EXECUTION CONTROL OF ROBOTIC TASKS FOR MARINE SYSTEMS

Massimo Caccia, Gabriele Bruzzone *

* *CNR-ISSIA Sez. di Genova*
Via De Marini 6
16149 Genova
Italy

Abstract: The problem of controlling the execution of navigation, guidance and control tasks of a mobile robot, reconfiguring their states according to the demands of the human operator or automatic coordinator, is faced in this paper focusing on the design and implementation of a Petri net based execution controller for marine robots*Copyright*© *2005 IFAC.*

Keywords: Petri-nets, discrete-event systems, marine systems, supervision, architectures.

1. INTRODUCTION

The concept of *execution control* as interface between the asynchronous, event-driven, decision making and planning levels and the synchronous, continuous time, functional execution level was introduced and discussed in the work carried out by CNRS-LAAS on architectures for autonomous mobile robots (Alami *et al.*, 1998), where the role of the *execution controller* in handling the conflicts between different functional modules and maintaining a logical description of their operating states was pointed out. In particular, this layer guarantees the safety of the system by checking the commands sent to the functional level preventing it from entering in an unconsistent condition with respect to a model of desirable or undesirable states (Ingrand and Py, 2002). The execution control level is synchronous with the underlying functional modules, in the sense that it processes all the commands sent to and reports coming back from them, and acts in guaranteed real-time. The need for the pilot of an advanced ROV of reconfiguring the set of active motion estimation and control tasks according to the current goal and environmental conditions motivated the research on execution control in the field of underwater robotics (Coletta *et al.*, 2001). Indeed, when the system is complex, it is impossible for the human operator remembering and executing in a few tenths of seconds relatively long sequences of task activation and deactivation commands in order to switch from different sets of active tasks maintaining the system consistent. On the other hand, embedding all the task dependencies and conflicts, and the consequent sequences of task activation and deactivation commands when switching between different operating contexts, in the software is not reliable as soon as the number of tasks increases. Research in the field of ROVs focused on the definition of a set of rules, governing the behaviour of the execution control level, which can be derived by the analysis of the hierarchical I/O relationships between tasks and variables (the so called *task-variable graph*), independently from the semantics of the specific tasks. Since these rules can be easily translated in constraints on the state of a generic finite state machine representing the execution level, the notable results of (Yamalidou *et al.*, 1996) in automatically generating a controlling net from the marking of the original one, and the well-known capabilities of Petri nets in modelling concurrency, parallelism and resource contention, Petri nets were used to represent the execution level as a discrete event system. Results, including suitable Petri net search algorithm for system reconfiguration, are summarised in (Bruzzone *et al.*, 2003), while an examples of the use of Petri net for mission control of marine systems is reported in (Oliveira *et al.*, 1998). This research pointed out strong differences in controlling the execution of estimate and control tasks. In particular, the fundamental requirement of a monitoring-oriented sensing and perception architecture, activating as more as sensors and estimators is possible and making available all the data processing results, makes the part of the execution controller concerning the motion estimation tasks basically a free evoluting system, mainly conditioned by the uncontrollable results of estimator inizializations. On the other hand activation and deactivation of control tasks can be executed instantaneously without any initialization phase. This motivated the separation of the execution control of motion estimation and control tasks, producing, as a consequence, a more effective capability in managing the switch between different configurations of the guidance and control system executing the same high level function.

2. EXECUTION LEVEL

The Execution level embeds a set $\mathcal{T}$ of elementary tasks, i.e., software components capable of performing specific motion estimation and control functions, which communicate through variables, i.e., shared memory used for task I/O. According to their semantics, the set $\mathcal{V}$ of variables can be divided in the distinct subsets of estimation variables $\mathcal{EV}$, containing the values measured by sensors as well as the outputs of the filtering algorithms, and control variables $\mathcal{CV}$, representing the references to be tracked by the control tasks. For each task $t \in \mathcal{T}$, the symbols $\mathcal{I}(t)$ and $\mathcal{O}(t)$ denote the sets of input and output variables of the task, and the symbols $\mathcal{CI}(t)$, $\mathcal{CO}(t)$, $\mathcal{EI}(t)$, and $\mathcal{EO}(t)$ indicate the control input variables, control output variables, estimation input variables, and estimation output variables, respectively. Moreover, the symbol $\mathcal{CT}$ denotes the set of control tasks, i.e., $\mathcal{CT} \triangleq \{t \in \mathcal{T} : \mathcal{EO}(t) = \emptyset\}$, and $\mathcal{ET}$ represents the set of estimation tasks, i.e., $\mathcal{ET} \triangleq \{t \in \mathcal{T} : \mathcal{EO}(t) \neq \emptyset\}$.

