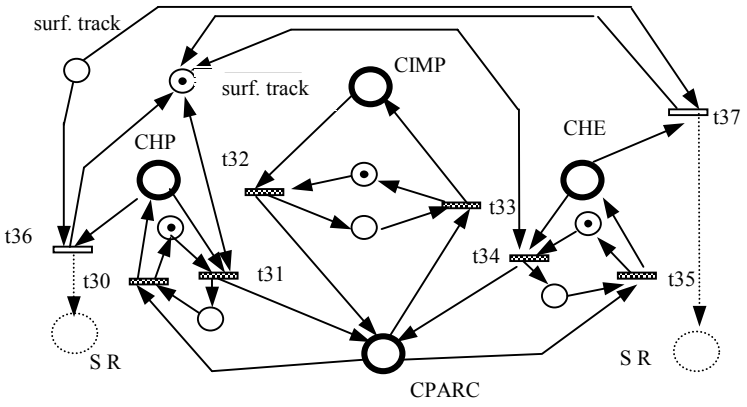
model and set the periodic activation clock (Fig. 3). The event estimation is automatically stopped as soon as the event has been detected.

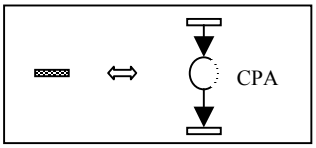
3.4. *The Context Manager*

The context manager receives the goals to achieve from the mission manager. Then it has to send requests to the event detection and control modules and to program, enable/start or disable/stop them according to the current situation. For presentation reasons, the context manager PN model is presented in two parts which have to be merged together to get the overall model (except the interactions with the upper and lower levels). The first part of the model is dedicated to control law switches due to intrinsic singularities (Fig. 4): such commutations imposed on the context may be seen as autonomous jumps. The second one (Fig. 5) represents all the sequencing of control law calls in order to perform the mission goal: these tactical commutations correspond to controlled jumps.

Each thick line place represents an abstract view of its associated control module. If the place is marked by the (unique) object token <objective> (mission goal, point to reach, sub-goals, etc.), the corresponding control module is activated; only one of them can possibly be active at the same time (structural property). All the gray transitions of the singularity management part of the model (transitions t_{30} to t_{35} on Fig. 4) are uncontrollable as their firing correspond to autonomous jumps. All the gray transitions of the tactical sequencing of sub-goals part of the model (transitions t_{38} to t_{43} on Fig. 5) are controllable as their firing correspond to controlled jumps. These gray transitions are equivalent to macro-transitions; their firing implies the execution of an underlying sequence (right schema of Fig. 4-5). The firing of the first transition induces the commutation to the CPA law for stopping the arm. When it gets stopped, the second transition may be fired in order to activate the new selected control module. Activation and desactivation of control modules, and event detection configuration (not represented on the figures) are performed when such transitions are fired. Before to activate the new control module, the junction function associated to the fired transition, is computed to correctly initialise variables of the new control law. For instance consider the switch from CPA to CHP. The intialisation of the CHP integral term (T_i) is done when firing the transition t_{30} according to the given junction function : $T_i = U_c/K_i$, where U_c is the last command vector applied by CPA and K_i the integral gain of CHP. Junction functions, often more complex, are defined for all the possible control law switches, i.e. for all the corresponding transitions.

Some simulation results, corresponding to a commutation from CHP to CPARC (necessarily done through CPA) in order to deal with a singularity, are shown on figure 6. The overall stability is preserved in spite of control law switching.

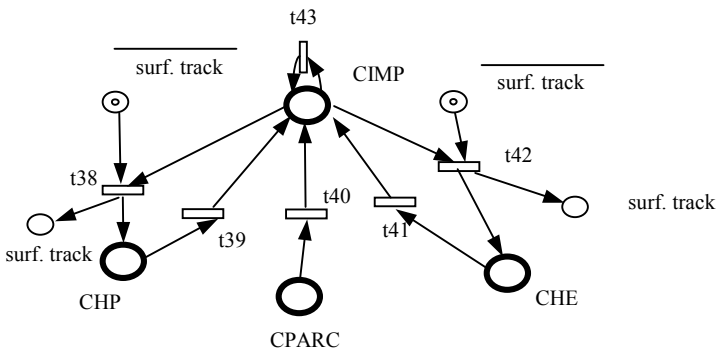


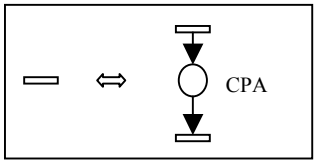


Associated Events

t30, t33, t35 : singularity crossed over.
t31, t32, t34: singularity occurrence.
t36, t37: unrecoverable singularity occurrence.

Fig 4: PN based singularity management (model to be merged with fig.5)





Associated Events

t38, t42 : enabled if contact reached and no palpation goal
t39, t40, t41: enabled if position reached
t43 : enabled if contact reached and palpation goal.

