Copyright © 2002 IFAC
15th Triennial World Congress, Barcelona, Spain

EXECUTION LEVEL CONTROL AND RECONFIGURATION FOR REMOTELY OPERATED VEHICLES

G. Bruzzone, M. Caccia, P. Coletta, and G. Veruggio

Consiglio Nazionale delle Ricerche - Istituto Automazione Navale
Via De Marini, 6 - 16149 Genova – Italy
{gabry,max,paolo,gian}@ian.ge.cnr.it

Abstract: A method to design the control architecture for Remotely Operated Vehicles is presented that is based on the idea to use the Petri net formalism. The “safe” behaviour of the architecture is guaranteed by enforcing some place invariants on the marking of the net. An algorithm to find out feasible sequences of operations to switch between configurations is introduced. Finally, a method to automatically reconfigure the system is shown. *Copyright © 2002 IFAC*

Keywords: Petri-nets, control system design, autonomous mobile robots.

1. INTRODUCTION

The increasing demand of highly sophisticated controllers able to achieve high performances in uncertain and adverse conditions has led to the development of intelligent control systems, characterized by three level hierarchical functional architectures (Antsaklis and Passino, 1993; Valavanis and Saridis, 1992). Asynchronous decisions about the proper tasks required to accomplish the current mission are made by the upper organization and coordination levels, while the execution level embeds a library of synchronous/continuous-state functions with real-time motion estimation and control abilities. Thus, an interface is required, which represents the underlying execution level as a discrete event system, generating events from the continuous state domain to the discrete-state domain and mapping symbols in the opposite versus. The interface guarantees the correct behaviour of the execution level, checking that no forbidden state is reached and that the proper task activation and deactivation order is respected. A thorough discussion on architectures for hybrid control and autonomous systems can be found in (Fierro and Lewis, 1997) and (Alami, *et al.*, 1998). In this framework, the research presented in the following focuses on the control of the execution of synchronous/continuous-time functions in the case of (semi-)autonomous tele-operated robots acting in poorly structured environments, e.g., underwater vehicles. According to the tele-operation paradigm (Sheridan, 1989), the need for the human operator of interacting with the execution activities at various levels, i.e. setting the desired force/torque, speed or position variables in order to cope with different operating conditions, motivated the design of hierarchical motion estimation and control architectures which can be activated from bottom upwards. The basic properties of "from bottom upwards activation" and "data consistency" determine a set of constraints on the task activation/deactivation sequences which are related to the topological structure of the execution level, i.e. the graph representing the I/O relationships between the tasks, rather than the semantics of each task. It is worth noting that, in the following, control architectures including integral controllers will be considered. In this case, the property that control tasks can get active only if their output variables are actually applied to the robot shall be verified too.

A method to design an execution control module that deals with these constraints has been already proposed in (Caccia, *et al.*, 2001), and it is based on the Petri net formalism. In the present work, that method is furtherly investigated in two ways. First, a backward search algorithm is proposed, that can be used to find a suitable sequence of transition firings that can satisfy a user request. Roughly speaking, when a command to activate or deactivate a task is received, this algorithms looks for the correct sequence of operations such that the request can be fulfilled. Second, a method to monitor the behaviour of the execution level, based on the continuous analysis of the Petri net status, is presented; this method allows one to embed in the execution control module the ability to automatically activate the estimation leg of the control architecture, and to reconfigure the system in order to maintain the best possible configuration.

The paper is organised as it follows. Conventional intelligent control architectures are summarised in Section 2, discussing the role played by the execution control module the paper deals with. Section 3 summarizes the results from the paper (Caccia, *et al.*, 2001), while Section 4 and 5 describe the extensions that are the main topic of this paper, i.e., the

backward search algorithm and the on-line monitoring. Simulation results are depicted in Section 6; finally, Section 7 gives some concluding remarks.

2. SYSTEM ARCHITECTURE

An example of a conventional intelligent control architecture is sketched in Figure 1.