2.1 DES task representation

From the point of view of its execution state, a generic task can be *running* (**R**) or *idle* (**I**), and the state transitions are caused by the operations of *activating* (**A**) or *deactivating* (**D**) the task. When the activation of a task may fail, as, for instance, in the case of estimation tasks which may sometimes converge too slowly (or not converging at all), the *init* (In) state has to be introduced to complete the task description. Depending on the conclusion of the initialization phase, the *success* (**S**) and *fail* (**F**) transitions lead the task in the running and idle state respectively. It is worth noting that these transitions are uncontrollable (their triggering is function of the perceived data), but observable. Petri net representation of tasks is shown in Figure 1. Denoting with $x(X)$ the

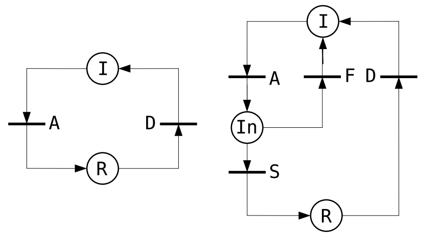

Fig. 1. Petri Net representation of tasks.

number of tokens in place X, the place invariants $x(R) + x(I) = 1$ and $x(R) + x(I) + x(In) = 1$ hold in the case of not initialized and initialized tasks respectively.

2.2 Properties and rules

The execution of complex missions requires that the connections between tasks are dynamically established according to mission events and requirements. Each task does not a priori know which tasks will produce/consume its input/output variables, and suitable task activation and deactivation operations determine these connections at any time instant. A set of run-time constraints, linking the task I/O relationships to the structure of the control and motion estimation architecture, enable the verification of the correctness of any task configuration, guaranteeing that fundamental properties of data consistency, *from bottom upward activation* and hierarchical structure of the control leg are complied. In particular, data consistency is guaranteed by the property of *no concurrent writing* stating that there cannot be two or more running tasks sharing a common output variable

$$\forall v \in \mathcal{V}, \sum_{t \in \mathcal{T} : v \in \mathcal{O}(t)} x(R_t) \leq 1 \qquad (1)$$

The assumption that this property holds at any time instant is the basic rule controlling the behaviour of the guidance and control system, while, due to the need of distinguishing each estimate variable for system monitoring purposes, it gets a *design constraint* in the case of the motion estimation system. The *from bottom upward activation* of the execution level is guaranteed by the rules of *complete writing of consumed estimation variables* (2) and *complete tracking of written control variables* (3), stating that, at any time instant, each used estimate must be generated by a running task and all the computed control values must be tracked.

$$\begin{aligned} &\forall v \in \mathcal{EV}, \forall t \in \mathcal{T} : v \in \mathcal{EI}(t) \\ &x(I_t) + \sum_{\tau \in \mathcal{T} : v \in \mathcal{EO}(\tau)} x(R_\tau) \geq 1 \end{aligned} \tag{2}$$

$$\begin{aligned} &\forall v \in \mathcal{CV},\ \forall t \in \mathcal{CT} : v \in \mathcal{CO}(t) \\ &x(I_t) + \sum_{\tau \in \mathcal{T} : v \in \mathcal{CI}(\tau)} x(R_\tau) \geq 1 \end{aligned} \tag{3}$$

In addition to rule (3), the hierarchical nature of the control architecture is established by the *no concurrent tracking* property, which guarantees at any time the uniqueness of the control strategy:

$$\forall v \in \mathcal{CV}, \sum_{t \in \mathcal{CT} : v \in \mathcal{CI}(t)} x(R_t) \leq 1 \tag{4}$$

It is worth noting that rules (1) and (4) represent mutual exclusion constraints between tasks, while rules (2) and (3) express the dependency of task t from the set of tasks τ.