Fig 5: PN based tactical sequencing (model to be merged with fig. 4)

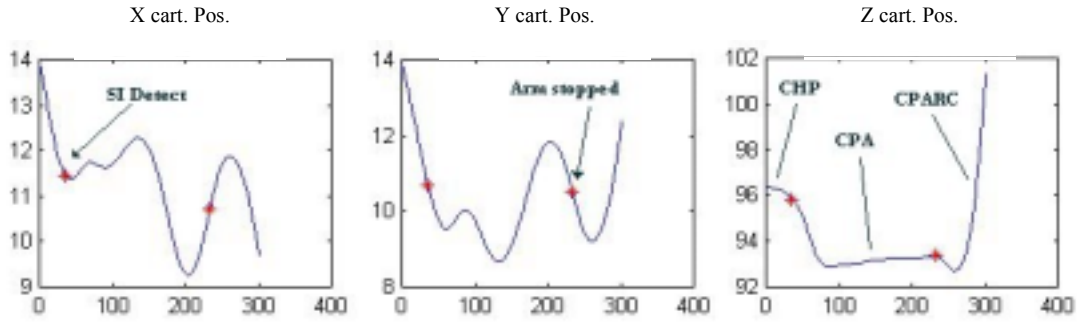


Fig 6: Simulation results of a commutation from CHP to CPARC

4. CONCLUSION

In this paper we have described a new approach to structure an autonomous robot controller based on context management. Events are the key of an architecture composed of synchronous and asynchronous layers. A modeling technique for such hybrid systems has been proposed: it is based on Hybrid Petri Nets with Objects that associate different functions to the node of the net. A real-time "token player" of H-PNO is under development.

5. REFERENCES

Andreu D., J.C. Pascal, R. Valette (1996). Events as a key of a batch process control system. *IEEE CESA'96*, pp. 297-302, Lille - France, July 1996.

Antsaklis P., X. Koutsoukos, J. Zaytoon (1998). On hybrid control of complex systems: a survey. *ADMP'98, European Journal of Automation*, vol. 32, n°9-10, pp. 1023-1045,1998.

Alami R., R. Chatila, S. Fleury, M. Ghallab, F. Ingrand (1998). An architecture for autonomy. *International journal of robotics research*, vol 17, n. 4, pp. 315-337, 1998.

Borrelly JJ., E. Costes-Maniere, B. Espiau, K. Kapellos, R. Pissart Gibollet, D. Simon, N. Turro (1998). The ORCCAD architecture. *International journal of robotics research*, vol 17, n. 4, pp. 338-359, 1998.

Causse O., H.I. Christensen (1995). Hierarchical control design based on Petri net modeling for an autonomous mobile robot. *Intelligent Autonomous Systems*, U. Rembold Eds., IOS Press, 1995

Champagnat R., P. Esteban, H. Pingaud, R. Valette (1998). Modeling and simulation of hybrid systems through Pr/Tr PN-DAE Model. *ADMP'98*, pp. 131-137, Reims - France, March 1998.

Degoulange E., P. Dauchez, F. Pierrot (1993). *Determination of a force control law for an industrial robot in contact with a rigid environment.* IEEE SMC'93, vol 2, pp. 270-275, Le Touquet - France, October 1993.

Fleury S., M; Herrb, R. Chatila (1994). Design of a modular architecture for autonomous robot. *IEEE ICRA'94*, San Diego - USA, May 1994.

Freedman P. (1994). Time, Petri nets and robotics. *IEEE Trans. on Robotics and Automation*, vol. 7, n°4, pp. 47-433, August 1991.

Healey A.J., D.B. Marco, P. Oliviera, A. Pascoal, V. Silva, C. Silvestre (1996). Strategic level mission control - An evaluation of CORAL and PROLOG implementations for mission control specifications. *IARP'96, workshop on underwater robotics*, Toulon-France, March 1996.

Leleve A., P. Dauchez, P. Fraisse, F. Pierrot (2000). An enhanced Mobile Manipulator. *WAC'2000*, Hawaii-USA, June 2000.

Lima P., H. Gracio, V. Veiga, A. Karlsson (1998). Petri nets for Modeling and Coordination of Robotic Tasks. *IEEE SMC*, pp. 190-195, San Diego - USA, October 1998.

Marco D.B., A.J. Healey, R.B. McGhee (1996). Autonomous underwater vehicles: hybrid control of mission and motion. *Autonomous Robots 3*, pp. 169-186, Kluwer Academic Publishers, 1996.

Medeiros A.D., R. chatila, S. Fleury (1996). Specification and validation of a control architecture for autonomous mobile robots. *IEEE IRS'96*, pp.162-169, Osaka - Japan , Nov. 1996.

Santos V.M., J.P. Castro, M.I. Ribeiro (2000). A nested-loop architecture for mobile robot navigation. *Int. journal of robotics research*, vol 19, n. 12, pp. 1218-1235, 2000.

Simon D., P. Freedman, E. Castillo (1994). Analyzing the temporal behavior of real-time closed-loop robotic tasks. *IEEE ICRA'94*, pp. 841-847, San Diego - USA, 1994.

Simon D., K. Kapellos, B. Espiau (1998) Formalization of hybrid structures in robot controllers : the ORCCAD approach. *INCOM'98*, pp. 87-92, Nancy-Metz - France, June 1998.

Wang F.Y., K.J. Kyriakopoulos (1991). A Petri-net coordination model for an intelligent mobile robot. *IEEE SMC*, vol. 21, n° 4, July 1991.