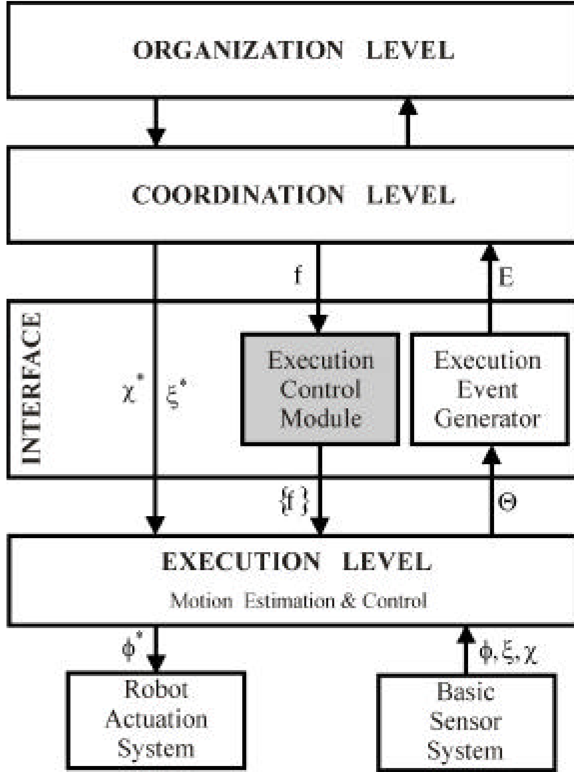

Fig. 1. Intelligent control architecture.

The actuation and sensor systems provide the execution level with a set of logical propulsion and sensing modules embedding the interfaces with physical devices. In the scope of this research the execution level (EL) relies on a hierarchical control system handling the robot's kinematics, both absolute and environment-based, and dynamics (speed control). The operational variables χ, that describe the kinematics task functions, and robot's velocities ξ are estimated by suitable motion estimation modules which process basic internal and external sensor measurements. Events E, signalling particular interactions of the robot with the operating environment and the status of advancement of motion estimation and control tasks, are generated by the execution event generator which monitors the tasks' state and performances Θ. On this basis, the coordination level dynamically schedules the motion control and estimation tasks in order to reach the desired goal. The execution control module, dealt with in this paper, checks if the commanded activation/deactivation f of the EL tasks can be executed. In the case that this leads to a forbidden state, the execution control determines a suitable sequence of commands {f} which enable the execution of the desired one.

3. PETRI NET-BASED DESIGN OF THE EXECUTION CONTROL MODULE

This section briefly recalls the results from a previous work (Caccia, *et al.*, 2001). The main idea is to identify the basic components of the execution module and to represent them by means of the Petri net formalism. Then, a set of rules on the behaviour of the execution level is stated, and these rules are expressed as predicates on the Petri net marking. Finally, results from the Petri net theory are used to modify the Petri net such that it enforces the requested predicates. The execution level embeds a set 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, variables can be classified in the distinct sets of estimation and control variables. The estimation variables contain the values measured by sensors as well as the outputs of the filtering algorithms, while the control variables represent the references to be tracked by the control tasks.

2.1. Nomenclature and execution level properties.

The following symbols will be used throughout the paper to refer to tasks, variables, and related items.

V: set of variables
EV: set of estimation variables, i.e. sensor data and estimates
CV: set of control variables, i.e. reference values
T: set of tasks
EI(t): set of estimation input variables of task t
CI(t): set of control input variables of task t
I(t): set of input variables of task t
EO(t): set of estimation output variables of task t
CO(t): set of control output variables of task t
O(t): set of output variables of task t
CT: set of control tasks = $\{t \in T : EO(t) \equiv 0\}$
ET: set of estimation tasks = $\{t \in T : EO(t) \neq 0\}$
ECT(t): set of equivalent control tasks to task t = $t \cup \{\mathbf{t} \in CT : I(\mathbf{t}) \equiv I(t) \wedge O(\mathbf{t}) \equiv O(t)\}$
EST($\underline{v}$): set of estimate $\underline{v}$ DEMUX tasks = $\{t \in ET : O(t) \equiv \underline{v}\}$

The definition of *ECT*, i.e., the observation that some control tasks can share the same set of input and output variables (see Figure 2), lets one to add the ability to switch among these tasks. This is of particular importance, since it allows to modify the control system configuration through a simple operation involving only the two tasks among which the switch occurs. Similarly, the definition of the estimate DEMUX tasks refers to the capability, for the proposed architecture, to dynamically modify the connections among the tasks. In fact, tasks are statically linked to their input and output variables; however, on the estimation leg, it is possible to assign to a variable, a value chosen among a set of

alternatives, thus overcoming the limitation (see Figure 3).