3. EXECUTION CONTROL

As shown in section 2.2 the execution of the estimate tasks is controlled only by the design constraint (1) and rule (2), while control tasks, as well as *switch between estimation variables* (Bruzzone *et al.*, 2003), do not involve any initialization phase. In addition, the requirement of having, at any time, a representation of the robot and environment state as wider as possible forces the execution controller to automatically activate any estimate task as soon as it is possible, while control tasks must be activated only to satisfy specific requests of the decision layer. Thus, the different behaviour in the management of control and estimate tasks suggested the implementation of a *Motion Estimation Execution Controller* (MEEC) supervising the free evolution, i.e. complete activation, of the sensing and perception system and executing external commands establishing some task has to be idle, and of a *Guidance and Control Execution Controller* (GCEC), managing, through Petri net control and reconfiguration, mission control commands and sensing failure events. The Petri net based GCEC will be discussed in the following.

3.1 Controlling net generation

Since rules (1)-(4) on the behavior of the execution level are expressed as predicates on the Petri net marking, the Petri net can be modified such that it enforces the requested predicates. According the approach proposed and discussed in (Yamalidou *et al.*, 1996), this can be done adding a controlling net, obtaining a resulting net of the form (the classical matrix notation for Petri nets is used):

$$\begin{bmatrix} \underline{x} \\ \underline{x}_c \end{bmatrix}_{k+1} = \begin{bmatrix} \underline{x} \\ \underline{x}_c \end{bmatrix}_{k} + \begin{bmatrix} D \\ D_c \end{bmatrix} \underline{f}_k \ , \tag{5}$$

where the vectors $\underline{x}$ and D denote the state and the transition matrix of the *original net*, while $\underline{x}_c$ and D_c denote the state and the transition matrix of the *controlling net*, respectively. Details on the application of the above-mentioned approach to the an execution level structure including estimate tasks too can be found in (Bruzzone *et al.*, 2002). Here, it is worth noting that information about the conflicts and dependencies among the tasks is completely embedded by the controlling net. In order to simplify the reconfiguration of the net, it is quite important to minimise the size of the controlling net, i.e. the number of introduced constraints on the marking of the net. This can be done on the basis of some simple observations.

(1) Given two sets of constraints $\sum_{t \in A} x_t \leq 1$ and $\sum_{t \in B} x_t \leq 1$ such that $B \subseteq A$ then the constraint on places B is redundant. This can be, for instance, the case of mutex constraints between tasks originated by the application of rules (1) and (4) to different variables.
(2) Defined a set of consumers $\mathcal{C}$ and a set of producers $\mathcal{P}$ of a resource r, if any consumer is active then at least one producer must be active, i.e.

$$\sum_{p \in \mathcal{P}} x\left(R_p\right) \geq \min\left\{1, \sum_{c \in \mathcal{C}} x\left(R_c\right)\right\} \tag{6}$$

In the case $\sum_{c \in \mathcal{C}} x\left(R_c\right) \leq 1$ the constraint (6) can be written as

$$\sum_{c \in \mathcal{C}} x\left(I_c\right) + \sum_{p \in \mathcal{P}} x\left(R_p\right) \geq card\left(\mathcal{C}\right) \tag{7}$$

and if $\sum_{p \in \mathcal{P}} x\left(R_p\right) \leq 1$ the constraint on the mutual exclusion of the consumers is redundant. Considering the sets of producers $\{\tau \in \mathcal{T} : v \in \mathcal{CI}(\tau)\}$ and of consumers $\{\tau \in \mathcal{T} : v \in \mathcal{CO}(\tau)\}$, this property can be immediately used to group multiple instances of task dependency constraints originated by applying rule (3) to control variables generated by more than one task. The fact that the number of running tasks for each set is ≤ 1 is guaranteed by rules (1) and (4). As far as rule (2) is concerned, the set of producers $\{\tau \in \mathcal{T} : v \in \mathcal{EO}(\tau)\}$ is guaranteed to be of cardinality one by the design constraint (1), while constraints can be grouped only for the subsets $\mathcal{M}$ of the consumers $\{\tau \in \mathcal{T} : v \in \mathcal{EI}(\tau)\}$ such that $\sum_{m \in \mathcal{M}} x(R_m) \leq 1$.