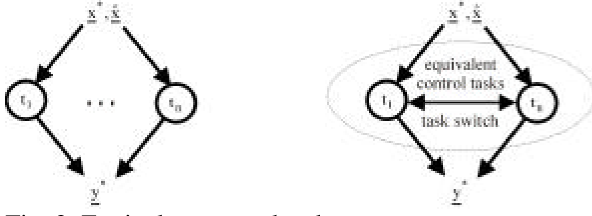

Fig. 2. Equivalent control tasks.

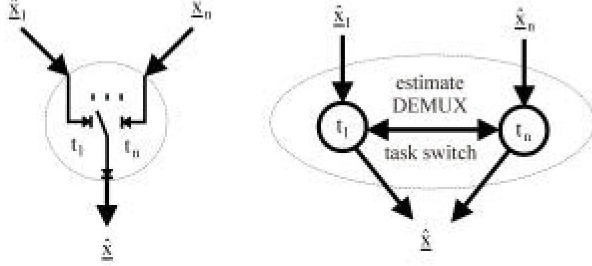

Fig. 3. Switch among estimation variables.

The above-described components of the execution level are represented by means of Petri nets. A standard task *t* is represented by means of the Petri net (*PNT*) depicted in the following picture.

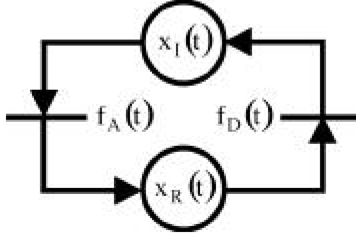

Fig. 4. Petri net representing a single task.

The net has two places $\{x_I(t), x_R(t)\}$, where $x_R(t)$ is the place corresponding to task *t* running, whereas $x_I(t)$ is the one corresponding to task *t* idle. That is, each of the two places is marked when and only when the corresponding task in the execution level is running or idle, respectively. The two transitions $\{f_A(t), f_D(t)\}$ correspond to the operations of activating or deactivating the task. Finally, using the matrix representation, the dynamics of the Petri net related to this task is described by:

$$\begin{bmatrix} x_I(t) \\ x_R(t) \end{bmatrix}_{k+1} = \begin{bmatrix} x_I(t) \\ x_R(t) \end{bmatrix}_k + \begin{bmatrix} -1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} f_A(t) \\ f_D(t) \end{bmatrix}_k .$$

Both the DEMUX task and the case of equivalent control tasks share the same Petri net representation, as they are both reconducible to a set of concurrent conflicting tasks. Hence, the Petri net (*PNST*) in Figure 5 is used to describe both structures. The net has *n+1* places $\{x_I(\mathbf{a}), x_R(t_1), \ldots, x_R(t_n)\}$ (where *n* is the number of equivalent control tasks or of possible switch positions). $x_R(t_i)$ is the place corresponding to task t_i being running (in the case of estimate demux task, it corresponds to the switch being in position t_i); $x_I(\mathbf{a})$ is the place corresponding to task α idle, i.e. all the tasks $\{t_1, \ldots, t_n\}$ idle (or switch in an opened position, for the case of the estimate demux tasks).

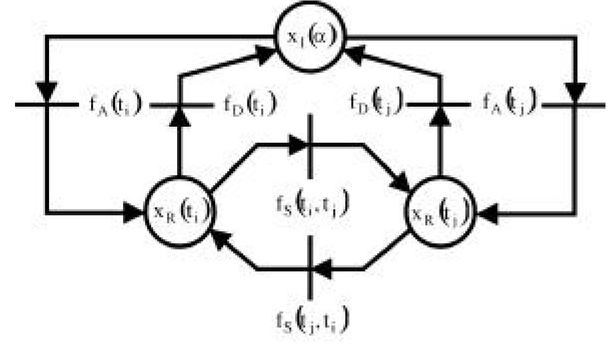

Fig. 5. Petri net representing two equivalent control tasks, or a switch between two estimation variables.

The *2n+n(n-1)* transitions are:

$$\{f_A(t_1), f_D(t_1), \ldots, f_A(t_n), f_D(t_n), f_S(t_1, t_2), \ldots$$
$$\ldots, f_S(t_i, t_j), \ldots, f_S(t_n, t_{n-1})\}$$

where, $f_A(t_i)$ is the transition corresponding to the activation of task t_i; $f_D(t_i)$ is the one corresponding to the deactivation of task t_i; finally, $f_S(t_i, t_j)$ is the transition corresponding to switch from task t_i to task t_j. The matrix notation provides the following description of the net:

$$\begin{bmatrix} x_I(\mathbf{a}) \\ x_R(t_i) \\ x_R(t_j) \end{bmatrix}_{k+1} = \begin{bmatrix} x_I(\mathbf{a}) \\ x_R(t_i) \\ x_R(t_j) \end{bmatrix}_k + D_{\mathbf{a}} \begin{bmatrix} f_A(t_i) \\ f_D(t_i) \\ f_A(t_j) \\ f_D(t_j) \\ f_S(t_i, t_j) \\ f_S(t_j, t_i) \end{bmatrix}_k ,$$

where:

$$D_{\mathbf{a}} = \begin{bmatrix} -1 & 1 & -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 & 1 & -1 \end{bmatrix} .$$

By superposing the above-described structures, it is possible to obtain a raw Petri net representation of the execution level. This net is the basis for the application of the supervision method proposed by (Yamalidou, *et al.*, 1996). Using the matrix notation, the raw Petri net behaviour is described by:

$$\underline{x}_{k+1} = \underline{x}_k + D_r \underline{f}_k .$$

Let denote with m_r and *n* the total number of places and transitions respectively. Moreover, let define the following symbols, that will turn out to be useful in the remainder of the paper:

$\mathrm{R}(t)$: sum of the tokens in the set of places corresponding to task t running, i.e., $R(t) \equiv x_R(t)$

$\overline{\mathrm{R}}(t)$: sum of the tokens in the set of places corresponding to task t not running, i.e.,

$$\overline{R}(t_i) \equiv \begin{cases} x_I(t_i), & \text{if } t_i \in PNT \\ x_I(\mathbf{a}) + \sum_{j \in [1,n], j \neq i} x_R(t_j), & \text{if } t_i \in PNST \end{cases}$$

3.1 Task connection 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 knows which tasks will produce/consume its input/output variables, and suitable task activation, deactivation, and switching 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.

(R1) *no concurrent writing*

$$\forall v \in V,\ \sum_{t: v \in O(t)} R(t) \leq 1$$

(R2) *no concurrent tracking*

$$\forall v \in CV,\ \sum_{t: v \in CI(t)} R(t) \leq 1$$

(R3) *complete tracking of written control variables*

$$\forall v \in CV, \forall t \in CT : v \in CO(t),\ \overline{R}(t) + \sum_{\mathbf{t}: v \in CI(\mathbf{t})} R(\mathbf{t}) \geq 1$$

(R4) *complete writing of consumed estimation variables*

$$\forall v \in EV, \forall t \in T : v \in EI(t),\ \overline{R}(t) + \sum_{\mathbf{t}: v \in EO(\mathbf{t})} R(\mathbf{t}) \geq 1$$

Rule (R1) states that, at any time instant, there can not be two or more running tasks having the same output variable. Rule (R2) ensures that there are no control tasks tracking the same reference value at the same time. Roughly speaking, this rule establishes the uniqueness of the control strategy. Rules (R3) and (R4) establish the "from bottom upwards" activation of the control system. (R3) affirms that generated reference values must be tracked, while rule (R4) states that each input estimation variable of a running task must be the output of another (necessarily, estimation) running task. That is, estimation tasks must be activated from bottom up, and before the control tasks using their outputs.