(3) Given two multiple dependency constraints of the type consumer-producer denoted by subscripts a and b respectively, if $\mathcal{C}_a \subset \mathcal{C}_b$ and $\mathcal{P}_b \subset \mathcal{P}_a$ then constraint a is redundant.

3.2 Petri net reconfiguration

The execution control module is triggered by events, i.e. external commands on the desired state of some tasks. A goal is defined as the desired presence of a token in a subset of places of the *original net*. Denoting with $\mathcal{G}$ the set of the indexes of the desired places, the corresponding goal vector $\underline{x}^* : x^*_g = \begin{cases} 1, \ g \in \mathcal{G} \\ 0, \ g \notin \mathcal{G} \end{cases}$ is satisfied by any marking of the net $\underline{x}$ such that

$$\begin{cases} \underline{x} . \wedge \underline{x}^* = \underline{x}^* \\ \underline{x} \geq \underline{0} \end{cases} \tag{8}$$

where $.\wedge$ is the element-wise AND operator and the condition $\underline{x} \geq \underline{0}$ represents the admissibility, i.e. non negative marking, of the Petri net configuration $\underline{x}$. Unlike the case discussed in (Caccia *et al.*, 2001), the reduction of the problem to controlling the execution of guidance and control tasks, which have no initialization phase, allows the assumption that all the transitions are instantaneous and can be fired simultaneously. This simplifies the work of the execution controller, that, when a goal $\underline{x}_g$ is established, has to find a firing vector $\underline{f}_g$ such that $\underline{x}_0 + D\underline{f}_g = \underline{x}_g$, being $\underline{x}_0$ the actual state of the net, without taking any care of the consistency of the net with respect to the transition firing order. Thus, a simple search algorithm based on the *boolean* backward spreading of the utility throgh Petri net transitions and controlling places is proposed. At first, given a Petri net with state $\underline{x}_0$, all the predecessor transitions of the marked places of the *original net* are inhibited from firing (*state-inhibition*). Then the algorithm is initialized by constructing a set of useful transitions consisting of the predecessors of the goal places. It is worth noting that, since the goal places are in the *original net* their correspondence with the predecessor transitions is biunique. At the generic step, given a set of useful transitions $\mathcal{T}_u$ and the associated firing vector $\underline{f}(\mathcal{T}_u) : \underline{f}_t = \begin{cases} 1, \ t \in \mathcal{T}_u \\ 0, \ t \notin \mathcal{T}_u \end{cases}$ leading the net to the state $\underline{x} = \underline{x}_0 + D\underline{f}(\mathcal{T}_u)$, a set of useful controlling places is defined as $\mathcal{P}_u = \{p : x_p < 0, \ p > n_r\}$, where n_r denotes the size of the original net. In practice, the empty places, where the presence of a token is needed to allow the firing of a set of transitions $\mathcal{T}_u$, are considered useful for $\mathcal{T}_u$. Of course, if $\underline{x}_0 + D\underline{f}(\mathcal{T}_u) \geq \underline{0}$ then $\mathcal{T}_u$ is an admissible solution. If transitions can be seen as a logical *AND* in the backward propagation of utility, each useful place $p \in \mathcal{P}_u$ behaves as logical *OR* between its n_p not state-inhibited predecessor transitions $\mathcal{T}_u(p) = \{t_{p_1} \ldots t_{p_{n(p)}}\}$. Thus, the set of useful places $\mathcal{P}_u$ generates all the possible combinations of useful transitions $\mathcal{T}^{new}_u = \{t_1, \ldots, t_m\}$ with $t_1 \in \mathcal{T}_u(p_1), \ldots, t_m \in \mathcal{T}_u(p_m)$ and $m = card(\mathcal{P}_u)$. Each combination of useful transitions is then applied to the Petri net iterating the algorithm. If a combination of useful transitions $\mathcal{T}^{new}_u$ is equal to the previous set $\mathcal{T}_u$, this means that the backward propagation of utility has encountered a deadlock condition.Since the above-proposed algorithms allows to find all the possible solutions, i.e. sets of firing transitions satisfying the goal, that is compatible with the typical complexity of guidance and control systems of marine robots and the state-of-the-art computing power, the problem is to define some suitable critera to select the, in some sense, *optimal* solution. As shown in section 4, the basic criterium of minimising the number of fired transitions can force the deactivation of high level guidance functions which could be executed by alternative set of tasks involving, to be consistently activated, the firing of a higher number of transitions. The execution controller has to take into account, in some way, the level of the tasks in the guidance and control hierarchy. Defining a *task chain of order* n an ordered list of tasks $\mathcal{C}_n = \{\tau_n, \ldots, \tau_1\}$ such that $\forall j \in [1, \ n-1], \exists v \in \mathcal{CV} : v \in \mathcal{CO}(\tau_{j+1}) \wedge v \in \mathcal{CI}(\tau_j)$, a task τ is of order N, when N is the maximum order of the task chains led by τ. The order of a solution is defined as the maximum order of the tasks whose state is modified by its transitions. Thus, among the solutions of minimum order, the one constituted by the minimum number of transitions is selected.

4. EXPERIMENTAL RESULTS

The above described methodology for controlling the execution of a robot execution level represented through Petri nets, which is the natural evolution of a system already implemented in the control system of the Romeo ROV, is part of the *Free Robot Architecture* project, currently under development at CNR-ISSIA Sez. di Genova, representing the temptative of developing an architecture for the rapid prototyping, design, implementation and running of robotic control systems completely based on free operating systems and software tools. In this framework a *gcpetrinetgenerator*, generating a controlled Petri net from a description of the guidance and control system based on a task-variable graph, and a *gcexecutioncontroller*, controlling the execution of the guidance and control tasks, have been implemented in the Linux 2.6.6 operating system using the GNU C++ 3.2.3 compiler and run on a 3.06 GHz laptop. In the following an example of managing the guidance and control tasks of a marine robot on the horizontal plane is examined, pointing out the system capabilities in dealing with alternative possibilities of executing high order guidance functions. The I/O relationships between the guidance and control tasks and variables of the steering system of a marine vehicle are shown in the task-variable graph of Figure 2, where x, psi, r and Tr denote position, heading, yaw rate and yaw torque respectively, and the $*$s indicate the reference values. In particular, Figure 2 represents different

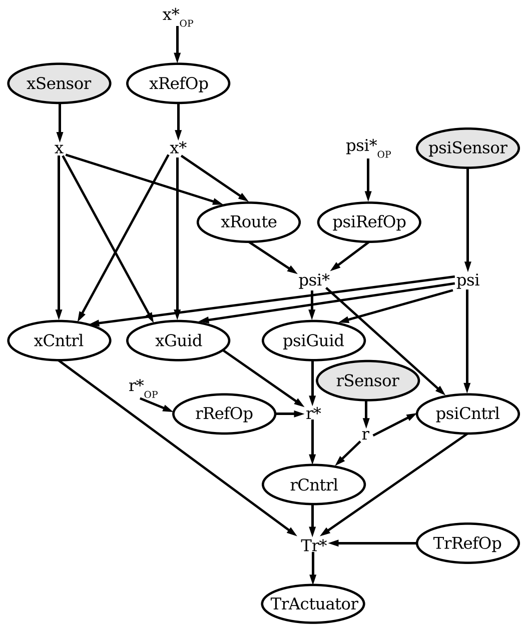

Fig. 2. Task-Variable Graph representation of the execution level.