These constraints can be enforced by means of the results from (Yamalidou, *et al.*, 1996). The method will not be furtherly described here, since a thorough investigation is already available in (Caccia, *et al.*, 2001), and the reader is referred to that paper for more details. For our purposes, it is sufficient to know that it is possible to obtain a Petri net embedding the constraint, that is described by:

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

The total number of places of the net is denoted by m, whereas the total number of transitions remains unchanged (i.e., n). The net $\{\underline{x}_c, D_c\}$ is called the controlling net, and m_c denotes the number of its places.

The execution control module embeds the Petri net representation of the execution level. During the on-line operations it checks every command received from the upper levels, and verifies its feasibility. When the command cannot be executed in the trivial way (that is, simply firing the corresponding transition in the Petri net), a valid firing sequence is searched for, by means of the algorithm illustrated in the next section. Moreover, the execution control module performs a continuous monitoring of the configuration of the execution level. In the case that a better feasible configuration is found, the control architecture is automatically reconfigured, as it is described in Section 5

4 PETRI NET RECONFIGURATION

The request from the upper levels of the control architecture to activate/deactivate a task can be mapped to the requirement of having a certain place of the net marked with one token. Basing on this observation, we define a *Goal* of the control architecture as the requirement to have a set of places of the net marked. It can be easily verified that there is not a unique marking satisfying the goal, in general. In some cases there can be multiple alternatives, in other cases the goal might be inadmissible. Moreover, the satisfaction of the goal can be subject to auxiliary constraints, such as the invariance of a part of the marking, that can correspond to keeping active some previous goal. Thus, the aim of the search algorithm is to find a suitable sequence of transition firings such that a set of places is marked, and another set of places does not change its marking. The second set might be empty in some cases.

An algorithm for fulfilling this requirement has been implemented, and it is based on the idea of backward analysing the net, starting from the places that have to be marked, and searching for the transitions that can turn out to be of some utility for marking these places. Thus, in some way, the method resembles the backward phase of the spreading methodologies (Bagchi, *et al.*, 2000).

Before detailing the algorithm, let's introduce some further notation. Transition j is a predecessor of place i iff $D(i, j) \equiv 1$; place i is a predecessor of transition j iff $D(i, j) \equiv -1$. Symbol *uX(i)* denotes the utility of the i_{th} place, with $i=1..m$. Analogously, symbol *uf(j)* denotes the utility of the j_{th} transition, with $j=1..n$. Moreover, g represents the index of the goal place; finally, E_G is the coefficient of the utility injected by the goal at each iteration. Furthermore, $\boldsymbol{z}_P$ and $\boldsymbol{z}_T$ are the values assigned to each place or transition, respectively. These values are established during the design phase of the control architecture.

Now, it is possible to illustrate a sketch of the algorithm.

while <X(g) ¹ 1> at each step:

utility injection by goal:

$$uX(g) = uX(g) + E_G \boldsymbol{z}_P(g)$$

backward spreading of place utilities, i.e., activation of predecessor transitions:

defined $\delta uf(i,j)$ as the utility spread backward by place *i* to transition *j*, if transition *j* is a predecessor of place *i* and *j* is activable

$$\mathbf{d}uf(i,j) = uX(i)\frac{z_T(j)}{\sum\limits_{k:[D(i,k)=1]\wedge[k\ \ activable]} z_T(k)}$$

$$\mathbf{d}uf(j) = \sum_{i=1}^{m} \mathbf{d}uf(i,j)$$

backward spreading of transition utilities, i.e., activation of predecessor controlling places:

defined $\mathbf{d}uX(i,j)$ as the utility spread backward by transition *j* to place *i* if place *i* is a predecessor of transition *j*

$$\mathbf{d}uX(i,j) = uf(j)\frac{z_P(i)}{\sum\limits_{k:[D(k,j)=-1]} z(k)}, i \in [m_{r+1}, m]$$

$$\mathbf{d}uX(i) = \sum_{j=1}^{n} \mathbf{d}uX(i,j), i \in [m_{r+1}, m]$$

backward search stop conditions

In order to avoid the propagation the backward search through the loops in the PN, some simple rules are applied:

- transitions that are predecessors of marked places are inhibited
- transitions that are successors of useful places are inhibited (these transitions would eliminate a desired token from the net)

Note that the termination of the algorithm is also guaranteed by the fact that backward utility is spread only to places belonging to the controlling net; this is because the controlling places are the only ones embedding the information about the conflicts and dependencies among the tasks, hence, they are the only to be considered when looking for the feasibility of a sequence of tasks activations/deactivations.

5. PETRI NET-BASED OPTIMIZATION

Besides the reconfiguration that occurs upon a command receipt, the execution control module has the capability to perform a continuous monitoring of the state of the control architecture, and to reconfigure it, if a better configuration is found. In fact, it is possible to assign a value to the state (running/idle) of each task, and it is represented by the function z_P. On the basis of this information a monitor can be built that analyzes the state of the Petri net and computes the overall value associated to the current configuration. Then, continuously, it verifies if there exist transition firings that can lead the net (and hence, the execution level) to a configuration with an higher total value. Not all transitions are considered for this purpose, but only the ones related to the estimation tasks or to the switches between control tasks. Practically, this means that the estimation leg of the control architecture is kept active whenever possible; moreover, the proper estimation variable and control functions are automatically chosen by the execution control module.

6. SIMULATION RESULTS

A part of the control system of an underwater remotely operated vehicle has been simulated (see Figure 6). The tasks involved in the example are described in the following. *altimeterBow*, *altimeterStern*, and *depthMeter* are tasks interfacing with the related physical devices; correspondingly, *verticalPropulsion* is the control task that provides the interface to the actuators. *heaveEKF*, *hFilterBS*, and *hFilterB* are estimation tasks devoted to computing the estimates of the vertical speed and position (*heaveEKF*) and of the altitude from the sea-bed (*hFilterB* and *hFilterBS*). Note that two different estimates of the altitude are available, and a selector is provided to switch between them at any time. *autoDepth*, *autoHeave*, *autoAltitudeP*, and *autoAltitudePI* are standard controllers. Finally, *zRefOp* and *hRefOp* are the tasks receiving the depth and altitude reference values from the upper levels of the control architecture, or from the operator.

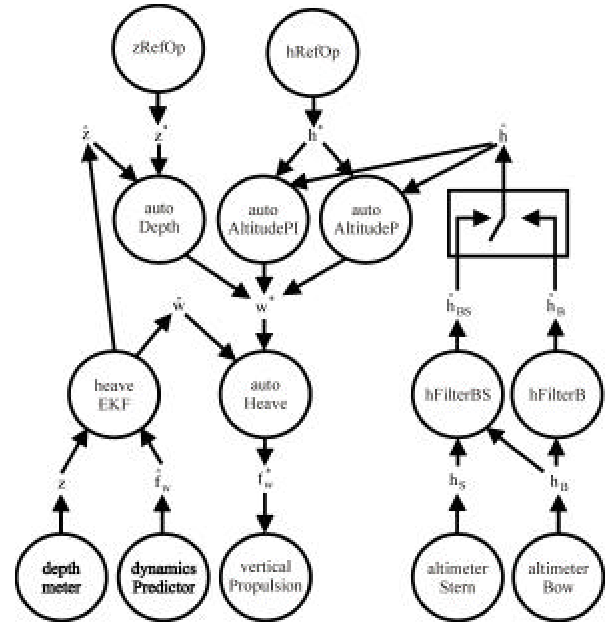

Fig. 6. Sketch of Romeo's control system.

Table 1 summarizes the behaviour of the control system. The clock column, as well as the references to time steps in the following paragraph, are to be interpreted as occurrences of events. At time step 2 a command is received to activate task *zRefOp*, the following operations (3-11) represent the correct sequence of actions to be performed in order to satisfy the request. Next, the system autonomously evolves (steps 12 to 16) activating a set of estimation tasks. The next command, received at time instant 21, is accomplished by deactivating the tasks *zRefOp* and *autoDepth*, which are in conflict with the desired tasks *autoAltitudePI* and *hRefOp*. Note that the

activation of the required estimation tasks is not necessary, since it was done during the autonomous evolution of the system.