guidance and control strategies of steering a vehicle to reach a desired position x^*_{OP}, assuming that the vehicle surge speed is controlled by some other module. Mutual exclusion constraints guaranteeing the consistency of the data and the uniqueness of the adopted control strategy are visible by the reader at a glance, and are summarised in table 1 as computed by *gcpetrinetgenerator*. In this case there are no redundant mutual exclusion constraints. The constraints, computed by *gcpetrinetgenerator* by applying the compact *consumer-producer* form of rules (2) and (3) discussed in section 3.1, observation (2), are reported in table

Table 1. Mutual exclusion constraints

R(1): no concurrent writing	
conflicting tasks	*on variable*
TrRefOp xCntrl psiCntrl rCntrl	TrRef
rRefOp xGuid psiGuid	rRef
psiRefOp xRoute	psiRef
R(4): no concurrent tracking	
conflicting tasks	*on variable*
xRoute xGuid xCntrl	xRef
psiGuid psiCntrl	psiRef
Rule (1) and (4) MUTEX constraints	
TrRefOp xCntrl psiCntrl rCntrl	
rRefOp xGuid psiGuid	
psiRefOp xRoute	
xRoute xGuid xCntrl	
psiGuid psiCntrl	

Table 2. Task dependence constraints.

R(2): complete writing of consumed estimates		
on variable	*consumer (Idle)*	*producer (Run)*
r	psiCntrl rCntrl	rSensor
x	xRoute xGuid xCntrl	xSensor
psi	xCntrl psiCntrl	psiSensor
psi	xGuid psiGuid	psiSensor
psi	xGuid xCntrl	psiSensor
psi	psiGuid psiCntrl	psiSensor
R(3): complete tracking of written controls		
on variable	*consumer (Idle)*	*producer (Run)*
TrRef	TrRefOp xCntrl psiCntrl rCntrl	TrActuator
rRef	rRefOp xGuid psiGuid	rCntrl
xRef	xRefOp	xRoute xGuid xCntrl
psiRef	psiRefOp xRoute	psiGuid psiCntrl
Redundant MUTEX constraints of consumers		
TrRefOp xCntrl psiCntrl rCntrl		
rRefOp xGuid psiGuid		
psiRefOp xRoute		
xRoute xGuid xCntrl		
psiGuid psiCntrl		

Table 3. Constraint summary.

Rule (1) and (4) MUTEX constraints	
—	
Rule (2) and (3) dependence constraints	
consumer (Idle)	*producer (Running)*
psiCntrl rCntrl	rSensor
xRoute xGuid xCntrl	xSensor
xCntrl psiCntrl	psiSensor
xGuid psiGuid	psiSensor
xGuid xCntrl	psiSensor
psiGuid psiCntrl	psiSensor
TrRefOp xCntrl psiCntrl rCntrl	TrActuator
rRefOp xGuid psiGuid	rCntrl
xRefOp	xRoute xGuid xCntrl
psiRefOp xRoute	psiGuid psiCntrl

2. In particular, the constraints on the variable *psi* are grouped according the subsets of the above-computed MUTEX constraints. It is worth noting how the constraints on the mutual exclusion are redundant, reducing the constraints originated by the application of rules (1) to (4) to those reported in table 3. Once built the controlled Petri net,

Table 4. *xRefOp* task activation.

Goal: xRefOp RUNNING		solutions: 4
Solution 1:	order 4,	size 8
	xSensor	ACTIVATE
	psiSensor	ACTIVATE
	rSensor	ACTIVATE
	xRefOp	ACTIVATE
	xRoute	ACTIVATE
	psiGuid	ACTIVATE
	rCntrl	ACTIVATE
	TrActuator	ACTIVATE
Solution 2:	order 4,	size 5
	xSensor	ACTIVATE
	psiSensor	ACTIVATE
	xRefOp	ACTIVATE
	xCntrl	ACTIVATE
	TrActuator	ACTIVATE
Solution 3:	order 4,	size 7
	xSensor	ACTIVATE
	psiSensor	ACTIVATE
	rSensor	ACTIVATE
	xRefOp	ACTIVATE
	xRoute	ACTIVATE
	psiCntrl	ACTIVATE
	TrActuator	ACTIVATE
Solution 4:	order 4,	size 7
	xSensor	ACTIVATE
	psiSensor	ACTIVATE
	rSensor	ACTIVATE
	xRefOp	ACTIVATE
	xGuid	ACTIVATE
	rCntrl	ACTIVATE
	TrActuator	ACTIVATE
Selected solution: 2		*elapsed time: 7.146 ms*