Table 1 Simulation results.

clock	description
2	**command**: activate task zRefOp
5	activate verticalPropulsion
6	activate depthmeter
7	activate dynamicsSensor
8	activate heaveEKF
9	activate autoHeave
10	activate autoDepth
11	activate zRefOp
	List of active tasks:
	'zRefOp' 'autoDepth' 'autoHeave'
	'verticalPropulsion' 'heaveEKF'
	'depthmeter' 'dynamicsSensor'
12	activate altimeterS
13	activate altimeterB
14	activate hFilterBS
15	*hhat ← hhatBS*
16	activate hFilterB
	List of active tasks:
	'zRefOp' 'autoDepth' 'autoHeave'
	'verticalPropulsion' 'heaveEKF'
	'hFilterBS' 'hFilterB'
	'depthmeter' 'dynamicsSensor'
	'altimeterS' 'altimeterB'
	'hhat' ← 'hhatBS'
21	**command**: activate task hRefOp
24	deactivate zRefOp
25	deactivate autoDepth
26	activate autoAltitudePI
27	activate hRefOp
	List of active tasks:
	'hRefOp' 'autoHeave'
	'verticalPropulsion' 'heaveEKF'
	'hFilterBS' 'hFilterB'
	'depthmeter' 'dynamicsSensor'
	'altimeterS' 'altimeterB'
	'autoAltitudePI'
	'hhat' ← 'hhatBS'
35	**emergency**
	deactivate task altimeterS
37	hhat variable switch hhatBS hhatB
38	deactivate hFilterBS
39	deactivate altimeterS
	List of active tasks:
	'hRefOp' 'autoHeave'
	'verticalPropulsion' 'heaveEKF'
	'hFilterB' 'depthmeter'
	'dynamicsSensor' 'altimeterB'
	'autoAltitudePI'
	'hhat' ← 'hhatB'
44	**end emergency**
	activate task altimeterS
44	activate altimeterS
45	*activate hFilterBS*
46	*hhat ← hhatBS (~~hhatB~~)*
	List of active tasks:
	'hRefOp' 'autoHeave'
	'verticalPropulsion' 'heaveEKF'
	'hFilterBS' 'hFilterB'
	'depthmeter' 'dynamicsSensor'
	'altimeterS' 'altimeterB'
	'autoAltitudePI'
	'hhat' ← 'hhatBS'

Later on (time step 35), a temporary malfunctioning is detected for the *altimeterStern*, hence, the related task is switched off. The system is reconfigured by simply deactivating the corresponding estimator, and switching to the alternative altitude estimate (*hHatB*). Finally, at time step 44, the altimeter is switched on again (the temporary problem is supposed to be terminated). Consequently (time steps 44 to 46), the system is automatically reconfigured to use this sensor.

7. CONCLUSIONS

A methodology to exploit the information embedded in the I/O relationships among tasks, in order to control their execution, has been presented. The representation of the execution level as a Petri net allows the automatic reconfiguration of the control architecture. An algorithm for finding the proper sequence of operations to switch between various configurations has been introduced, and it is based on the analysis of the Petri net. Finally, it has been showed how to design a monitor that maintains the best possible configuration of the control architecture, given the current availability of hardware and software components.

References

Alami, R., Chatila, R., Fleury, S., Ghallab, M. and Ingrand, F. (1998). An Architecture for Autonomy. *The International Journal of Robotics Research*. **17**, pp. 315-337.

Antsaklis, P.J. and Passino, K.M. (1993) Introduction to intelligent control systems with high degrees of autonomy. In: *An introduction to intelligent and autonomous control.* Kluwer Academic Publishers, Boston, USA.

Bagchi, S., Biswas, G. and Kawamura, K. (2000). Task planning under uncertainty using a spreading activation network. *IEEE Transactions on Systems, man, and Cybernetics – Part A: Systems and Humans*, **30**, pp. 639-650.

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

Fierro, R.F. and Lewis, L. (1997). A framework for hybrid control design. *IEEE Transactions on Systems, man, and Cybernetics – Part A: Systems and Humans*, **27**, pp. 765-773.

Sheridan, T.B. (1998). Telerobotics. *Automatica.* **25**, pp. 487-507.

Valavanis, K.P. and Saridis, G.N. (1992). *Intelligent Robotic Systems: Theory, Design and Applications*. Kluwer Academic Publishers, USA.

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