Running Tasks
xSensor psiSensor xRefOp xCntrl TrActuator

the execution controller can process the operator commands according to the rules defined in 3.2. At first the *gcexecutioncontroller* is commanded to activate the tracking of the desired position, i.e. to put the *xRefOp* task in the *running* state, starting from a complete *idle* state. As shown in table 4, the execution controller individuated all the possible ways of executing the desired command, and selected the one of minimum size. Indeed, all the computed solutions were of order 4, imposed by firing a transition of task *xRefOp*. Once applied the selected solution, the execution controller is asked to deactivate the *xCntrl* task. In this case, as shown in table 5, the solution with the minimum number of firing transitions would deactivate any position tracking function. This system capability is maintained active thanks to the *minimum order* criterium, introduced at the end of section 3.2, which forces the selection of *solution 4* involving the swith between position control and guidance. The capability of handling the switch between different control strategies without inserting any switch transition explicitely in the Petri net is the main advance of this research with respect to previous work summarised in (Bruzzone *et al.*, 2003).

Table 5. *xCntrl* task deactivation.

Goal: xCntrl IDLE		solutions: 4
Solution 1:	order 3,	size 5
	rSensor	ACTIVATE
	xRoute	ACTIVATE
	xCntrl	DEACTIVATE
	psiGuid	ACTIVATE
	rCntrl	ACTIVATE
Solution 2:	order 4,	size 2
	xRefOp	DEACTIVATE
	xCntrl	DEACTIVATE
Solution 3:	order 3,	size 4
	rSensor	ACTIVATE
	xRoute	ACTIVATE
	xCntrl	DEACTIVATE
	psiCntrl	ACTIVATE
Solution 4:	order 2,	size 4
	rSensor	ACTIVATE
	xGuid	ACTIVATE
	xCntrl	DEACTIVATE
	rCntrl	ACTIVATE
Selected solution: 4		*elapsed time: 6.045 ms*

Running Tasks
xSensor psiSensor rSensor xRefOp xGuid rCntrl TrActuator

REFERENCES

Alami, R., R. Chatila, S. Fleury, M. Ghallab and F. Ingrand (1998). An architecture for autonomy. *International Journal of Robotic Research* **17**(4), 315–337.

Bruzzone, Ga., M. Caccia, P. Coletta and G. Veruggio (2002). A reconfigurable control architecture for mobile robots. In: *Proc. of MCCA 2002*. Lisboa, Portugal.

Bruzzone, Ga., M. Caccia, P. Coletta and G. Veruggio (2003). Execution control and reconfiguration of navigation, guidance and control tasks for UUVs. In: *Proc. of MCMC 2003*. Girona, Spain. pp. 137–142.

Caccia, M., P. Coletta, G. Bruzzone and G. Veruggio (2001). Petri net-based execution control of robotic tasks. In: *Proc. of MCCA 2001*. Dubrovnik, Croatia.

Coletta, P., R. Bono, G. Bruzzone, M. Caccia and G. Veruggio (2001). Execution control of the NGC tasks for ROVs. In: *Proc. of 2001 IEEE International Conference on Robotics and Automation*. pp. 2369–2374.

Ingrand, F. and F. Py (2002). An execution control system for autonomous robots. In: *Proc. of ICRA '02*. Vol. 2. pp. 1333–1338.

Oliveira, P., A. Pascoal, V. Silva and C. Silvestre (1998). The mission control system of MARIUS AUV: System design, implementation, and tests at sea. *Int. J. on Sys. Science - Special Issue on Underwater Robotics* **29**(10), 1065–1080.

Yamalidou, K., J. Moody, J. Lemmon and P.J. Antsaklis (1996). Feedback control of Petri nets based on place invariants. *Automatica* **32**(1), 15–28.